

**Linear algebra, signal processing, and  
wavelets. A unified approach.  
Python version**

Øyvind Ryan

Feb 12, 2017

# Contents

<b>1</b>	<b>Sound and Fourier series</b>	<b>1</b>
1.1	Sound and digital sound: Loudness and frequency . . . . .	3
1.1.1	The frequency of a sound . . . . .	5
1.1.2	Working with digital sound on a computer . . . . .	6
1.2	Fourier series: Basic concepts . . . . .	10
1.2.1	Fourier series for symmetric and antisymmetric functions . . . . .	19
1.3	Complex Fourier series . . . . .	21
1.4	Some properties of Fourier series . . . . .	28
1.4.1	Rate of convergence for Fourier series . . . . .	31
1.4.2	Differentiating Fourier series . . . . .	32
1.5	Operations on sound: filters . . . . .	36
1.6	Convergence of Fourier series* . . . . .	38
1.6.1	Interpretation in terms of filters . . . . .	42
1.7	The MP3 standard . . . . .	45
1.8	Summary . . . . .	47
<b>2</b>	<b>Digital sound and Discrete Fourier analysis</b>	<b>49</b>
2.1	Discrete Fourier analysis and the discrete Fourier transform . . . . .	49
2.1.1	Properties of the DFT . . . . .	55
2.2	Connection between the DFT and Fourier series. Sampling and the sampling theorem . . . . .	59
2.2.1	Change in frequency representation when windowing a signal . . . . .	65
2.3	The Fast Fourier Transform (FFT) . . . . .	69
2.3.1	Reduction in the number of arithmetic operations . . . . .	74
2.3.2	The FFT when $N$ is not a power of 2 . . . . .	77
2.4	Summary . . . . .	85
<b>3</b>	<b>Operations on digital sound: digital filters</b>	<b>87</b>
3.1	Matrix representations of filters . . . . .	87
3.1.1	Convolution . . . . .	91
3.2	Formal definition of filters and the vector frequency response . . . . .	95
3.2.1	Time delay . . . . .	96
3.2.2	Using digital filters to approximate filters . . . . .	100
3.3	The continuous frequency response and properties . . . . .	103

3.3.1	Windowing operations . . . . .	107
3.4	Some examples of filters . . . . .	111
3.5	More general filters . . . . .	125
3.6	Implementation of filters . . . . .	128
3.6.1	Implementation of filters using the DFT . . . . .	128
3.6.2	Factoring a filter . . . . .	128
3.7	Summary . . . . .	129
<b>4</b>	<b>Symmetric filters and the DCT</b>	<b>132</b>
4.1	Symmetric vectors and the DCT . . . . .	133
4.2	Improvements using the DCT for interpolation . . . . .	148
4.2.1	Implementations of symmetric filters . . . . .	149
4.3	Efficient implementations of the DCT . . . . .	151
4.3.1	Efficient implementations of the IDCT . . . . .	154
4.3.2	Reduction in the number of arithmetic operations . . . . .	155
4.4	Summary . . . . .	159
<b>5</b>	<b>Motivation for wavelets and some simple examples</b>	<b>161</b>
5.1	Why wavelets? . . . . .	162
5.2	A wavelet based on piecewise constant functions . . . . .	163
5.2.1	Function approximation property . . . . .	167
5.2.2	Detail spaces and wavelets . . . . .	168
5.3	Implementation of the DWT and examples . . . . .	178
5.4	A wavelet based on piecewise linear functions . . . . .	187
5.4.1	Detail spaces and wavelets . . . . .	190
5.5	Alternative wavelet based on piecewise linear functions . . . . .	197
5.6	Multiresolution analysis: A generalization . . . . .	205
5.7	Summary . . . . .	210
<b>6</b>	<b>The filter representation of wavelets</b>	<b>212</b>
6.1	The filters of a wavelet transformation . . . . .	213
6.1.1	The dual filter bank transform and the <code>dua1</code> parameter . . . . .	220
6.1.2	The support of the scaling function and the mother wavelet . . . . .	222
6.1.3	Symmetric extensions and the <code>bd_mode</code> parameter. . . . .	224
6.2	Properties of the filter bank transforms of a wavelet . . . . .	233
6.3	A generalization of the filter representation, and its use in audio coding . . . . .	241
6.3.1	Forward filter bank transform in the MP3 standard . . . . .	243
6.3.2	Reverse filter bank transform in the MP3 standard . . . . .	246
6.4	Summary . . . . .	250
<b>7</b>	<b>Constructing interesting wavelets</b>	<b>253</b>
7.1	From filters to scaling functions and mother wavelets . . . . .	253
7.2	Vanishing moments . . . . .	262
7.3	Characterization of wavelets w.r.t. number of vanishing moments . . . . .	265
7.3.1	Symmetric filters . . . . .	266

7.3.2	Orthonormal wavelets . . . . .	269
7.3.3	The proof of Bezouts theorem . . . . .	272
7.4	A design strategy suitable for lossless compression . . . . .	273
7.4.1	The Spline 5/3 wavelet . . . . .	274
7.5	A design strategy suitable for lossy compression . . . . .	276
7.6	Orthonormal wavelets . . . . .	279
7.7	Summary . . . . .	282
<b>8</b>	<b>The polyphase representation and wavelets</b>	<b>284</b>
8.1	The polyphase representation and the lifting factorization . . . . .	285
8.1.1	Reduction in the number of arithmetic operations . . . . .	291
8.2	Examples of lifting factorizations . . . . .	294
8.2.1	The piecewise linear wavelet . . . . .	295
8.2.2	The Spline 5/3 wavelet . . . . .	295
8.2.3	The CDF 9/7 wavelet . . . . .	296
8.2.4	Orthonormal wavelets . . . . .	297
8.3	Cosine-modulated filter banks and the MP3 standard . . . . .	302
8.3.1	Polyphase representations of the filter bank transforms . . . . .	303
8.3.2	The prototype filters . . . . .	308
8.3.3	Perfect reconstruction . . . . .	310
8.4	Summary . . . . .	315
<b>9</b>	<b>Digital images</b>	<b>316</b>
9.1	What is an image? . . . . .	317
9.2	Some simple operations on images with Python . . . . .	321
9.3	Filter-based operations on images . . . . .	329
9.3.1	Tensor product notation for operations on images . . . . .	331
9.4	Change of coordinates in tensor products . . . . .	345
9.5	Summary . . . . .	350
<b>10</b>	<b>Using tensor products to apply wavelets to images</b>	<b>354</b>
10.1	Tensor product of function spaces . . . . .	354
10.1.1	Tensor products of polynomials . . . . .	355
10.1.2	Tensor products of Fourier spaces . . . . .	355
10.2	Tensor product of function spaces in a wavelet setting . . . . .	357
10.2.1	Interpretation . . . . .	361
10.3	Experiments with images using wavelets . . . . .	364
10.4	An application to the FBI standard for compression of fingerprint images . . . . .	376
10.5	Summary . . . . .	380
<b>11</b>	<b>The basics and applications</b>	<b>382</b>
11.1	The basic concepts . . . . .	383
11.2	Some applications . . . . .	384
11.3	Multivariate calculus and linear algebra . . . . .	390



<b>12 A crash course in convexity</b>	<b>396</b>
12.1 Convex sets . . . . .	396
12.2 Convex functions . . . . .	397
12.3 Properties of convex functions . . . . .	399
<b>13 Nonlinear equations</b>	<b>407</b>
13.1 Equations and fixed points . . . . .	407
13.2 Newton's method . . . . .	409
<b>14 Unconstrained optimization</b>	<b>415</b>
14.1 Optimality conditions . . . . .	415
14.2 Methods . . . . .	419
<b>15 Constrained optimization - theory</b>	<b>432</b>
15.1 Equality constraints and the Lagrangian . . . . .	432
15.2 Inequality constraints and KKT . . . . .	437
15.3 Convex optimization . . . . .	444
15.3.1 A useful theorem on convex optimization . . . . .	447
<b>16 Constrained optimization - methods</b>	<b>452</b>
16.1 Equality constraints . . . . .	452
16.2 Inequality constraints . . . . .	453
<b>A Basic Linear Algebra</b>	<b>463</b>
A.1 Matrices . . . . .	463
A.2 Vector spaces . . . . .	463
A.3 Inner products and orthogonality . . . . .	464
A.4 Coordinates and change of coordinates . . . . .	464
A.5 Eigenvectors and eigenvalues . . . . .	465
A.6 Diagonalization . . . . .	465
<b>B Signal processing and linear algebra: a translation guide</b>	<b>466</b>
B.1 Complex numbers . . . . .	466
B.2 Functions . . . . .	466
B.3 Vectors . . . . .	467
B.4 Inner products and orthogonality . . . . .	467
B.5 Matrices and filters . . . . .	467
B.6 Convolution . . . . .	468
B.7 Polyphase factorizations and lifting . . . . .	468
B.8 Transforms in general . . . . .	469
B.9 Perfect reconstruction systems . . . . .	469
B.10 Z-transform and frequency response . . . . .	470
<b>Nomenclature</b>	<b>471</b>
<b>Bibliography</b>	<b>473</b>

*CONTENTS*

vi

**Index**

**477**

# List of Examples and Exercises

Example 1.1: Listen to different channels . . . . .	7
Example 1.2: Playing the sound backwards . . . . .	7
Example 1.3: Playing pure tones. . . . .	7
Example 1.4: The square wave . . . . .	8
Example 1.5: The triangle wave . . . . .	9
Exercise 1.6: The Krakatoa explosion . . . . .	9
Exercise 1.7: Sum of two pure tones . . . . .	9
Exercise 1.8: Sum of two pure tones . . . . .	9
Exercise 1.9: Playing with different sample rates . . . . .	9
Exercise 1.10: Play sound with added noise . . . . .	10
Exercise 1.11: Playing the triangle wave . . . . .	10
Example 1.12: Fourier coefficients of the square wave . . . . .	14
Example 1.13: Fourier coefficients of the triangle wave . . . . .	17
Example 1.14: Fourier coefficients of a simple function . . . . .	19
Exercise 1.15: Shifting the Fourier basis vectors . . . . .	20
Exercise 1.16: Playing the Fourier series of the triangle wave . . . . .	20
Exercise 1.17: Riemann-integrable functions which are not square-integrable	20
Exercise 1.18: When are Fourier spaces included in each other? . . . . .	20
Exercise 1.19: antisymmetric functions are sine-series . . . . .	20
Exercise 1.20: More connections between symmetric-/antisymmetric func- tions and sine-/cosine series . . . . .	21
Exercise 1.21: Fourier series for low-degree polynomials . . . . .	21
Exercise 1.22: Fourier series for polynomials . . . . .	21
Exercise 1.23: Fourier series of a given polynomial . . . . .	21
Example 1.24: Complex Fourier coefficients of a simple function . . . . .	25
Example 1.25: Complex Fourier coefficients of composite function . . . . .	25
Example 1.26: Complex Fourier coefficients of $f(t) = \cos^3(2\pi t/T)$ . . . . .	27
Exercise 1.27: Orthonormality of Complex Fourier basis . . . . .	27
Exercise 1.28: Complex Fourier series of $f(t) = \sin^2(2\pi t/T)$ . . . . .	27
Exercise 1.29: Complex Fourier series of polynomials . . . . .	27
Exercise 1.30: Complex Fourier series and Pascals triangle . . . . .	27
Exercise 1.31: Complex Fourier coefficients of the square wave . . . . .	28

Exercise 1.32: Complex Fourier coefficients of the triangle wave . . . . .	28
Exercise 1.33: Complex Fourier coefficients of low-degree polynomials . . .	28
Exercise 1.34: Complex Fourier coefficients for symmetric and antisym- metric functions . . . . .	28
Example 1.35: Periodic extension . . . . .	35
Exercise 1.36: Fourier series of a delayed square wave . . . . .	35
Exercise 1.37: Find function from its Fourier series . . . . .	36
Exercise 1.38: Relation between complex Fourier coefficients of $f$ and cosine-coefficients of $\check{f}$ . . . . .	36
Exercise 1.39: Filters preserve sine- and cosine-series . . . . .	38
Exercise 1.40: Approximation in norm with continuous functions . . . . .	44
Exercise 1.41: The Dirichlet kernel . . . . .	44
Exercise 1.42: The Fejer summability kernel . . . . .	44
Example 2.1: DFT of a cosine . . . . .	53
Example 2.2: DFT on a square wave . . . . .	53
Example 2.3: Computing the DFT by hand . . . . .	54
Example 2.4: Direct implementation of the DFT . . . . .	55
Example 2.5: Computing the DFT when multiplying with a complex exponential . . . . .	57
Exercise 2.6: Computing the DFT by hand . . . . .	57
Exercise 2.7: Exact form of low-order DFT matrix . . . . .	57
Exercise 2.8: DFT of a delayed vector . . . . .	57
Exercise 2.9: Using symmetry property . . . . .	57
Exercise 2.10: DFT of $\cos^2(2\pi k/N)$ . . . . .	57
Exercise 2.11: DFT of $e^{k\mathbf{x}}$ . . . . .	57
Exercise 2.12: Rewrite a complex DFT as real DFT's . . . . .	58
Exercise 2.13: DFT implementation . . . . .	58
Exercise 2.14: Symmetry . . . . .	58
Exercise 2.15: DFT on complex and real data . . . . .	59
Example 2.16: Using the DFT to adjust frequencies in sound . . . . .	65
Example 2.17: Compression by zeroing out small DFT coefficients . . . . .	67
Example 2.18: Compression by quantizing DFT coefficients . . . . .	68
Exercise 2.19: Comment code . . . . .	69
Exercise 2.20: Which frequency is changed? . . . . .	69
Exercise 2.21: Implement interpolant . . . . .	69
Exercise 2.22: Extra results for the FFT when $N = N_1N_2$ . . . . .	79
Exercise 2.23: Extend implementation . . . . .	79
Exercise 2.24: Compare execution time . . . . .	80
Exercise 2.25: Combine two FFT's . . . . .	80
Exercise 2.26: FFT operation count . . . . .	80
Exercise 2.27: FFT algorithm adapted to real data . . . . .	81
Exercise 2.28: Non-recursive FFT algorithm . . . . .	81
Exercise 2.29: The Split-radix FFT algorithm . . . . .	82
Exercise 2.30: Bit-reversal . . . . .	84
Example 3.1: Finding the matrix elements from the filter coefficients . . .	90
Example 3.2: Finding the filter coefficients from the matrix . . . . .	91

Example 3.3: Writing down compact filter notation . . . . .	91
Exercise 3.4: Finding the filter coefficients and the matrix . . . . .	94
Exercise 3.5: Finding the filter coefficients from the matrix . . . . .	94
Exercise 3.6: Convolution and polynomials . . . . .	94
Exercise 3.7: Implementation of convolution . . . . .	94
Exercise 3.8: Filters with a different number of coefficients with positive and negative indices . . . . .	95
Exercise 3.9: Implementing filtering with convolution . . . . .	95
Example 3.10: Frequency response of a simple filter . . . . .	98
Example 3.11: Matrix form . . . . .	98
Example 3.12: Computing the output of a filter . . . . .	99
Exercise 3.13: Time reversal is not a filter . . . . .	102
Exercise 3.14: When is a filter symmetric? . . . . .	102
Exercise 3.15: Eigenvectors and eigenvalues . . . . .	102
Exercise 3.16: Composing filters . . . . .	103
Exercise 3.17: Keeping every second component . . . . .	103
Example 3.18: Plotting a simple frequency response . . . . .	106
Example 3.19: Computing a composite filter . . . . .	107
Exercise 3.20: Plotting a simple frequency response . . . . .	109
Exercise 3.21: Low-pass and high-pass filters . . . . .	109
Exercise 3.22: Circulant matrices . . . . .	110
Exercise 3.23: Composite filters . . . . .	110
Exercise 3.24: Maximum and minimum . . . . .	110
Exercise 3.25: Plotting a simple frequency response . . . . .	110
Exercise 3.26: Continuous- and vector frequency responses . . . . .	110
Exercise 3.27: Starting with circulant matrices . . . . .	111
Exercise 3.28: When the filter coefficients are powers . . . . .	111
Exercise 3.29: The Hanning window . . . . .	111
Example 3.30: Adding echo . . . . .	112
Example 3.31: Reducing the treble with moving average filters . . . . .	114
Example 3.32: Ideal low-pass filters . . . . .	115
Example 3.33: Dropping filter coefficients . . . . .	116
Example 3.34: Filters and the MP3 standard . . . . .	117
Example 3.35: Reducing the treble using Pascals triangle . . . . .	119
Example 3.36: Reducing the bass using Pascals triangle . . . . .	121
Exercise 3.37: Composing time delay filters . . . . .	122
Exercise 3.38: Adding echo filters . . . . .	122
Exercise 3.39: Reducing bass and treble . . . . .	123
Exercise 3.40: Constructing a high-pass filter . . . . .	123
Exercise 3.41: Combining low-pass and high-pass filters . . . . .	123
Exercise 3.42: Composing filters . . . . .	123
Exercise 3.43: Composing filters . . . . .	123
Exercise 3.44: Filters in the MP3 standard . . . . .	124
Exercise 3.45: Explain code . . . . .	124
Example 3.46: Moving average filter . . . . .	126
Exercise 3.47: A concrete IIR filter . . . . .	127

Exercise 3.48: Implementing the factorization . . . . .	129
Exercise 3.49: Factoring concrete filter . . . . .	129
Example 4.1: Computing lower order DCTs . . . . .	144
Exercise 4.2: Computing eigenvalues . . . . .	145
Exercise 4.3: Writing down lower order $S_r$ . . . . .	146
Exercise 4.4: Writing down lower order DCTs . . . . .	146
Exercise 4.5: DCT-IV . . . . .	146
Exercise 4.6: MDCT . . . . .	146
Exercise 4.7: Component expressions for a symmetric filter . . . . .	151
Exercise 4.8: Trick for reducing the number of multiplications with the DCT	156
Exercise 4.9: An efficient joint implementation of the DCT and the FFT .	157
Exercise 4.10: Implementation of the IFFT/IDCT . . . . .	158
Exercise 5.1: The vector of samples is the coordinate vector . . . . .	176
Exercise 5.2: Proposition 5.12 . . . . .	176
Exercise 5.3: Computing projections 1 . . . . .	176
Exercise 5.4: Computing projections 2 . . . . .	176
Exercise 5.5: Computing projections 3 . . . . .	177
Exercise 5.6: Finding the least squares error . . . . .	177
Exercise 5.7: Projecting on $W_0$ . . . . .	177
Exercise 5.8: When $N$ is odd . . . . .	177
Example 5.9: Computing the DWT by hand . . . . .	180
Example 5.10: DWT on sound . . . . .	181
Example 5.11: DWT on the samples of a mathematical function . . . . .	183
Example 5.12: Computing the wavelet coefficients . . . . .	184
Exercise 5.13: Implement IDWT for The Haar wavelet . . . . .	185
Exercise 5.14: Computing projections 4 . . . . .	185
Exercise 5.15: Scaling a function . . . . .	185
Exercise 5.16: Direct sums . . . . .	185
Exercise 5.17: Eigenvectors of direct sums . . . . .	185
Exercise 5.18: Invertibility of direct sums . . . . .	186
Exercise 5.19: Multiplying direct sums . . . . .	186
Exercise 5.20: Finding $N$ . . . . .	186
Exercise 5.21: Different DWTs for similar vectors . . . . .	186
Exercise 5.22: Construct a sound . . . . .	186
Exercise 5.23: Exact computation of wavelet coefficients 1 . . . . .	187
Exercise 5.24: Exact computation of wavelet coefficients 2 . . . . .	187
Exercise 5.25: Computing the DWT of a simple vector . . . . .	187
Exercise 5.26: The Haar wavelet when $N$ is odd . . . . .	187
Exercise 5.27: in-place DWT . . . . .	187
Example 5.28: DWT on sound . . . . .	194
Example 5.29: DWT on the samples of a mathematical function . . . . .	195
Exercise 5.30: The vector of samples is the coordinate vector 2 . . . . .	195
Exercise 5.31: Computing projections . . . . .	196
Exercise 5.32: Non-orthogonality for the piecewise linear wavelet . . . . .	196
Exercise 5.33: Implement elementary lifting steps of odd type . . . . .	197
Exercise 5.34: Wavelets based on polynomials . . . . .	197

Example 5.35: DWT on sound . . . . .	200
Example 5.36: DWT on the samples of a mathematical function . . . . .	201
Exercise 5.37: Implement elementary lifting steps of even type . . . . .	201
Exercise 5.38: Two vanishing moments . . . . .	202
Exercise 5.39: Implement finding $\psi$ with vanishing moments . . . . .	203
Exercise 5.40: $\psi$ for the Haar wavelet with two vanishing moments . . . . .	204
Exercise 5.41: More vanishing moments for the Haar wavelet . . . . .	204
Exercise 5.42: Listening experiments . . . . .	205
Example 5.43: Implementing the cascade algorithm . . . . .	210
Example 6.1: The Haar wavelet . . . . .	217
Example 6.2: Wavelet for piecewise linear functions . . . . .	218
Example 6.3: The alternative piecewise linear wavelet . . . . .	219
Example 6.4: Plotting the frequency responses . . . . .	222
Exercise 6.5: Implement the dual filter bank transforms . . . . .	226
Exercise 6.6: Transpose of the DWT and IDWT . . . . .	226
Exercise 6.7: Reduced matrices for elementary lifting . . . . .	227
Exercise 6.8: Prove expression for $S_r$ . . . . .	227
Exercise 6.9: Orthonormal basis for the symmetric extensions . . . . .	228
Exercise 6.10: Diagonalizing $S_r$ . . . . .	228
Exercise 6.11: Compute filters and frequency responses 1 . . . . .	230
Exercise 6.12: Symmetry of MRA matrices vs. symmetry of filters 1 . . . . .	230
Exercise 6.13: Symmetry of MRA matrices vs. symmetry of filters 2 . . . . .	230
Exercise 6.14: Finding $H_0, H_1$ from $H$ . . . . .	230
Exercise 6.15: Finding $G_0, G_1$ from $G$ . . . . .	231
Exercise 6.16: Finding $H$ from $H_0, H_1$ . . . . .	231
Exercise 6.17: Finding $G$ from $G_0, G_1$ . . . . .	231
Exercise 6.18: Computing by hand . . . . .	231
Exercise 6.19: Comment code . . . . .	231
Exercise 6.20: Computing filters and frequency responses . . . . .	232
Exercise 6.21: Computing filters and frequency responses 2 . . . . .	232
Exercise 6.22: Implementing with symmetric extension . . . . .	232
Exercise 6.23: Finding FIR filters . . . . .	240
Exercise 6.24: The Haar wavelet as an alternative QMF filter bank . . . . .	241
Exercise 6.25: Plotting frequency responses . . . . .	250
Exercise 6.26: Implementing forward and reverse filter bank transforms . . . . .	250
Exercise 7.1: Implementation of the cascade algorithm . . . . .	262
Exercise 7.2: Using the cascade algorithm . . . . .	262
Exercise 7.3: Compute filters . . . . .	273
Exercise 7.4: Viewing the frequency response . . . . .	275
Exercise 7.5: Wavelets based on higher degree polynomials . . . . .	276
Example 7.6: The CDF 9/7 wavelet . . . . .	276
Exercise 8.1: The frequency responses of the polyphase components . . . . .	292
Exercise 8.2: Finding new filters . . . . .	292
Exercise 8.3: Relating to the polyphase components . . . . .	293
Exercise 8.4: QMF filter banks . . . . .	293
Exercise 8.5: Alternative QMF filter banks . . . . .	294

Exercise 8.6: Alternative QMF filter banks with additional sign . . . . .	294
Example 8.7: Lifting factorization of the alternative piecewise linear wavelet	295
Exercise 8.8: Polyphase components for symmetric filters . . . . .	300
Exercise 8.9: Implementing kernels transformations using lifting . . . . .	300
Exercise 8.10: Lifting orthonormal wavelets . . . . .	300
Exercise 8.11: 4 vanishing moments . . . . .	301
Exercise 8.12: Wavelet based on piecewise quadratic scaling function . . .	302
Exercise 8.13: Run forward and reverse transform . . . . .	313
Exercise 8.14: Verify statement of filters . . . . .	313
Exercise 8.15: Lifting . . . . .	314
Example 9.1: Normalising the intensities . . . . .	322
Example 9.2: Extracting the different colors . . . . .	323
Example 9.3: Converting from color to grey-level . . . . .	324
Example 9.4: Computing the negative image . . . . .	326
Example 9.5: Increasing the contrast . . . . .	326
Exercise 9.6: Generate black and white images . . . . .	328
Exercise 9.7: Adjust contrast in images . . . . .	328
Exercise 9.8: Adjust contrast with another function . . . . .	328
Example 9.9: Smoothing an image . . . . .	333
Example 9.10: Edge detection . . . . .	335
Example 9.11: Second-order derivatives . . . . .	338
Example 9.12: Chess pattern image . . . . .	340
Exercise 9.13: Implement a tensor product . . . . .	341
Exercise 9.14: Generate images . . . . .	341
Exercise 9.15: Interpret tensor products . . . . .	341
Exercise 9.16: Computational molecule of moving average filter . . . . .	342
Exercise 9.17: Bilinearity of the tensor product . . . . .	342
Exercise 9.18: Attempt to write as tensor product . . . . .	342
Exercise 9.19: Computational molecules . . . . .	342
Exercise 9.20: Computational molecules 2 . . . . .	342
Exercise 9.21: Comment on code . . . . .	343
Exercise 9.22: Eigenvectors of tensor products . . . . .	343
Exercise 9.23: The Kronecker product . . . . .	343
Example 9.24: Change of coordinates with the DFT . . . . .	347
Example 9.25: Change of coordinates with the DCT . . . . .	348
Exercise 9.26: Implement DFT and DCT on blocks . . . . .	348
Exercise 9.27: Implement two-dimensional FFT and DCT . . . . .	349
Exercise 9.28: Zeroing out DCT coefficients . . . . .	350
Exercise 9.29: Comment code . . . . .	350
Example 10.1: Piecewise constant functions . . . . .	359
Example 10.2: Piecewise linear functions . . . . .	361
Example 10.3: Applying the Haar wavelet to a very simple example image	365
Example 10.4: Creating thumbnail images . . . . .	365
Example 10.5: Detail and low-resolution approximations for different wavelets	366
Example 10.6: The Spline 5/3 wavelet and removing bands in the detail spaces . . . . .	367



Exercise 10.7: Implement two-dimensional DWT . . . . .	369
Exercise 10.8: Comment code . . . . .	370
Exercise 10.9: Comment code . . . . .	372
Exercise 10.10: Experiments on a test image . . . . .	373
Exercise 10.11: Implement the fingerprint compression scheme . . . . .	380
Exercise 11.1: Solve . . . . .	393
Exercise 11.2: Solve . . . . .	393
Exercise 11.3: Solve . . . . .	394
Exercise 11.4: Solve . . . . .	394
Exercise 11.5: Solve . . . . .	394
Exercise 11.6: Level sets . . . . .	394
Exercise 11.7: Sub-level sets . . . . .	394
Exercise 11.8: Portfolio optimization . . . . .	394
Exercise 11.9: Solve . . . . .	394
Exercise 11.10: Solve . . . . .	395
Exercise 11.11: Solve . . . . .	395
Exercise 11.12: Solve . . . . .	395
Exercise 12.1: The intersection of convex sets is convex. . . . .	403
Exercise 12.2: Solve . . . . .	403
Exercise 12.3: The convexity of a product of functions. . . . .	404
Exercise 12.4: The convexity of the composition of functions. . . . .	404
Exercise 12.5: Solve . . . . .	404
Exercise 12.6: Solve . . . . .	404
Exercise 12.7: Solve . . . . .	405
Exercise 12.8: Solve . . . . .	405
Exercise 12.9: Solve . . . . .	405
Exercise 12.10: The set of convex combinations is convex . . . . .	405
Exercise 12.11: Solve . . . . .	405
Exercise 12.12: A convex function defined on a closed real interval attains its maximum in one of the end points. . . . .	405
Exercise 12.13: The maximum of convex functions is convex. . . . .	406
Exercise 12.14: Solve . . . . .	406
Exercise 12.15: The distance to a convex set is a convex function. . . . .	406
Exercise 13.1: Solve . . . . .	412
Exercise 13.2: Solve . . . . .	413
Exercise 13.3: Solve . . . . .	413
Exercise 13.4: Solve . . . . .	413
Exercise 13.5: Solve . . . . .	413
Exercise 13.6: Solve . . . . .	413
Exercise 13.7: Broyden's method . . . . .	413
Exercise 13.8: Solve . . . . .	414
Exercise 13.9: Solve . . . . .	414
Exercise 13.10: Solve . . . . .	414
Exercise 14.1: Solve . . . . .	428
Exercise 14.2: Solve . . . . .	428
Exercise 14.3: Solve . . . . .	428

Exercise 14.4: Solve . . . . .	428
Exercise 14.5: Solve . . . . .	428
Exercise 14.6: Solve . . . . .	429
Exercise 14.7: When steepest descent finds the minimum in one step . . .	429
Exercise 14.8: Solve . . . . .	429
Exercise 14.9: Solve . . . . .	429
Exercise 14.10: Solve . . . . .	429
Exercise 14.11: Solve . . . . .	430
Exercise 14.12: Solve . . . . .	430
Exercise 14.13: Solve . . . . .	430
Example 15.1: A simple optimization problem . . . . .	439
Example 15.2: a one-variable problem . . . . .	442
Example 15.3: a multi-variable problem . . . . .	442
Example 15.4: Quadratic optimization problem with linear equality con- straints . . . . .	443
Example 15.5: Extension . . . . .	443
Example 15.6: Linear optimization . . . . .	444
Example 15.7: Comparing the primal and the dual problem . . . . .	447
Exercise 15.8: Find min . . . . .	448
Exercise 15.9: Find min . . . . .	448
Exercise 15.10: Find min . . . . .	449
Exercise 15.11: Find min . . . . .	449
Exercise 15.12: Find min . . . . .	449
Exercise 15.13: Find min . . . . .	449
Exercise 15.14: Find min . . . . .	450
Exercise 15.15: Find min . . . . .	450
Exercise 15.16: Find min . . . . .	450
Exercise 15.17: Find min . . . . .	450
Exercise 15.18: Find min . . . . .	450
Exercise 15.19: Find min . . . . .	450
Exercise 15.20: Find min . . . . .	450
Exercise 15.21: Find min . . . . .	451
Exercise 15.22: Find min . . . . .	451
Exercise 15.23: Find min . . . . .	451
Exercise 15.24: Find min . . . . .	451
Exercise 15.25: Find min . . . . .	451
Example 16.1: Numeric test of the internal-point barrier method . . . . .	459
Example 16.2: Analytic test of the internal-point barrier method . . . . .	460
Exercise 16.3: Solve . . . . .	461
Exercise 16.4: Solve . . . . .	461
Exercise 16.5: Solve . . . . .	462
Exercise 16.6: Solve . . . . .	462
Exercise 16.7: Solve . . . . .	462
Exercise 16.8: Solve . . . . .	462
Exercise 16.9: Solve . . . . .	462

# Chapter 1

## Sound and Fourier series

A major part of the information we receive and perceive every day is in the form of audio. Most sounds are transferred directly from the source to our ears, like when we have a face to face conversation with someone or listen to the sounds in a forest or a street. However, a considerable part of the sounds are generated by loudspeakers in various kinds of audio machines like cell phones, digital audio players, home cinemas, radios, television sets and so on. The sounds produced by these machines are either generated from information stored inside, or electromagnetic waves are picked up by an antenna, processed, and then converted to sound.

What we perceive as sound corresponds to the physical phenomenon of slight variations in air pressure near our ears. Air pressure is measured by the SI-unit Pa (Pascal) which is equivalent to  $N/m^2$  (force / area). In other words, 1 Pa corresponds to the force exerted on an area of  $1 m^2$  by the air column above this area. Larger variations mean louder sounds, while faster variations correspond to sounds with a higher pitch.

**Observation 1.1.** *Continuous Sound.*

A sound can be represented as a function, corresponding to air pressure measured over time. When a function represents a sound, it is often referred to as a *continuous sound*.

Continuous sounds are defined for all time instances. On computers and various kinds of media players, however, the sound is *digital*, i.e. it is represented by a large number of function values, stored in a suitable number format. Such digital sound is easier to manipulate and process on a computer.

**Observation 1.2.** *Digital sound.*

A *digital sound* is a sequence  $\mathbf{x} = \{x_i\}_{i=0}^{N-1}$  that corresponds to measurements of a continuous sound  $f$ , recorded at a fixed rate of  $f_s$  (the sampling frequency or *sample rate*) measurements per second, i.e.,

$$x_k = f(k/f_s), \quad \text{for } k = 0, 1, \dots, N-1.$$

Note that the indexing convention for digital sound is not standard in mathematics, where vector indices start at 1. The components in digital sound are often referred to as *samples*, the time between successive samples is called the *sampling period*, denoted  $T_s$ . and measuring the sound is also referred to as sampling the sound.

The quality of digital sound is often measured by the *bit rate* (number of bits per second), i.e. the product of the sampling rate and the number of bits (binary digits) used to store each sample. Both the sample rate and the number format influence the quality of the resulting sound. These are encapsulated in *digital sound formats*. A couple of them are described below.

**Telephony.** For telephony it is common to sample the sound 8000 times per second and represent each sample value as a 13-bit integer. These integers are then converted to a kind of 8-bit floating-point format with a 4-bit significand. Telephony therefore generates a bit rate of 64 000 bits per second, i.e. 64 kb/s.

**The CD-format.** In the classical CD-format the audio signal is sampled 44 100 times per second and the samples stored as 16-bit integers. The value 44 100 for the sampling rate is not coincidental, and we will return to this shortly. 16-bit integers work well for music with a reasonably uniform dynamic range, but is problematic when the range varies. Suppose for example that a piece of music has a very loud passage. In this passage the samples will typically make use of almost the full range of integer values, from  $-2^{15} - 1$  to  $2^{15}$ . When the music enters a more quiet passage the sample values will necessarily become much smaller and perhaps only vary in the range  $-1000$  to  $1000$ , say. Since  $2^{10} = 1024$  this means that in the quiet passage the music would only be represented with 10-bit samples. This problem can be avoided by using a floating-point format instead, but very few audio formats appear to do this.

The bit rate for CD-quality stereo sound is  $44100 \times 2 \times 16$  bits/s = 1411.2 kb/s. This quality measure is particularly popular for lossy audio formats where the uncompressed audio usually is the same (CD-quality). However, it should be remembered that even two audio files in the same file format and with the same bit rate may be of very different quality because the encoding programs may be of different quality.

Below we will read files in the *wav*-format. This format was developed by Microsoft and IBM, and is one of the most common file formats for CD-quality audio. It uses a 32-bit integer to specify the file size at the beginning of the file, which means that a WAV-file cannot be larger than 4 GB.

**Newer formats.** Newer formats with higher quality are available. Music is distributed in various formats on DVDs (DVD-video, DVD-audio, Super Audio CD) with sampling rates up to 192 000 and up to 24 bits per sample. These formats also support surround sound (up to seven channels in contrast to the two stereo channels on a CD). In the following we will assume all sound to be

digital. Later we will return to how we reconstruct audible sound from digital sound.

In the following we will briefly discuss the basic properties of sound: *loudness* (the size of the variations), and *frequency* (the number of variations per second). We will then address to what extent sounds can be decomposed as a sum of different frequencies, and look at important operations on sound, called *filters*, which preserve frequencies. We will also see how we can experiment with digital sound.

The functionality for accessing sound in this chapter is collected in a module called `sound`.

## 1.1 Sound and digital sound: Loudness and frequency

An example of a simple sound is shown in the left plot in Figure 1.1 where the oscillations in air pressure are plotted against time. The initial air pressure has the value 101 325 Pa, which is the normal air pressure at sea level. Then the pressure varies more and more until it oscillates regularly between 101 323 Pa and 101 327 Pa. In the area where the air pressure is constant, no sound will be heard, but as the variations increase in size, the sound becomes louder and louder until about time  $t = 0.03$  where the size of the oscillations becomes constant.

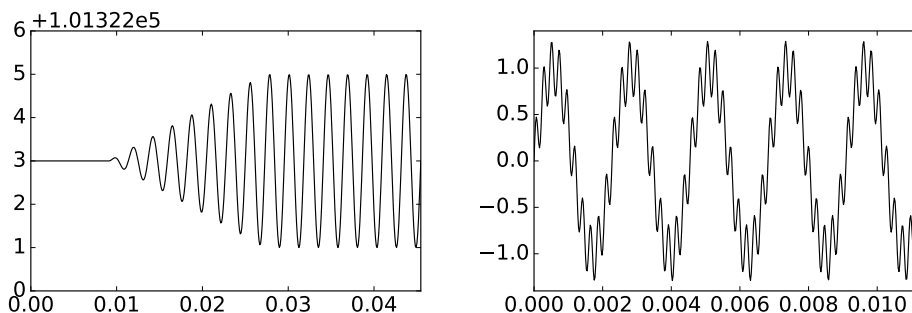


Figure 1.1: An audio signal shown in terms of air pressure (left), and in terms of the difference from the ambient air pressure (right).

When discussing sound, one is usually only interested in the variations in air pressure, so the ambient air pressure (101 325 Pa) is subtracted from the measurement. Everyday sounds typically correspond to variations in air pressure of about 0.00002–2 Pa (0.00002 Pa corresponds to a just audible sound), while a jet engine may cause variations as large as 200 Pa. Short exposure to variations of about 20 Pa may in fact lead to hearing damage. The volcanic eruption at Krakatoa, Indonesia, in 1883, produced a sound wave with variations as large as almost 100 000 Pa, and the explosion could be heard 5000 km away.

The right plot in Figure 1.1 shows another sound which displays a slow, cos-like, variation in air pressure, with some smaller and faster variations imposed on this. This combination of several kinds of systematic oscillations in air pressure is typical for general sounds. The size of the oscillations is directly related to the loudness of the sound. The range of the oscillations is so big that it is common to measure the loudness of a sound on a logarithmic scale:

**Fact 1.3.** *Sound pressure and decibels.*

It is common to relate a given sound pressure to the smallest sound pressure that can be perceived, as a level on a *decibel scale*,

$$L_p = 10 \log_{10} \left( \frac{p^2}{p_{\text{ref}}^2} \right) = 20 \log_{10} \left( \frac{p}{p_{\text{ref}}} \right).$$

Here  $p$  is the measured sound pressure while  $p_{\text{ref}}$  is the sound pressure of a just perceivable sound, usually considered to be 0.00002 Pa.

The square of the sound pressure appears in the definition of  $L_p$  since this represents the *power* of the sound which is relevant for what we perceive as loudness.

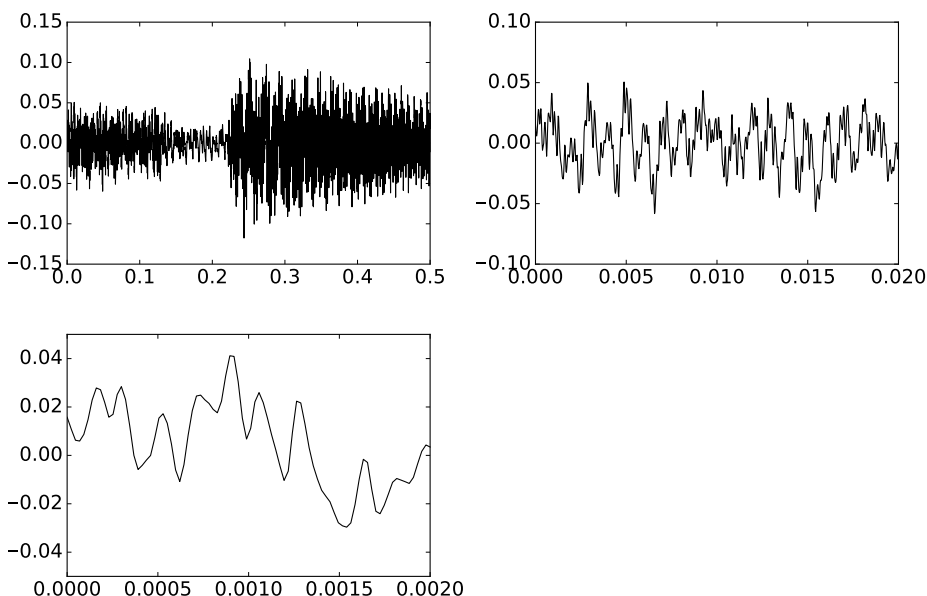


Figure 1.2: Variations in air pressure during parts of a song. The first 0.5 seconds, the first 0.02 seconds, and the first 0.002 seconds.

The sounds in Figure 1.1 are synthetic in that they were constructed from mathematical formulas. The sounds in Figure 1.2 on the other hand show the variation in air pressure for a song, where there is no mathematical formula involved. In the first half second there are so many oscillations that it is

impossible to see the details, but if we zoom in on the first 0.002 seconds we can see that there is a continuous function behind all the ink. In reality the air pressure varies more than this, even over this short time period, but the measuring equipment may not be able to pick up those variations, and it is also doubtful whether we would be able to perceive such rapid variations.

### 1.1.1 The frequency of a sound

The other important characteristic in sound is frequency, i.e. the speed of the variations. To make this concept more precise, let us start with a couple of definitions.

**Definition 1.4.** *Periodic functions.*

A real function  $f$  is said to be periodic with period  $T$  if

$$f(t + T) = f(t)$$

for all real numbers  $t$ .

Note that all the values of a periodic function  $f$  with period  $T$  are known if  $f(t)$  is known for all  $t$  in the interval  $[0, T)$ . The following will be our prototype for periodic functions:

**Observation 1.5.** *Frequency.*

If  $\nu$  is a real number, the function  $f(t) = \sin(2\pi\nu t)$  is periodic with period  $T = 1/\nu$ . When  $t$  varies in the interval  $[0, 1]$ , this function covers a total of  $\nu$  periods. This is expressed by saying that  $f$  has *frequency*  $\nu$ . Frequency is measured in Hz (Hertz) which is the same as  $s^{-1}$  (the time  $t$  is measured in seconds). The function  $\sin(2\pi\nu t)$  is also called a *pure tone*.

Clearly  $\sin(2\pi\nu t)$  and  $\cos(2\pi\nu t)$  have the same frequency, and they are simply shifted versions of one another (since  $\cos(2\pi\nu t) = \sin(2\pi\nu t + \pi/2)$ ). Both, as well as linear combinations of them, are called pure tones with frequency  $\nu$ . Due to this, the complex functions  $e^{\pm 2\pi i \nu t} = \cos(2\pi\nu t) \pm i \sin(2\pi\nu t)$  will also be called pure tones. They will also turn out to be useful in the following.

If we are to perceive variations in air pressure as sound, they must fall within a certain range. It turns out that, for a human with good hearing to perceive a sound, the number of variations per second must be in the range 20–20 000.

There is a simple way to change the period of a periodic function, namely by multiplying the argument by a constant. Figure 1.3 illustrates this. The function in the upper left is the plain  $\sin t$  which covers one period when  $t$  varies in the interval  $[0, 2\pi]$ . By multiplying the argument by  $2\pi$ , the period is squeezed into the interval  $[0, 1]$  so the function  $\sin(2\pi t)$  has frequency  $\nu = 1$ . Then, by also multiplying the argument by 2, we push two whole periods into the interval  $[0, 1]$ , so the function  $\sin(2\pi 2t)$  has frequency  $\nu = 2$ . In the lower right the argument has been multiplied by 5 — hence the frequency is 5 and there are five whole periods in the interval  $[0, 1]$ .

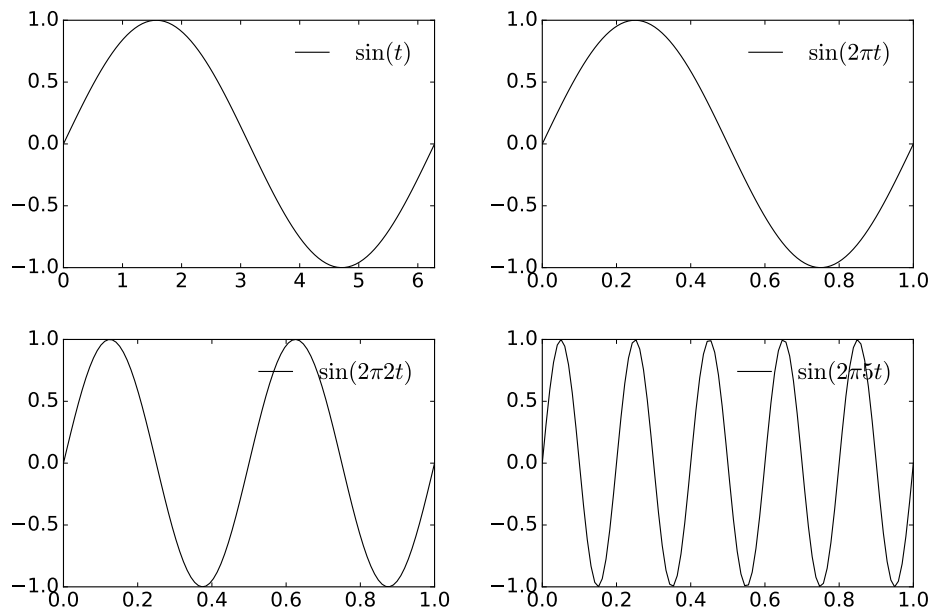


Figure 1.3: Versions of sin with different frequencies.

### 1.1.2 Working with digital sound on a computer

Before we can do anything at all with digital sound, we need to know how we can read and write such data from and to files, and also how to play the data on the computer. These commands are as follows.

```
x, fs = audioread(filename) # Read from file
play(x, fs)                 # Play the entire sound
audiowrite(filename, x, fs) # Write to file
```

These functions can be found in the module `sound`. Note that the method `play` does not block - if we play several sounds in succession they will be played simultaneously. To avoid this, we can block the program ourselves using the `raw_input` function, in order to wait for input from the terminal.

`play` basically sends the array of sound samples `x` and sample rate `fs` to the sound card, which uses some method for reconstructing the sound to an analog sound signal. This analog signal is then sent to the loudspeakers and we hear the sound.

The sound samples can have different data types. We will always assume that they are of type `double`. The computer requires that they have values between  $-1$  and  $1$  (0 corresponding to no variation in air pressure from ambience, and  $-1$  and  $1$  the largest variations in air pressure). If they are not the behaviour when the sound is played is undefined.

You can also create the vector `x` you play on your own, without reading it from file. Below we do this for pure tones.



**Example 1.1: Listen to different channels**

Our audio sample file actually has two sound channels. In such cases  $\mathbf{x}$  is actually a matrix with two columns, and each column represents a sound channel. To listen to each channel we can run the following code.

```
play(x[:, 0], fs)
```

```
play(x[:, 1], fs)
```

You may not hear a difference between the two channels. There may still be differences, however, they may only be notable when the channels are sent to different loudspeakers.

We will later apply different operations to sound. It is possible to apply these operations to the sound channels simultaneously, and we will mostly do this. Sounds we generate on our own, such as pure tones, will mostly be generated in one channel.

**Example 1.2: Playing the sound backwards**

At times a popular game has been to play music backwards to try and find secret messages. In the old days of analog music on vinyl this was not so easy, but with digital sound it is quite simple; we just need to reverse the samples. To do this we just loop through the array and put the last samples first.

Let  $\mathbf{x} = (x_i)_{i=0}^{N-1}$  be the samples of a digital sound. Then the samples  $\mathbf{y} = (y_i)_{i=0}^{N-1}$  of the reverse sound are given by

$$y_i = x_{N-i-1}, \text{ for } i = 0, 1, \dots, N-1.$$

When we reverse the sound samples, we have to reverse the elements in both sound channels. For our audio sample file this can be performed as follows.

```
z = x[:, (-1), :]  
play(z, fs)
```

Performing this on our sample file you generate the sound you can find in the file `castanetsreverse.wav`.

**Example 1.3: Playing pure tones.**

To create the samples of a pure tone we can write

```
t = linspace(0, antsec, fs*antsec)  
x = sin(2*pi*f*t)
```

Here  $f$  is the frequency, `antsec` the length in seconds, and `fs` the sampling rate.

A pure tone with frequency 440 Hz can be found in the file `puretone440.wav`, and a pure tone with frequency 1500 Hz can be found in the file `puretone1500.wav`.

**Example 1.4: The square wave**

There are many other ways in which a function can oscillate regularly. The *square wave* is one such, but we will see later that it can not be written as a simple, trigonometric function. Given a period  $T$  we define the square wave to be 1 on the first half of each period, and  $-1$  on the second half:

$$f_s(t) = \begin{cases} 1, & \text{if } 0 \leq t < T/2; \\ -1, & \text{if } T/2 \leq t < T. \end{cases} \quad (1.1)$$

In the left part of Figure 1.4 we have plotted the square wave with the same period we used for the pure tone.

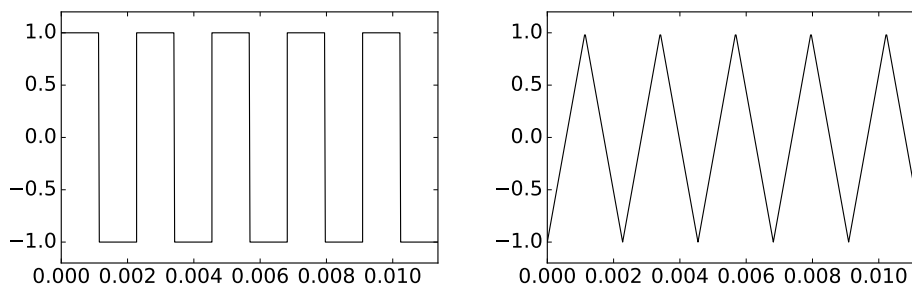


Figure 1.4: The first five periods of the square wave and the triangle wave.

Let us listen also listen to the square wave. We can first create the samples for one period.

```
antsec = 3
samplesperperiod = fs/f # The number of samples for one period
oneperiod = hstack([ones((samplesperperiod/2),dtype=float), \
                    -ones((samplesperperiod/2),dtype=float)])
```

Then we repeat one period to obtain a sound with the desired length, and play it as follows.

```
x = tile(oneperiod, antsec*f) # Repeat one period
play(x, fs)
```

You can listen to this square wave in the file [square440.wav](#). We hear a sound which seems to have the same "base frequency" as  $\sin(2\pi 440t)$ , but the square wave is less pleasant to listen to: There seems to be some "sharp corners" in the sound, translating into a rather shrieking, piercing sound. We will later explain this by the fact that the square wave can be viewed as a sum of many frequencies, and that many frequencies pollute the sound so that it is not pleasant to listen to.

**Example 1.5: The triangle wave**

Given a period  $T$  we define the *triangle wave* to increase linearly from  $-1$  to  $1$  on the first half of each period, and decrease linearly from  $1$  to  $-1$  on the second half of each period. This means that we can define it as the function

$$f_t(t) = \begin{cases} 4t/T - 1, & \text{if } 0 \leq t < T/2; \\ 3 - 4t/T, & \text{if } T/2 \leq t < T. \end{cases} \quad (1.2)$$

In the right part of Figure 1.4 we have plotted the square wave with the same period we used for the pure tone. In Exercise 1.11 you will be asked to reproduce this plot, as well as construct and play the corresponding sound, which also can be found in the file [triangle440.wav](#). Again you will note that the triangle wave has the same "base frequency" as  $\sin(2\pi 440t)$ , and is less pleasant to listen to than this pure tone. However, one can argue that it is somewhat more pleasant to listen to than a square wave. This will also be explained in terms of pollution with other frequencies later.

In the next sections we will address why many sounds may be approximated well by adding many pure sounds together. In particular, this will apply for the square wave and the triangle wave above, and we will also have something to say about why they sound so different.

**Exercise 1.6: The Krakatoa explosion**

Compute the loudness of the Krakatoa explosion on the decibel scale, assuming that the variation in air pressure peaked at 100 000 Pa.

**Exercise 1.7: Sum of two pure tones**

Consider a sum of two pure tones,  $f(t) = A_1 \sin(2\pi\nu_1 t) + A_2 \sin(2\pi\nu_2 t)$ . For which values of  $A_1, A_2, \nu_1, \nu_2$  is  $f$  periodic? What is the period of  $f$  when it is periodic?

**Exercise 1.8: Sum of two pure tones**

Find two constant  $a$  and  $b$  so that the function  $f(t) = a \sin(2\pi 440t) + b \sin(2\pi 4400t)$  resembles the right plot of Figure 1.1 as closely as possible. Generate the samples of this sound, and listen to it.

**Exercise 1.9: Playing with different sample rates**

If we provide another sample rate `fs` to the `play` functions, the sound card will assume a different time distance between neighboring samples. Play and listen to the audio sample file again, but with three different sample rates: `2*fs`, `fs`, and `fs/2`, where `fs` is the sample rate returned by `audioread`.

**Exercise 1.10: Play sound with added noise**

To remove noise from recorded sound can be very challenging, but adding noise is simple. There are many kinds of noise, but one kind is easily obtained by adding random numbers to the samples of a sound. For this we can use the function `random.random` as follows.

```
z = x + c*(2*random.random(shape(x))-1)
```

This adds noise to all channels. The function for returning random numbers returns numbers between 0 and 1, and above we have adjusted these so that they are between  $-1$  and  $1$  instead, as for other sound which can be played by the computer.  $c$  is a constant (usually smaller than 1) that dampens the noise.

Write code which adds noise to the audio sample file, and listen to the result for damping constants  $c=0.4$  and  $c=0.1$ . Remember to scale the sound values after you have added noise, since they may be outside  $[-1, 1]$  then.

**Exercise 1.11: Playing the triangle wave**

Repeat what you did in Example 1.4, but now for the triangle wave of Example 1.5. Start by generating the samples for one period of the triangle wave, then plot five periods, before you generate the sound over a period of three seconds, and play it. Verify that you generate the same sound as in Example 1.5.

**1.2 Fourier series: Basic concepts**

We will now discuss the idea of decomposing a sound into a linear combination of pure sounds. A coefficient in such a decomposition then gives the content at a given frequency. Such a decomposition will pave the way for constructing useful operations on sound, such as amplifying or annihilating certain frequencies: Certain frequencies may not be important for our perception of the sound, so that annihilating or rounding these may not affect how we perceive them. For simplicity we will first restrict to functions which are periodic with period  $T$ , so that they are uniquely defined by their values on  $[0, T]$ . Our analysis can be carried out for square-integrable functions:

**Definition 1.6.** *Continuous and square-integrable functions.*

The set of continuous, real functions defined on an interval  $[0, T]$  is denoted  $C[0, T]$ .

A real function  $f$  defined on  $[0, T]$  is said to be *square integrable* if  $f^2$  is Riemann-integrable, i.e., if the Riemann integral of  $f^2$  on  $[0, T]$  exists,

$$\int_0^T f(t)^2 dt < \infty.$$

The set of all square integrable functions on  $[0, T]$  is denoted  $L^2[0, T]$ .

The sets of continuous and square-integrable functions can be equipped with an inner-product, a generalization of the so-called dot-product for vectors.

**Theorem 1.7.** *Inner product spaces.*

Both  $L^2[0, T]$  and  $C[0, T]$  are vector spaces. Moreover, if the two functions  $f$  and  $g$  lie in  $L^2[0, T]$  (or in  $C[0, T]$ ), then the product  $fg$  is Riemann-integrable (or in  $C[0, T]$ ). Moreover, both spaces are inner product spaces<sup>1</sup> with inner product<sup>2</sup> defined by

$$\langle f, g \rangle = \frac{1}{T} \int_0^T f(t)g(t) dt, \quad (1.3)$$

and associated norm

$$\|f\| = \sqrt{\frac{1}{T} \int_0^T f(t)^2 dt}. \quad (1.4)$$

*Proof.* Since

$$|f + g|^2 \leq (2 \max(|f|, |g|))^2 \leq 4(|f|^2 + |g|^2).$$

$f + g$  is square integrable whenever  $f$  and  $g$  are. It follows that  $L^2[0, T]$  is a vector space. The properties of an inner product space follow directly from the properties of Riemann-integrable functions. Also, since  $|fg| \leq |f|^2 + |g|^2$ , it follows that  $\langle f, g \rangle < \infty$  whenever  $f$  and  $g$  are square integrable. It follows immediately that  $fg$  is Riemann-integrable whenever  $f$  and  $g$  are square integrable.  $\square$

The mysterious factor  $1/T$  is included so that the constant function  $f(t) = 1$  has norm 1, i.e., its role is as a normalizing factor.

Definition 1.6 and Theorem 1.7 state how general we will allow our sounds to be. Theorem 1.7 also explains how we may determine approximations: Recall from linear algebra that the projection of a function  $f$  onto a subspace  $W$  with respect to an inner product  $\langle \cdot, \cdot \rangle$  is the function  $g \in W$  which minimizes  $\|f - g\|$ , also called *the error* (or *least square error*) in the approximation<sup>3</sup>. This projection is therefore also called a best approximation of  $f$  from  $W$  and is characterized by the fact that the function  $f - g$ , also called the *error function*, should be orthogonal to the subspace  $W$ , i.e.

$$\langle f - g, h \rangle = 0, \quad \text{for all } h \in W.$$

If we have an orthogonal basis  $\phi = \{\phi_i\}_{i=1}^m$  for  $W$ , the *orthogonal decomposition theorem* states that the best approximation from  $W$  is

$$g = \sum_{i=1}^m \frac{\langle f, \phi_i \rangle}{\langle \phi_i, \phi_i \rangle} \phi_i. \quad (1.5)$$

<sup>1</sup>See Section 6.1 in [25] for a review of inner products and orthogonality.

<sup>2</sup>See Section 6.7 in [25] for a review of function spaces as inner product spaces.

<sup>3</sup>See Section 6.3 in [25] for a review of projections and least squares approximations.

What we would like is a sequence of spaces

$$V_1 \subset V_2 \subset \cdots \subset V_n \subset \cdots$$

of increasing dimensions so that most sounds can be approximated arbitrarily well by choosing  $n$  large enough, and use the orthogonal decomposition theorem to compute the approximations. It turns out that pure tones can be used for this purpose:

**Definition 1.8.** *Fourier series.*

Let  $V_{N,T}$  be the subspace of  $C[0, T]$  spanned by the set of functions given by

$$\begin{aligned} \mathcal{D}_{N,T} = \{ & 1, \cos(2\pi t/T), \cos(2\pi 2t/T), \cdots, \cos(2\pi Nt/T), \\ & \sin(2\pi t/T), \sin(2\pi 2t/T), \cdots, \sin(2\pi Nt/T) \}. \end{aligned} \quad (1.6)$$

The space  $V_{N,T}$  is called the  $N$ 'th order Fourier space. The  $N$ th-order Fourier series approximation of  $f$ , denoted  $f_N$ , is defined as the best approximation of  $f$  from  $V_{N,T}$  with respect to the inner product defined by (1.3).

We see that pure tones at frequencies  $1/T, 2/T, \dots, N/T$  are a basis for  $V_{N,T}$ . A best approximation at these frequencies, as described above will be called a *Fourier series*. They are similar to Taylor series, where instead polynomials are used in the approximation, but we will see that there is a major difference in how the two approximations are computed. The theory of approximation of functions with Fourier series is referred to as *Fourier analysis*, and is a central tool in practical fields like image- and signal processing, but is also an important field of research within pure mathematics. The approximation  $f_N \in V_{N,T}$  can serve as a compressed version of  $f$  if many of the coefficients can be set to 0 without the error becoming too big.

Note that all the functions in the set  $\mathcal{D}_{N,T}$  are periodic with period  $T$ , but most have an even shorter period ( $\cos(2\pi nt/T)$  also has period  $T/n$ ). In general, the term *fundamental frequency* is used to denote the lowest frequency of a given periodic function.

The next theorem explains that the  $\mathcal{D}_{N,T}$  actually forms a basis for the Fourier spaces, and also how to obtain the coefficients in this basis.

**Theorem 1.9.** *Fourier coefficients.*

The set  $\mathcal{D}_{N,T}$  is an orthogonal basis for  $V_{N,T}$ . In particular, the dimension of  $V_{N,T}$  is  $2N + 1$ , and if  $f$  is a function in  $L^2[0, T]$ , we denote by  $a_0, \dots, a_N$  and  $b_1, \dots, b_N$  the coordinates of  $f_N$  in the basis  $\mathcal{D}_{N,T}$ , i.e.

$$f_N(t) = a_0 + \sum_{n=1}^N (a_n \cos(2\pi nt/T) + b_n \sin(2\pi nt/T)). \quad (1.7)$$

The  $a_0, \dots, a_N$  and  $b_1, \dots, b_N$  are called the (real) Fourier coefficients of  $f$ , and they are given by

$$a_0 = \langle f, 1 \rangle = \frac{1}{T} \int_0^T f(t) dt, \quad (1.8)$$

$$a_n = 2 \langle f, \cos(2\pi nt/T) \rangle = \frac{2}{T} \int_0^T f(t) \cos(2\pi nt/T) dt \quad \text{for } n \geq 1, \quad (1.9)$$

$$b_n = 2 \langle f, \sin(2\pi nt/T) \rangle = \frac{2}{T} \int_0^T f(t) \sin(2\pi nt/T) dt \quad \text{for } n \geq 1. \quad (1.10)$$

*Proof.* Assume first that  $m \neq n$ . We compute the inner product

$$\begin{aligned} & \langle \cos(2\pi mt/T), \cos(2\pi nt/T) \rangle \\ &= \frac{1}{T} \int_0^T \cos(2\pi mt/T) \cos(2\pi nt/T) dt \\ &= \frac{1}{2T} \int_0^T (\cos(2\pi mt/T + 2\pi nt/T) + \cos(2\pi mt/T - 2\pi nt/T)) \\ &= \frac{1}{2T} \left[ \frac{T}{2\pi(m+n)} \sin(2\pi(m+n)t/T) + \frac{T}{2\pi(m-n)} \sin(2\pi(m-n)t/T) \right]_0^T \\ &= 0. \end{aligned}$$

Here we have added the two identities  $\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y$  together to obtain an expression for  $\cos(2\pi mt/T) \cos(2\pi nt/T) dt$  in terms of  $\cos(2\pi mt/T + 2\pi nt/T)$  and  $\cos(2\pi mt/T - 2\pi nt/T)$ . By testing all other combinations of sin and cos also, we obtain the orthogonality of all functions in  $\mathcal{D}_{N,T}$ . We also obtain that

$$\begin{aligned} \langle \cos(2\pi mt/T), \cos(2\pi mt/T) \rangle &= \frac{1}{2} \\ \langle \sin(2\pi mt/T), \sin(2\pi mt/T) \rangle &= \frac{1}{2} \\ \langle 1, 1 \rangle &= 1, \end{aligned}$$

From the orthogonal decomposition theorem (1.5) it follows from this that

$$\begin{aligned}
f_N(t) &= \frac{\langle f, 1 \rangle}{\langle 1, 1 \rangle} 1 + \sum_{n=1}^N \frac{\langle f, \cos(2\pi nt/T) \rangle}{\langle \cos(2\pi nt/T), \cos(2\pi nt/T) \rangle} \cos(2\pi nt/T) \\
&\quad + \sum_{n=1}^N \frac{\langle f, \sin(2\pi nt/T) \rangle}{\langle \sin(2\pi nt/T), \sin(2\pi nt/T) \rangle} \sin(2\pi nt/T) \\
&= \frac{\frac{1}{T} \int_0^T f(t) dt}{1} + \sum_{n=1}^N \frac{\frac{1}{T} \int_0^T f(t) \cos(2\pi nt/T) dt}{\frac{1}{2}} \cos(2\pi nt/T) \\
&\quad + \sum_{n=1}^N \frac{\frac{1}{T} \int_0^T f(t) \sin(2\pi nt/T) dt}{\frac{1}{2}} \sin(2\pi nt/T) \\
&= \frac{1}{T} \int_0^T f(t) dt + \sum_{n=1}^N \left( \frac{2}{T} \int_0^T f(t) \cos(2\pi nt/T) dt \right) \cos(2\pi nt/T) \\
&\quad + \sum_{n=1}^N \left( \frac{2}{T} \int_0^T f(t) \sin(2\pi nt/T) dt \right) \sin(2\pi nt/T).
\end{aligned}$$

Equations (1.8)-(1.10) now follow by comparison with Equation (1.7).  $\square$

From the orthogonality and the inner products of the Fourier basis functions it immediately follows that

$$\|f_N\|^2 = a_0^2 + \frac{1}{2} \sum_{n=1}^N (a_n^2 + b_n^2)$$

Since  $f$  is a function in time, and the  $a_n, b_n$  represent contributions from different frequencies, the Fourier series can be thought of as a change of coordinates, from what often is called the *time domain*, to the *frequency domain* (or *Fourier domain*). We will call the basis  $\mathcal{D}_{N,T}$  the *N'th order Fourier basis* for  $V_{N,T}$ . We note that  $\mathcal{D}_{N,T}$  is not an orthonormal basis; it is only orthogonal.

In the signal processing literature, Equation (1.7) is known as *the synthesis equation*, since the original function  $f$  is synthesized as a sum of the basis functions. Equations (1.8)-(1.10) are also called *analysis equations*.

An illustration of convergence of Fourier series is shown in Figure 1.5 where the cubic polynomial  $f(x) = -\frac{1}{3}x^3 + \frac{1}{2}x^2 - \frac{3}{16}x + 1$  is approximated by a 9'th order Fourier series. The trigonometric approximation is periodic with period 1 so the approximation becomes poor at the ends of the interval since the cubic polynomial is not periodic. The approximation is plotted on a larger interval in the right plot in Figure 1.5, where its periodicity is clearly visible.

Let us compute the Fourier series of some interesting functions.

### Example 1.12: Fourier coefficients of the square wave

Let us compute the Fourier coefficients of the square wave, as defined by Equation (1.1) in Example 1.4. If we first use Equation (1.8) we obtain



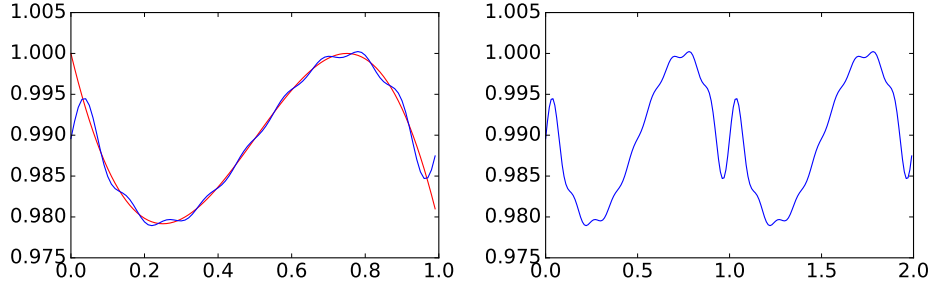


Figure 1.5: The cubic polynomial  $f(x) = -\frac{1}{3}x^3 + \frac{1}{2}x^2 - \frac{3}{16}x + 1$  on the interval  $[0, 1]$ , together with its Fourier series approximation from  $V_{9,1}$ . The function and its Fourier series is shown left. The Fourier series on a larger interval is shown right.

$$a_0 = \frac{1}{T} \int_0^T f_s(t) dt = \frac{1}{T} \int_0^{T/2} dt - \frac{1}{T} \int_{T/2}^T dt = 0.$$

Using Equation (1.9) we get

$$\begin{aligned} a_n &= \frac{2}{T} \int_0^T f_s(t) \cos(2\pi nt/T) dt \\ &= \frac{2}{T} \int_0^{T/2} \cos(2\pi nt/T) dt - \frac{2}{T} \int_{T/2}^T \cos(2\pi nt/T) dt \\ &= \frac{2}{T} \left[ \frac{T}{2\pi n} \sin(2\pi nt/T) \right]_0^{T/2} - \frac{2}{T} \left[ \frac{T}{2\pi n} \sin(2\pi nt/T) \right]_{T/2}^T \\ &= \frac{2}{T} \frac{T}{2\pi n} ((\sin(n\pi) - \sin 0) - (\sin(2n\pi) - \sin(n\pi))) = 0. \end{aligned}$$

Finally, using Equation (1.10) we obtain

$$\begin{aligned}
b_n &= \frac{2}{T} \int_0^T f_s(t) \sin(2\pi nt/T) dt \\
&= \frac{2}{T} \int_0^{T/2} \sin(2\pi nt/T) dt - \frac{2}{T} \int_{T/2}^T \sin(2\pi nt/T) dt \\
&= \frac{2}{T} \left[ -\frac{T}{2\pi n} \cos(2\pi nt/T) \right]_0^{T/2} + \frac{2}{T} \left[ \frac{T}{2\pi n} \cos(2\pi nt/T) \right]_{T/2}^T \\
&= \frac{2}{T} \frac{T}{2\pi n} ((-\cos(n\pi) + \cos 0) + (\cos(2n\pi) - \cos(n\pi))) \\
&= \frac{2(1 - \cos(n\pi))}{n\pi} \\
&= \begin{cases} 0, & \text{if } n \text{ is even;} \\ 4/(n\pi), & \text{if } n \text{ is odd.} \end{cases}
\end{aligned}$$

In other words, only the  $b_n$ -coefficients with  $n$  odd in the Fourier series are nonzero. This means that the Fourier series of the square wave is

$$\frac{4}{\pi} \sin(2\pi t/T) + \frac{4}{3\pi} \sin(2\pi 3t/T) + \frac{4}{5\pi} \sin(2\pi 5t/T) + \frac{4}{7\pi} \sin(2\pi 7t/T) + \dots \quad (1.11)$$

With  $N = 20$ , there are 10 trigonometric terms in this sum. The corresponding Fourier series can be plotted over one period with the following code.

```

N = 20
T = 1/440.
t = linspace(0, T, 100)
x = zeros(len(t))
for k in range(1, N + 1, 2):
    x += (4/(k*pi))*sin(2*pi*k*t/T)
plt.figure()
plt.plot(t, x, 'k-')

```

The left plot in Figure 1.6 shows the resulting plot. In the right plot the values of the first 100 Fourier coefficients  $b_n$  are shown, to see that they actually converge to zero. This is clearly necessary in order for the Fourier series to converge.

Even though  $f$  oscillates regularly between  $-1$  and  $1$  with period  $T$ , the discontinuities mean that it is far from the simple  $\sin(2\pi t/T)$  which corresponds to a pure tone of frequency  $1/T$ . Clearly  $b_1 \sin(2\pi t/T)$  is the dominant term in the Fourier series. This is not surprising since the square wave has the same period as this term, but the additional terms in the Fourier series pollute the pure sound. As we include more and more of these, we gradually approach the square wave.

There is a connection between how fast the Fourier coefficients go to zero, and how we perceive the sound. A pure sine sound has only one nonzero coefficient, while the square wave Fourier coefficients decrease as  $1/n$ , making the sound

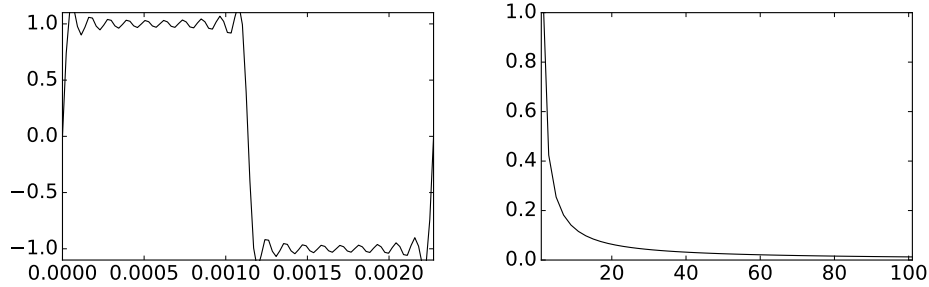


Figure 1.6: The Fourier series of the square wave with  $N = 20$ , and the values for the first 100 Fourier coefficients  $b_n$ .

less pleasant. This explains what we heard when we listened to the sound in Example 1.4. Also, it explains why we heard the same pitch as the pure tone, since the first frequency in the Fourier series has the same frequency as the pure tone we listened to, and since this had the highest value.

Let us listen to the Fourier series approximations of the square wave. For  $N = 1$  the sound can be found in the file [square440s1.wav](#). This sounds exactly like the pure sound with frequency 440Hz, as noted above. For  $N = 5$  the sound can be found in the file [square440s5.wav](#), and for  $N = 9$  it can be found in the file [square440s9.wav](#). The latter sounds are more like the square wave itself. As we increase  $N$  we can hear how the introduction of more frequencies gradually pollutes the sound more.

### Example 1.13: Fourier coefficients of the triangle wave

Let us also compute the Fourier coefficients of the triangle wave, as defined by Equation (1.2) in Example 1.5. We now have

$$a_0 = \frac{1}{T} \int_0^{T/2} \frac{4}{T} \left( t - \frac{T}{4} \right) dt + \frac{1}{T} \int_{T/2}^T \frac{4}{T} \left( \frac{3T}{4} - t \right) dt.$$

Instead of computing this directly, it is quicker to see geometrically that the graph of  $f_t$  has as much area above as below the  $x$ -axis, so that this integral must be zero. Similarly, since  $f_t$  is symmetric about the midpoint  $T/2$ , and  $\sin(2\pi nt/T)$  is antisymmetric about  $T/2$ , we have that  $f_t(t) \sin(2\pi nt/T)$  also is antisymmetric about  $T/2$ , so that

$$\int_0^{T/2} f_t(t) \sin(2\pi nt/T) dt = - \int_{T/2}^T f_t(t) \sin(2\pi nt/T) dt.$$

This means that, for  $n \geq 1$ ,

$$b_n = \frac{2}{T} \int_0^{T/2} f_t(t) \sin(2\pi nt/T) dt + \frac{2}{T} \int_{T/2}^T f_t(t) \sin(2\pi nt/T) dt = 0.$$

For the final coefficients, since both  $f$  and  $\cos(2\pi nt/T)$  are symmetric about  $T/2$ , we get for  $n \geq 1$ ,

$$\begin{aligned} a_n &= \frac{2}{T} \int_0^{T/2} f_t(t) \cos(2\pi nt/T) dt + \frac{2}{T} \int_{T/2}^T f_t(t) \cos(2\pi nt/T) dt \\ &= \frac{4}{T} \int_0^{T/2} f_t(t) \cos(2\pi nt/T) dt = \frac{4}{T} \int_0^{T/2} \frac{4}{T} \left( t - \frac{T}{4} \right) \cos(2\pi nt/T) dt \\ &= \frac{16}{T^2} \int_0^{T/2} t \cos(2\pi nt/T) dt - \frac{4}{T} \int_0^{T/2} \cos(2\pi nt/T) dt \\ &= \frac{4}{n^2 \pi^2} (\cos(n\pi) - 1) \\ &= \begin{cases} 0, & \text{if } n \text{ is even;} \\ -8/(n^2 \pi^2), & \text{if } n \text{ is odd.} \end{cases} \end{aligned}$$

where we have dropped the final tedious calculations (use integration by parts). From this it is clear that the Fourier series of the triangle wave is

$$-\frac{8}{\pi^2} \cos(2\pi t/T) - \frac{8}{3^2 \pi^2} \cos(2\pi 3t/T) - \frac{8}{5^2 \pi^2} \cos(2\pi 5t/T) - \frac{8}{7^2 \pi^2} \cos(2\pi 7t/T) + \dots \quad (1.12)$$

In Figure 1.7 we have repeated the plots used for the square wave, for the triangle wave. The figure indicates that the Fourier series coefficients decay faster.

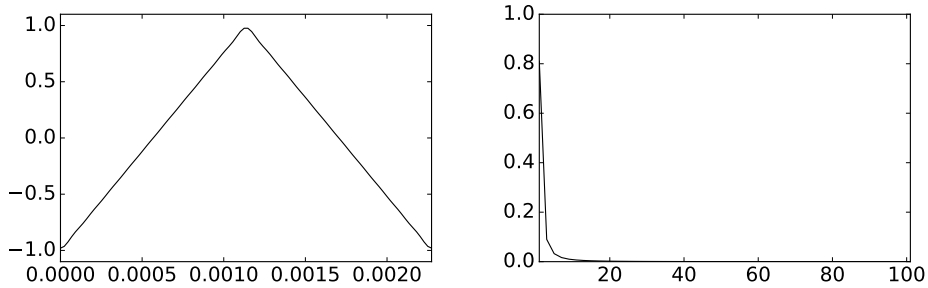


Figure 1.7: The Fourier series of the triangle wave with  $N = 20$ , and the values for the first 100 Fourier coefficients  $a_n$ .

Let us also listen to different Fourier series approximations of the triangle wave. For  $N = 1$  it can be found in the file [triangle440s1.wav](#). Again, this sounds exactly like the pure sound with frequency 440Hz. For  $N = 5$  the Fourier series approximation can be found in the file [triangle440s5.wav](#), and for  $N = 9$  it can be found in the file [triangle440s9.wav](#). Again the latter sounds are more like the triangle wave itself, and as we increase  $N$  we can hear that more frequencies pollutes the sound. However, since the triangle wave Fourier coefficients decrease as  $1/n^2$  rather than  $1/n$ , the sound is somewhat less unpleasant. The faster convergence can also be heard.

**Example 1.14: Fourier coefficients of a simple function**

There is an important lesson to be learned from the previous examples: Even if the signal is nice and periodic, it may not have a nice representation in terms of trigonometric functions. Thus, trigonometric functions may not be the best bases to use for expressing other functions. Unfortunately, many more such cases can be found, as we will now explain. Let us consider a periodic function which is 1 on  $[0, T_0]$ , but 0 is on  $[T_0, T]$ . This is a signal with short duration when  $T_0$  is small compared to  $T$ . We compute that  $y_0 = T_0/T$ , and

$$a_n = \frac{2}{T} \int_0^{T_0} \cos(2\pi nt/T) dt = \frac{1}{\pi n} [\sin(2\pi nt/T)]_0^{T_0} = \frac{\sin(2\pi n T_0/T)}{\pi n}$$

for  $n \geq 1$ . Similar computations hold for  $b_n$ . We see that  $|a_n|$  is of the order  $1/(\pi n)$ , and that infinitely many  $n$  contribute. This function may be thought of as a simple building block, corresponding to a small time segment. However, we see that it is not a simple building block in terms of trigonometric functions. This time segment building block may be useful for restricting a function to smaller time segments, and later on we will see that it still can be useful.

**1.2.1 Fourier series for symmetric and antisymmetric functions**

In Example 1.12 we saw that the Fourier coefficients  $b_n$  vanished, resulting in a sine-series for the Fourier series of the square wave. Similarly, in Example 1.13 we saw that  $a_n$  vanished, resulting in a cosine-series for the triangle wave. This is not a coincident, and is captured by the following result.

**Theorem 1.10. Symmetry and antisymmetry.**

If  $f$  is antisymmetric about 0 (that is, if  $f(-t) = -f(t)$  for all  $t$ ), then  $a_n = 0$ , so the Fourier series is actually a sine-series. If  $f$  is symmetric about 0 (which means that  $f(-t) = f(t)$  for all  $t$ ), then  $b_n = 0$ , so the Fourier series is actually a cosine-series.

The point is that the square wave is antisymmetric about 0, and the triangle wave is symmetric about 0.

*Proof.* Note first that we can write

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos(2\pi nt/T) dt \quad b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin(2\pi nt/T) dt,$$

i.e. we can change the integration bounds from  $[0, T]$  to  $[-T/2, T/2]$ . This follows from the fact that all  $f(t)$ ,  $\cos(2\pi nt/T)$  and  $\sin(2\pi nt/T)$  are periodic with period  $T$ .

Suppose first that  $f$  is symmetric. We obtain

$$\begin{aligned}
b_n &= \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin(2\pi nt/T) dt \\
&= \frac{2}{T} \int_{-T/2}^0 f(t) \sin(2\pi nt/T) dt + \frac{2}{T} \int_0^{T/2} f(t) \sin(2\pi nt/T) dt \\
&= \frac{2}{T} \int_{-T/2}^0 f(t) \sin(2\pi nt/T) dt - \frac{2}{T} \int_0^{-T/2} f(-t) \sin(-2\pi nt/T) dt \\
&= \frac{2}{T} \int_{-T/2}^0 f(t) \sin(2\pi nt/T) dt - \frac{2}{T} \int_{-T/2}^0 f(t) \sin(2\pi nt/T) dt = 0.
\end{aligned}$$

where we have made the substitution  $u = -t$ , and used that  $\sin$  is antisymmetric. The case when  $f$  is antisymmetric can be proved in the same way, and is left as an exercise.  $\square$

### Exercise 1.15: Shifting the Fourier basis vectors

Show that  $\sin(2\pi nt/T + a) \in V_{N,T}$  when  $|n| \leq N$ , regardless of the value of  $a$ .

### Exercise 1.16: Playing the Fourier series of the triangle wave

- Plot the Fourier series of the triangle wave.
- Write code so that you can listen to the Fourier series of the triangle wave. How high must you choose  $N$  for the Fourier series to be indistinguishable from the square/triangle waves themselves?

### Exercise 1.17: Riemann-integrable functions which are not square-integrable

Find a function  $f$  which is Riemann-integrable on  $[0, T]$ , and so that  $\int_0^T f(t)^2 dt$  is infinite.

### Exercise 1.18: When are Fourier spaces included in each other?

Given the two Fourier spaces  $V_{N_1, T_1}$ ,  $V_{N_2, T_2}$ . Find necessary and sufficient conditions in order for  $V_{N_1, T_1} \subset V_{N_2, T_2}$ .

### Exercise 1.19: antisymmetric functions are sine-series

Prove the second part of Theorem 1.10, i.e. show that if  $f$  is antisymmetric about 0 (i.e.  $f(-t) = -f(t)$  for all  $t$ ), then  $a_n = 0$ , i.e. the Fourier series is actually a sine-series.

**Exercise 1.20: More connections between symmetric-/antisymmetric functions and sine-/cosine series**

Show that

- a) Any cosine series  $a_0 + \sum_{n=1}^N a_n \cos(2\pi nt/T)$  is a symmetric function.  
 b) Any sine series  $\sum_{n=1}^N b_n \sin(2\pi nt/T)$  is an antisymmetric function.  
 c) Any periodic function can be written as a sum of a symmetric - and an antisymmetric function by writing  $f(t) = \frac{f(t)+f(-t)}{2} + \frac{f(t)-f(-t)}{2}$ .  
 d) If  $f_N(t) = a_0 + \sum_{n=1}^N (a_n \cos(2\pi nt/T) + b_n \sin(2\pi nt/T))$ , then

$$\frac{f_N(t) + f_N(-t)}{2} = a_0 + \sum_{n=1}^N a_n \cos(2\pi nt/T)$$

$$\frac{f_N(t) - f_N(-t)}{2} = \sum_{n=1}^N b_n \sin(2\pi nt/T).$$

**Exercise 1.21: Fourier series for low-degree polynomials**

Find the Fourier series coefficients of the periodic functions with period  $T$  defined by being  $f(t) = t$ ,  $f(t) = t^2$ , and  $f(t) = t^3$ , on  $[0, T]$ .

**Exercise 1.22: Fourier series for polynomials**

Write down difference equations for finding the Fourier coefficients of  $f(t) = t^{k+1}$  from those of  $f(t) = t^k$ , and write a program which uses this recursion. Use the program to verify what you computed in Exercise 1.21.

**Exercise 1.23: Fourier series of a given polynomial**

Use the previous exercise to find the Fourier series for  $f(x) = -\frac{1}{3}x^3 + \frac{1}{2}x^2 - \frac{3}{16}x + 1$  on the interval  $[0, 1]$ . Plot the 9th order Fourier series for this function. You should obtain the plots from Figure 1.5.

### 1.3 Complex Fourier series

In Section 1.2 we saw how a function can be expanded in a series of sines and cosines. These functions are related to the complex exponential function via Eulers formula

$$e^{ix} = \cos x + i \sin x$$

where  $i$  is the imaginary unit with the property that  $i^2 = -1$ . Because the algebraic properties of the exponential function are much simpler than those of cos and sin, it is often an advantage to work with complex numbers, even

though the given setting is real numbers. This is definitely the case in Fourier analysis. More precisely, we will make the substitutions

$$\cos(2\pi nt/T) = \frac{1}{2} \left( e^{2\pi int/T} + e^{-2\pi int/T} \right) \quad (1.13)$$

$$\sin(2\pi nt/T) = \frac{1}{2i} \left( e^{2\pi int/T} - e^{-2\pi int/T} \right) \quad (1.14)$$

in Definition 1.8. From these identities it is clear that the set of complex exponential functions  $e^{2\pi int/T}$  also is a basis of periodic functions (with the same period) for  $V_{N,T}$ . We may therefore reformulate Definition 1.8 as follows:

**Definition 1.11.** *Complex Fourier basis.*

We define the set of functions

$$\mathcal{F}_{N,T} = \{e^{-2\pi iNt/T}, e^{-2\pi i(N-1)t/T}, \dots, e^{-2\pi it/T}, \quad (1.15)$$

$$1, e^{2\pi it/T}, \dots, e^{2\pi i(N-1)t/T}, e^{2\pi iNt/T}\}, \quad (1.16)$$

and call this the order  $N$  complex Fourier basis for  $V_{N,T}$ .

The function  $e^{2\pi int/T}$  is also called a pure tone with frequency  $n/T$ , just as sines and cosines are. We would like to show that these functions also are orthogonal. To show this, we need to say more on the inner product we have defined by Equation (1.3). A weakness with this definition is that we have assumed real functions  $f$  and  $g$ , so that this can not be used for the complex exponential functions  $e^{2\pi int/T}$ . For general complex functions we will extend the definition of the inner product as follows:

$$\langle f, g \rangle = \frac{1}{T} \int_0^T f \bar{g} dt. \quad (1.17)$$

The associated norm now becomes

$$\|f\| = \sqrt{\frac{1}{T} \int_0^T |f(t)|^2 dt}. \quad (1.18)$$

The motivation behind Equation (1.17), where we have conjugated the second function, lies in the definition of an *inner product for vector spaces over complex numbers*. From before we are used to vector spaces over real numbers, but vector spaces over complex numbers are defined through the same set of axioms as for real vector spaces, only replacing real numbers with complex numbers. For complex vector spaces, the axioms defining an inner product are the same as for real vector spaces, except for that the axiom

$$\langle f, g \rangle = \langle g, f \rangle \quad (1.19)$$

is replaced with the axiom



$$\langle f, g \rangle = \overline{\langle g, f \rangle}, \quad (1.20)$$

i.e. a conjugation occurs when we switch the order of the functions. This new axiom can be used to prove the property  $\langle f, cg \rangle = \bar{c}\langle f, g \rangle$ , which is a somewhat different property from what we know for real inner product spaces. This follows by writing

$$\langle f, cg \rangle = \overline{\langle cg, f \rangle} = \overline{c\langle g, f \rangle} = \bar{c}\overline{\langle g, f \rangle} = \bar{c}\langle f, g \rangle.$$

Clearly the inner product given by (1.17) satisfies Axiom (1.20). With this definition it is quite easy to see that the functions  $e^{2\pi int/T}$  are orthonormal. Using the orthogonal decomposition theorem we can therefore write

$$\begin{aligned} f_N(t) &= \sum_{n=-N}^N \frac{\langle f, e^{2\pi int/T} \rangle}{\langle e^{2\pi int/T}, e^{2\pi int/T} \rangle} e^{2\pi int/T} = \sum_{n=-N}^N \langle f, e^{2\pi int/T} \rangle e^{2\pi int/T} \\ &= \sum_{n=-N}^N \left( \frac{1}{T} \int_0^T f(t) e^{-2\pi int/T} dt \right) e^{2\pi int/T}. \end{aligned}$$

We summarize this in the following theorem, which is a version of Theorem 1.9 which uses the complex Fourier basis:

**Theorem 1.12.** *Complex Fourier coefficients.*

We denote by  $y_{-N}, \dots, y_0, \dots, y_N$  the coordinates of  $f_N$  in the basis  $\mathcal{F}_{N,T}$ , i.e.

$$f_N(t) = \sum_{n=-N}^N y_n e^{2\pi int/T}. \quad (1.21)$$

The  $y_n$  are called the complex Fourier coefficients of  $f$ , and they are given by.

$$y_n = \langle f, e^{2\pi int/T} \rangle = \frac{1}{T} \int_0^T f(t) e^{-2\pi int/T} dt. \quad (1.22)$$

Let us consider two immediate and important consequences of the orthonormal basis we have established. The first one follows directly from the orthonormality.

**Theorem 1.13.** *Parseval's theorem.*

We have that

$$\|f_N\|^2 = \sum_{n=-N}^N |y_n|^2$$

**Theorem 1.14.** *Bessel's inequality.*

For any  $f \in L^2[0, T]$  we have that  $\|f\|^2 \geq \|f_N\|^2$ . In particular, the sequence  $\|f_N\|^2 = \sum_{n=-N}^N |y_n|^2$  is convergent, so that  $y_n \rightarrow 0$ .

Versions of these two results also hold for the real Fourier coefficients.

*Proof.* Since  $f_N(t)$  is the projection of  $f$  onto  $V_{N,T}$  we have that

$$\|f\|^2 = \|f - f_N\|^2 + \|f_N\|^2 \geq \|f_N\|^2,$$

□

In particular the Fourier coefficients go to zero. The results does not say that  $\sum_{n=-N}^N |y_n|^2 \rightarrow \|f\|^2$ , which would imply that  $\|f - f_N\| \rightarrow 0$ . This is more difficult to analyze, and we will only prove a particular case of it in Section 1.6.

If we reorder the real and complex Fourier bases so that the two functions  $\{\cos(2\pi nt/T), \sin(2\pi nt/T)\}$  and  $\{e^{2\pi int/T}, e^{-2\pi int/T}\}$  have the same index in the bases, equations (1.13)-(1.14) give us that the change of coordinates matrix<sup>4</sup> from  $\mathcal{D}_{N,T}$  to  $\mathcal{F}_{N,T}$ , denoted  $P_{\mathcal{F}_{N,T} \leftarrow \mathcal{D}_{N,T}}$ , is represented by repeating the matrix

$$\frac{1}{2} \begin{pmatrix} 1 & 1/i \\ 1 & -1/i \end{pmatrix}$$

along the diagonal (with an additional 1 for the constant function 1). In other words, since  $a_n, b_n$  are coefficients relative to the real basis and  $y_n, y_{-n}$  the corresponding coefficients relative to the complex basis, we have for  $n > 0$ ,

$$\begin{pmatrix} y_n \\ y_{-n} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1/i \\ 1 & -1/i \end{pmatrix} \begin{pmatrix} a_n \\ b_n \end{pmatrix}.$$

This can be summarized by the following theorem:

**Theorem 1.15.** *Change of coordinates between real and complex Fourier bases.*

The complex Fourier coefficients  $y_n$  and the real Fourier coefficients  $a_n, b_n$  of a function  $f$  are related by

$$\begin{aligned} y_0 &= a_0, \\ y_n &= \frac{1}{2}(a_n - ib_n), \\ y_{-n} &= \frac{1}{2}(a_n + ib_n), \end{aligned}$$

for  $n = 1, \dots, N$ .

Combining with Theorem 1.10, Theorem 1.15 can help us state properties of complex Fourier coefficients for symmetric- and antisymmetric functions. We look into this in Exercise 1.34.

Due to the somewhat nicer formulas for the complex Fourier coefficients when compared to the real Fourier coefficients, we will write most Fourier series in complex form in the following.

Let us consider some examples where we compute complex Fourier series.

<sup>4</sup>See Section 4.7 in [25], to review the mathematics behind change of coordinates.

### Example 1.24: Complex Fourier coefficients of a simple function

Let us consider the pure sound  $f(t) = e^{2\pi it/T_2}$  with period  $T_2$ , but let us consider it only on the interval  $[0, T]$  instead, where  $T < T_2$ . Note that this  $f$  is not periodic, since we only consider the part  $[0, T]$  of the period  $[0, T_2]$ . The Fourier coefficients are

$$\begin{aligned} y_n &= \frac{1}{T} \int_0^T e^{2\pi it/T_2} e^{-2\pi int/T} dt = \frac{1}{2\pi iT(1/T_2 - n/T)} \left[ e^{2\pi it(1/T_2 - n/T)} \right]_0^T \\ &= \frac{1}{2\pi i(T/T_2 - n)} \left( e^{2\pi iT/T_2} - 1 \right). \end{aligned}$$

Here it is only the term  $1/(T/T_2 - n)$  which depends on  $n$ , so that  $y_n$  can only be large when  $n$  is close  $T/T_2$ . In Figure 1.8 we have plotted  $|y_n|$  for two different combinations of  $T, T_2$ .

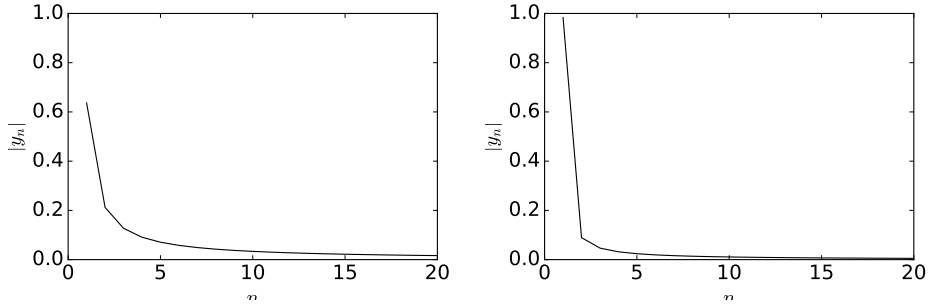


Figure 1.8: Plot of  $|y_n|$  when  $f(t) = e^{2\pi it/T_2}$ , and  $T_2 > T$ . Left:  $T/T_2 = 0.5$ . Right:  $T/T_2 = 0.9$ .

In both examples it is seen that many Fourier coefficients contribute, but this is more visible when  $T/T_2 = 0.5$ . When  $T/T_2 = 0.9$ , most contribution is seen to be in the  $y_1$ -coefficient. This sounds reasonable, since  $f$  then is closest to the pure tone  $f(t) = e^{2\pi it/T}$  of frequency  $1/T$  (which in turn has  $y_1 = 1$  and all other  $y_n = 0$ ).

Apart from computing complex Fourier series, there is an important lesson to be learned from this example: In order for a periodic function to be approximated by other periodic functions, their period must somehow match.

### Example 1.25: Complex Fourier coefficients of composite function

What often is the case is that a sound changes in content over time. Assume that it is equal to a pure tone of frequency  $n_1/T$  on  $[0, T/2)$ , and equal to a pure tone of frequency  $n_2/T$  on  $[T/2, T)$ , i.e.

$$f(t) = \begin{cases} e^{2\pi i n_1 t/T} & \text{on } [0, T_2] \\ e^{2\pi i n_2 t/T} & \text{on } [T_2, T] \end{cases}.$$

When  $n \neq n_1, n_2$  we have that

$$\begin{aligned} y_n &= \frac{1}{T} \left( \int_0^{T/2} e^{2\pi i n_1 t/T} e^{-2\pi i n t/T} dt + \int_{T/2}^T e^{2\pi i n_2 t/T} e^{-2\pi i n t/T} dt \right) \\ &= \frac{1}{T} \left( \left[ \frac{T}{2\pi i (n_1 - n)} e^{2\pi i (n_1 - n)t/T} \right]_0^{T/2} + \left[ \frac{T}{2\pi i (n_2 - n)} e^{2\pi i (n_2 - n)t/T} \right]_{T/2}^T \right) \\ &= \frac{e^{\pi i (n_1 - n)} - 1}{2\pi i (n_1 - n)} + \frac{1 - e^{\pi i (n_2 - n)}}{2\pi i (n_2 - n)}. \end{aligned}$$

Let us restrict to the case when  $n_1$  and  $n_2$  are both even. We see that

$$y_n = \begin{cases} \frac{1}{2} + \frac{1}{\pi i (n_2 - n_1)} & n = n_1, n_2 \\ 0 & n \text{ even, } n \neq n_1, n_2 \\ \frac{n_1 - n_2}{\pi i (n_1 - n)(n_2 - n)} & n \text{ odd} \end{cases}$$

Here we have computed the cases  $n = n_1$  and  $n = n_2$  as above. In Figure 1.9 we have plotted  $|y_n|$  for two different combinations of  $n_1, n_2$ .

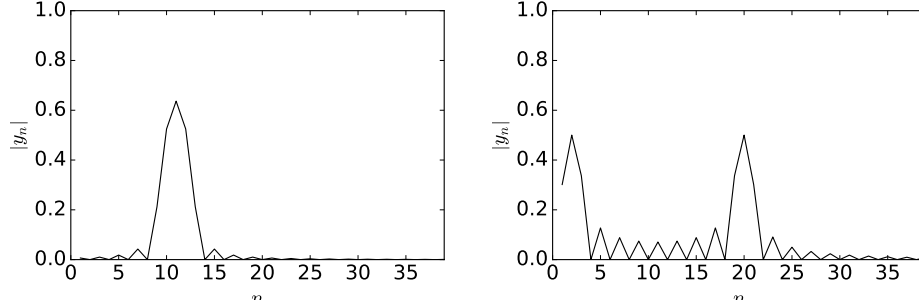


Figure 1.9: Plot of  $|y_n|$  when we have two different pure tones at the different parts of a period. Left:  $n_1 = 10, n_2 = 12$ . Right:  $n_1 = 2, n_2 = 20$ .

We see that, when  $n_1, n_2$  are close, the Fourier coefficients are close to those of a pure tone with  $n \approx n_1, n_2$ , but that also other frequencies contribute. When  $n_1, n_2$  are further apart, we see that the Fourier coefficients are like the sum of the two base frequencies, but that other frequencies contribute also here.

There is an important lesson to be learned from this as well: We should be aware of changes in a sound over time, and it may not be smart to use a frequency representation over a large interval when we know that there are simpler frequency representations on the smaller intervals. The following example shows that, in some cases it is not necessary to compute the Fourier integrals at all, in order to compute the Fourier series.

**Example 1.26: Complex Fourier coefficients of  $f(t) = \cos^3(2\pi t/T)$** 

Let us compute the complex Fourier series of the function  $f(t) = \cos^3(2\pi t/T)$ , where  $T$  is the period of  $f$ . We can write

$$\begin{aligned}\cos^3(2\pi t/T) &= \left(\frac{1}{2}(e^{2\pi it/T} + e^{-2\pi it/T})\right)^3 \\ &= \frac{1}{8}(e^{2\pi i3t/T} + 3e^{2\pi it/T} + 3e^{-2\pi it/T} + e^{-2\pi i3t/T}) \\ &= \frac{1}{8}e^{2\pi i3t/T} + \frac{3}{8}e^{2\pi it/T} + \frac{3}{8}e^{-2\pi it/T} + \frac{1}{8}e^{-2\pi i3t/T}.\end{aligned}$$

From this we see that the complex Fourier series is given by  $y_1 = y_{-1} = \frac{3}{8}$ , and that  $y_3 = y_{-3} = \frac{1}{8}$ . In other words, it was not necessary to compute the Fourier integrals in this case, and we see that the function lies in  $V_{3,T}$ , i.e. there are finitely many terms in the Fourier series. In general, if the function is some trigonometric function, we can often use trigonometric identities to find an expression for the Fourier series.

**Exercise 1.27: Orthonormality of Complex Fourier basis**

Show that the complex functions  $e^{2\pi int/T}$  are orthonormal.

**Exercise 1.28: Complex Fourier series of  $f(t) = \sin^2(2\pi t/T)$** 

Compute the complex Fourier series of the function  $f(t) = \sin^2(2\pi t/T)$ .

**Exercise 1.29: Complex Fourier series of polynomials**

Repeat Exercise 1.21, computing the complex Fourier series instead of the real Fourier series.

**Exercise 1.30: Complex Fourier series and Pascals triangle**

In this exercise we will find a connection with certain Fourier series and the rows in Pascal's triangle.

- a) Show that both  $\cos^n(t)$  and  $\sin^n(t)$  are in  $V_{N,2\pi}$  for  $1 \leq n \leq N$ .
- b) Write down the  $N$ 'th order complex Fourier series for  $f_1(t) = \cos t$ ,  $f_2(t) = \cos^2 t$ , og  $f_3(t) = \cos^3 t$ .
- c) In b) you should be able to see a connection between the Fourier coefficients and the three first rows in Pascal's triangle. Formulate and prove a general relationship between row  $n$  in Pascal's triangle and the Fourier coefficients of  $f_n(t) = \cos^n t$ .

**Exercise 1.31: Complex Fourier coefficients of the square wave**

Compute the complex Fourier coefficients of the square wave using Equation (1.22), i.e. repeat the calculations from Example 1.12 for the complex case. Use Theorem 1.15 to verify your result.

**Exercise 1.32: Complex Fourier coefficients of the triangle wave**

Repeat Exercise 1.31 for the triangle wave.

**Exercise 1.33: Complex Fourier coefficients of low-degree polynomials**

Use Equation (1.22) to compute the complex Fourier coefficients of the periodic functions with period  $T$  defined by, respectively,  $f(t) = t$ ,  $f(t) = t^2$ , and  $f(t) = t^3$ , on  $[0, T]$ . Use Theorem 1.15 to verify your calculations from Exercise 1.21.

**Exercise 1.34: Complex Fourier coefficients for symmetric and antisymmetric functions**

In this exercise we will prove a version of Theorem 1.10 for complex Fourier coefficients.

- a) If  $f$  is symmetric about 0, show that  $y_n$  is real, and that  $y_{-n} = y_n$ .
- b) If  $f$  is antisymmetric about 0, show that the  $y_n$  are purely imaginary,  $y_0 = 0$ , and that  $y_{-n} = -y_n$ .
- c) Show that  $\sum_{n=-N}^N y_n e^{2\pi i n t/T}$  is symmetric when  $y_{-n} = y_n$  for all  $n$ , and rewrite it as a cosine-series.
- d) Show that  $\sum_{n=-N}^N y_n e^{2\pi i n t/T}$  is antisymmetric when  $y_0 = 0$  and  $y_{-n} = -y_n$  for all  $n$ , and rewrite it as a sine-series.

**1.4 Some properties of Fourier series**

We continue by establishing some important properties of Fourier series, in particular the Fourier coefficients for some important functions. In these lists, we will use the notation  $f \rightarrow y_n$  to indicate that  $y_n$  is the  $n$ 'th (complex) Fourier coefficient of  $f(t)$ .

**Theorem 1.16.** *Fourier series pairs.*

The functions 1,  $e^{2\pi i n t/T}$ , and  $\chi_{-a,a}$  have the Fourier coefficients

$$\begin{aligned} 1 &\rightarrow \mathbf{e}_0 = (1, 0, 0, 0, \dots) \\ e^{2\pi i n t/T} &\rightarrow \mathbf{e}_n = (0, 0, \dots, 1, 0, 0, \dots) \\ \chi_{-a,a} &\rightarrow \frac{\sin(2\pi n a/T)}{\pi n}. \end{aligned}$$

The 1 in  $\mathbf{e}_n$  is at position  $n$  and the function  $\chi_{-a,a}$  is the characteristic function of the interval  $[-a, a]$ , defined by

$$\chi_{-a,a}(t) = \begin{cases} 1, & \text{if } t \in [-a, a]; \\ 0, & \text{otherwise.} \end{cases}$$

The first two pairs are easily verified, so the proofs are omitted. The case for  $\chi_{-a,a}$  is very similar to the square wave, but easier to prove, and therefore also omitted.

**Theorem 1.17.** *Fourier series properties.*

The mapping  $f \rightarrow y_n$  is linear: if  $f \rightarrow x_n$ ,  $g \rightarrow y_n$ , then

$$af + bg \rightarrow ax_n + by_n$$

For all  $n$ . Moreover, if  $f$  is real and periodic with period  $T$ , the following properties hold:

1.  $y_n = \overline{y_{-n}}$  for all  $n$ .
2. If  $f(t) = f(-t)$  (i.e.  $f$  is symmetric), then all  $y_n$  are real, so that  $b_n$  are zero and the Fourier series is a cosine series.
3. If  $f(t) = -f(-t)$  (i.e.  $f$  is antisymmetric), then all  $y_n$  are purely imaginary, so that the  $a_n$  are zero and the Fourier series is a sine series.
4. If  $g(t) = f(t-d)$  (i.e.  $g$  is the function  $f$  delayed by  $d$ ) and  $f \rightarrow y_n$ , then  $g \rightarrow e^{-2\pi i n d/T} y_n$ .
5. If  $g(t) = e^{2\pi i d t/T} f(t)$  with  $d$  an integer, and  $f \rightarrow y_n$ , then  $g \rightarrow y_{n-d}$ .
6. Let  $d$  be a number. If  $f \rightarrow y_n$ , then  $f(d+t) = f(d-t)$  for all  $t$  if and only if the argument of  $y_n$  is  $-2\pi n d/T$  for all  $n$ .

*Proof.* The proof of linearity is left to the reader. Property 1 follows immediately by writing

$$\begin{aligned} y_n &= \frac{1}{T} \int_0^T f(t) e^{-2\pi i n t/T} dt = \overline{\frac{1}{T} \int_0^T f(t) e^{2\pi i n t/T} dt} \\ &= \frac{1}{T} \int_0^T f(t) e^{-2\pi i (-n) t/T} dt = \overline{y_{-n}}. \end{aligned}$$

Also, if  $g(t) = f(-t)$ , we have that

$$\begin{aligned} \frac{1}{T} \int_0^T g(t) e^{-2\pi i n t / T} dt &= \frac{1}{T} \int_0^T f(-t) e^{-2\pi i n t / T} dt = -\frac{1}{T} \int_0^{-T} f(t) e^{2\pi i n t / T} dt \\ &= \frac{1}{T} \int_0^T f(t) e^{2\pi i n t / T} dt = \overline{y_n}. \end{aligned}$$

The first part of property 2 follows from this. The second part follows directly by noting that

$$y_n e^{2\pi i n t / T} + y_{-n} e^{-2\pi i n t / T} = y_n (e^{2\pi i n t / T} + e^{-2\pi i n t / T}) = 2y_n \cos(2\pi n t / T),$$

or by invoking Theorem 1.10. Property 3 is proved in a similar way. To prove property 4, we observe that the Fourier coefficients of  $g(t) = f(t - d)$  are

$$\begin{aligned} \frac{1}{T} \int_0^T g(t) e^{-2\pi i n t / T} dt &= \frac{1}{T} \int_0^T f(t - d) e^{-2\pi i n t / T} dt \\ &= \frac{1}{T} \int_0^T f(t) e^{-2\pi i n (t+d) / T} dt \\ &= e^{-2\pi i n d / T} \frac{1}{T} \int_0^T f(t) e^{-2\pi i n t / T} dt = e^{-2\pi i n d / T} y_n. \end{aligned}$$

For property 5 we observe that the Fourier coefficients of  $g(t) = e^{2\pi i d t / T} f(t)$  are

$$\begin{aligned} \frac{1}{T} \int_0^T g(t) e^{-2\pi i n t / T} dt &= \frac{1}{T} \int_0^T e^{2\pi i d t / T} f(t) e^{-2\pi i n t / T} dt \\ &= \frac{1}{T} \int_0^T f(t) e^{-2\pi i (n-d) t / T} dt = y_{n-d}. \end{aligned}$$

If  $f(d + t) = f(d - t)$  for all  $t$ , we define the function  $g(t) = f(t + d)$  which is symmetric about 0, so that it has real Fourier coefficients. But then the Fourier coefficients of  $f(t) = g(t - d)$  are  $e^{-2\pi i n d / T}$  times the (real) Fourier coefficients of  $g$  by property 4. It follows that  $y_n$ , the Fourier coefficients of  $f$ , has argument  $-2\pi n d / T$ . The proof in the other direction follows by noting that any function where the Fourier coefficients are real must be symmetric about 0, once the Fourier series is known to converge. This proves property 6.  $\square$

Let us analyze these properties, to see that they match the notion we already have for frequencies and sound. We will say that two sounds “essentially are the same” if the absolute values of each Fourier coefficient are equal. Note that this does not mean that the sounds sound the same, it merely says that the contributions at different frequencies are comparable.



The first property says that the positive and negative frequencies in a (real) sound essentially are the same. The second says that, when we play a sound backwards, the frequency content is essentially the same. This is certainly the case for all pure sounds. The third property says that, if we delay a sound, the frequency content also is essentially the same. This also matches our intuition on sound, since we think of the frequency representation as something which is time-independent. The fourth property says that, if we multiply a sound with a pure tone, the frequency representation is shifted (delayed), according to the value of the frequency. This is something we see in early models for the transmission of audio, where an audio signal is transmitted after having been multiplied with what is called a ‘carrier wave’. You can think of the carrier signal as a pure tone. The result is a signal where the frequencies have been shifted with the frequency of the carrier wave. The point of shifting the frequency of the transmitted signal is to make it use a frequency range in which one knows that other signals do not interfere. The last property looks a bit mysterious. We will not have use for this property before the next chapter.

From Theorem 1.17 we also see that there exist several cases of duality between a function and its Fourier series:

- Delaying a function corresponds to multiplying the Fourier coefficients with a complex exponential. Vice versa, multiplying a function with a complex exponential corresponds to delaying the Fourier coefficients.
- Symmetry/antisymmetry for a function corresponds to the Fourier coefficients being real/purely imaginary. Vice versa, a function which is real has Fourier coefficients which are conjugate symmetric.

Actually, one can show that these dualities are even stronger if we had considered Fourier series of complex functions instead of real functions. We will not go into this.

### 1.4.1 Rate of convergence for Fourier series

We have earlier mentioned criteria which guarantee that the Fourier series converges. Another important topic is the rate of convergence, given that it actually converges. If the series converges quickly, we may only need a few terms in the Fourier series to obtain a reasonable approximation. We have already seen examples which illustrate different convergence rates: The square wave seemed to have very slow convergence rate near the discontinuities, while the triangle wave did not seem to have the same problem.

Before discussing results concerning convergence rates we consider a simple lemma which will turn out to be useful.

**Lemma 1.18.** *The order of computing Fourier series and differentiation does not matter.*

Assume that  $f$  is differentiable. Then  $(f_N)'(t) = (f')_N(t)$ . In other words, the derivative of the Fourier series equals the Fourier series of the derivative.

*Proof.* We first compute

$$\begin{aligned}\langle f, e^{2\pi int/T} \rangle &= \frac{1}{T} \int_0^T f(t) e^{-2\pi int/T} dt \\ &= \frac{1}{T} \left( \left[ -\frac{T}{2\pi in} f(t) e^{-2\pi int/T} \right]_0^T + \frac{T}{2\pi in} \int_0^T f'(t) e^{-2\pi int/T} dt \right) \\ &= \frac{T}{2\pi in} \frac{1}{T} \int_0^T f'(t) e^{-2\pi int/T} dt = \frac{T}{2\pi in} \langle f', e^{2\pi int/T} \rangle.\end{aligned}$$

where we used integration by parts, and that  $-\frac{T}{2\pi in} f(t) e^{-2\pi int/T}$  are periodic with period  $T$ . It follows that  $\langle f, e^{2\pi int/T} \rangle = \frac{T}{2\pi in} \langle f', e^{2\pi int/T} \rangle$ . From this we get that

$$\begin{aligned}(f_N)'(t) &= \left( \sum_{n=-N}^N \langle f, e^{2\pi int/T} \rangle e^{2\pi int/T} \right)' = \frac{2\pi in}{T} \sum_{n=-N}^N \langle f, e^{2\pi int/T} \rangle e^{2\pi int/T} \\ &= \sum_{n=-N}^N \langle f', e^{2\pi int/T} \rangle e^{2\pi int/T} = (f')_N(t).\end{aligned}$$

where we substituted the connection between the inner products we just found.  $\square$

### 1.4.2 Differentiating Fourier series

The connection between the Fourier series of the function and its derivative can be used to simplify the computation of Fourier series for new functions. Let us see how we can use this to compute the Fourier series of the triangle wave, which was quite a tedious job in Example 1.13. However, the relationship  $f'_t(t) = \frac{4}{T} f_s(t)$  is straightforward to see from the plots of the square wave  $f_s$  and the triangle wave  $f_t$ . From this relationship and from Equation (1.11) for the Fourier series of the square wave it follows that

$$((f_t)')_N(t) = \frac{4}{T} \left( \frac{4}{\pi} \sin(2\pi t/T) + \frac{4}{3\pi} \sin(2\pi 3t/T) + \frac{4}{5\pi} \sin(2\pi 5t/T) + \dots \right).$$

If we integrate this we obtain

$$(f_t)_N(t) = -\frac{8}{\pi^2} \left( \cos(2\pi t/T) + \frac{1}{3^2} \cos(2\pi 3t/T) + \frac{1}{5^2} \cos(2\pi 5t/T) + \dots \right) + C.$$

What remains is to find the integration constant  $C$ . This is simplest found if we set  $t = T/4$ , since then all cosine terms are 0. Clearly then  $C = 0$ , and we

arrive at the same expression as in Equation (1.12) for the Fourier series of the triangle wave. This approach clearly had less computations involved. There is a minor point here which we have not addressed: the triangle wave is not differentiable at two points, as required by Lemma 1.18. It is, however, not too difficult to see that this result still holds in cases where we have a finite number of nondifferentiable points only.

We get the following corollary to Lemma 1.18:

**Corollary 1.19.** *Connection between the Fourier coefficients of  $f(t)$  and  $f'(t)$ .*

If the complex Fourier coefficients of  $f$  are  $y_n$  and  $f$  is differentiable, then the Fourier coefficients of  $f'(t)$  are  $\frac{2\pi in}{T}y_n$ .

If we turn this around, we note that the Fourier coefficients of  $f(t)$  are  $T/(2\pi in)$  times those of  $f'(t)$ . If  $f$  is  $s$  times differentiable, we can repeat this argument to show that the Fourier coefficients of  $f(t)$  are  $(T/(2\pi in))^s$  times those of  $f^{(s)}(t)$ . In other words, the Fourier coefficients of a function which is many times differentiable decay to zero very fast.

**Observation 1.20.** *Convergence speed of differentiable functions.*

The Fourier series converges quickly when the function is many times differentiable.

An illustration is found in examples 1.12 and 1.13, where we saw that the Fourier series coefficients for the triangle wave converged more quickly to zero than those of the square wave. This is explained by the fact that the square wave is discontinuous, while the triangle wave is continuous with a discontinuous first derivative. Also, the functions considered in examples 1.24 and 1.25 are not continuous, which partially explain why we there saw contributions from many frequencies.

The requirement of continuity in order to obtain quickly converging Fourier series may seem like a small problem. However, often the function is not defined on the whole real line: it is often only defined on the interval  $[0, T)$ . If we extend this to a periodic function on the whole real line, by repeating one period as shown in the left plot in Figure 1.10, there is no reason why the new function should be continuous at the boundaries  $0, T, 2T$  etc., even though the function we started with may be continuous on  $[0, T)$ . This would require that  $f(0) = \lim_{t \rightarrow T} f(t)$ . If this does not hold, the function may not be well approximated with trigonometric functions, due to a slowly convergence Fourier series.

We can therefore ask ourselves the following question:

**Idea 1.21.** *Continuous Extension.*

Assume that  $f$  is continuous on  $[0, T)$ . Can we construct another periodic function which agrees with  $f$  on  $[0, T]$ , and which is both continuous and periodic (maybe with period different from  $T$ )?

If this is possible the Fourier series of the new function could produce better approximations for  $f$ . It turns out that the following extension strategy does the job:

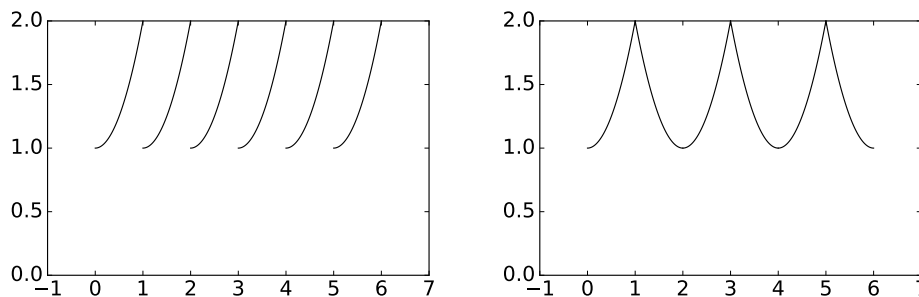


Figure 1.10: Two different extensions of  $f$  to a periodic function on the whole real line. Periodic extension (left) and symmetric extension (right).

**Definition 1.22.** *Symmetric extension of a function.*

Let  $f$  be a function defined on  $[0, T]$ . By the *symmetric extension* of  $f$ , denoted  $\check{f}$ , we mean the function defined on  $[0, 2T]$  by

$$\check{f}(t) = \begin{cases} f(t), & \text{if } 0 \leq t \leq T; \\ f(2T - t), & \text{if } T < t \leq 2T. \end{cases}$$

Clearly the following holds:

**Theorem 1.23.** *Continuous Extension.*

If  $f$  is continuous on  $[0, T]$ , then  $\check{f}$  is continuous on  $[0, 2T]$ , and  $\check{f}(0) = \check{f}(2T)$ . If we extend  $\check{f}$  to a periodic function on the whole real line (which we also will denote by  $\check{f}$ ), this function is continuous, agrees with  $f$  on  $[0, T]$ , and is a symmetric function.

This also means that the Fourier series of  $\check{f}$  is a cosine series, so that it is determined by the cosine-coefficients  $a_n$ . The symmetric extension of  $f$  is shown in the right plot in Figure 1.10.  $\check{f}$  is symmetric since, for  $0 \leq t \leq T$ ,

$$\check{f}(-t) = \check{f}(2T - t) = f(2T - (2T - t)) = f(t) = \check{f}(t).$$

In summary, we now have two possibilities for approximating a function  $f$  defined only on  $[0, T]$ , where the latter addresses a shortcoming of the first:

- By the Fourier series of  $f$
- By the Fourier series of  $\check{f}$  restricted to  $[0, T]$  (which actually is a cosine-series)

**Example 1.35: Periodic extension**

Let  $f$  be the function with period  $T$  defined by  $f(t) = 2t/T - 1$  for  $0 \leq t < T$ . In each period the function increases linearly from  $-1$  to  $1$ . Because  $f$  is discontinuous at the boundaries, we would expect the Fourier series to converge slowly. The Fourier series is a sine-series since  $f$  is antisymmetric, and we can compute  $b_n$  as

$$\begin{aligned} b_n &= \frac{2}{T} \int_0^T \frac{2}{T} \left( t - \frac{T}{2} \right) \sin(2\pi nt/T) dt = \frac{4}{T^2} \int_0^T \left( t - \frac{T}{2} \right) \sin(2\pi nt/T) dt \\ &= \frac{4}{T^2} \int_0^T t \sin(2\pi nt/T) dt - \frac{2}{T} \int_0^T \sin(2\pi nt/T) dt = -\frac{2}{\pi n}, \end{aligned}$$

so that

$$f_N(t) = - \sum_{n=1}^N \frac{2}{n\pi} \sin(2\pi nt/T),$$

which indeed converges slowly to  $0$ . Let us now instead consider the symmetric extension of  $f$ . Clearly this is the triangle wave with period  $2T$ , and the Fourier series of this was

$$(\check{f})_N(t) = - \sum_{n \leq N, n \text{ odd}} \frac{8}{n^2 \pi^2} \cos(2\pi nt/(2T)).$$

The second series clearly converges faster than the first, since its Fourier coefficients are  $a_n = -8/(n^2 \pi^2)$  (with  $n$  odd), while the Fourier coefficients in the first series are  $b_n = -2/(n\pi)$ .

If we use  $T = 1/440$ , the symmetric extension has period  $1/220$ , which gives a triangle wave where the first term in the Fourier series has frequency  $220\text{Hz}$ . Listening to this we should hear something resembling a  $220\text{Hz}$  pure tone, since the first term in the Fourier series is the most dominating in the triangle wave. Listening to the periodic extension we should hear a different sound. The first term in the Fourier series has frequency  $440\text{Hz}$ , but this drowns a bit in the contribution of the other terms in the Fourier series, due to the slow convergence of the Fourier series, just as for the square wave.

Let us plot the Fourier series with  $N = 7$  terms for  $f$ . These are shown in Figure 1.11.

It is clear from the plot that the Fourier series for  $f$  itself is not a very good approximation, while we cannot differentiate between the Fourier series and the function itself for the symmetric extension.

**Exercise 1.36: Fourier series of a delayed square wave**

Define the function  $f$  with period  $T$  on  $[-T/2, T/2)$  by

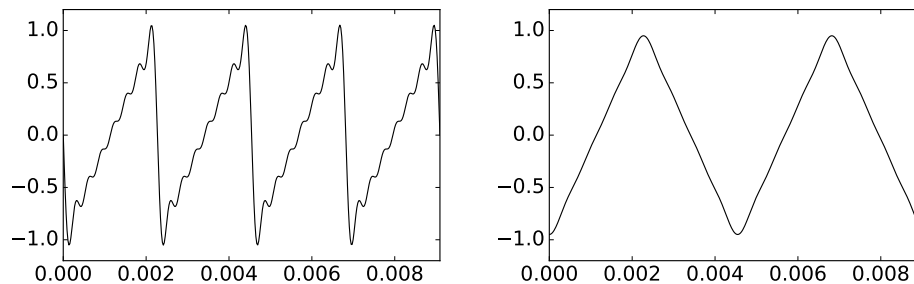


Figure 1.11: The Fourier series with  $N = 7$  terms of the periodic (left) and symmetric (right) extensions of the function in Example 1.35.

$$f(t) = \begin{cases} 1, & \text{if } -T/4 \leq t < T/4; \\ -1, & \text{if } T/4 \leq |t| < T/2. \end{cases}$$

$f$  is just the square wave, delayed with  $d = -T/4$ . Compute the Fourier coefficients of  $f$  directly, and use Property 4 in Theorem 1.17 to verify your result.

### Exercise 1.37: Find function from its Fourier series

Find a function  $f$  which has the complex Fourier series

$$\sum_{n \text{ odd}} \frac{4}{\pi(n+4)} e^{2\pi i n t / T}.$$

**Hint.** Attempt to use one of the properties in Theorem 1.17 on the Fourier series of the square wave.

### Exercise 1.38: Relation between complex Fourier coefficients of $f$ and cosine-coefficients of $\check{f}$

Show that the complex Fourier coefficients  $y_n$  of  $f$ , and the cosine-coefficients  $a_n$  of  $\check{f}$  are related by  $a_{2n} = y_n + y_{-n}$ . This result is not enough to obtain the entire Fourier series of  $\check{f}$ , but at least it gives us half of it.

## 1.5 Operations on sound: filters

It is easy to see how we can use Fourier coefficients to analyse or improve sound: Noise in a sound often corresponds to the presence of some high frequencies with large coefficients, and by removing these, we remove the noise. For example, we could set all the coefficients except the first one to zero. This would change, as

we have seen, the unpleasant square wave to the pure tone  $\sin(2\pi 440t)$ . Doing so is an example of an important operation on sound called a *filter*:

**Definition 1.24.** *Analog filters.*

An operation on sound is called a *filter* if it preserves the different frequencies in the sound. In other words,  $s$  is a filter if, for any sound on the form  $f = \sum_{\nu} c(\nu)e^{2\pi i\nu t}$ , the output  $s(f)$  is a sound which can be written on the form

$$s(f) = s\left(\sum_{\nu} c(\nu)e^{2\pi i\nu t}\right) = \sum_{\nu} c(\nu)\lambda_s(\nu)e^{2\pi i\nu t}.$$

$\lambda_s(\nu)$  is a function describing how  $s$  treats the different frequencies, and is also called the *frequency response* of  $s$ .

By definition any pure tone is an eigenvector of  $s$ , with the frequency response providing the eigenvalue. The notion of a filter makes sense for both periodic and non-periodic input functions. The problem is, however, that a function may be an infinite sum of frequencies, for which a sum on the form  $\sum_{\nu} c(\nu)e^{2\pi i\nu t}$  may not converge.

This general definition of filters may not be useful in practice, but if we restrict to Fourier spaces, however, we restrict ourselves to finite sums. We then clearly have that  $s(f) \in V_{N,T}$  whenever  $f \in V_{N,T}$ , so that the computation can be performed in finite dimensions. Let us now see how we can construct useful such filters.

**Theorem 1.25.** *Convolution kernels.*

Assume that  $g$  is a bounded Riemann-integrable function with compact support (i.e. that there exists an interval  $[a, b]$  so that  $g = 0$  outside  $[a, b]$ ). The operation

$$f(t) \rightarrow h(t) = \int_{-\infty}^{\infty} g(u)f(t-u)du. \quad (1.23)$$

is a filter. Also, the frequency response of the filter is  $\lambda_s(\nu) = \int_{-\infty}^{\infty} g(s)e^{-2\pi i\nu s}ds$ . The function  $g$  is also called the *kernel* of  $s$ .

Note that the requirement that  $g$  is bounded with compact support is just made for convenience, to ensure that the integral exists. Many weaker conditions can be put on  $g$  to ensure that the integral exists. In case of compact support there exist constants  $a$  and  $b$  so that the filter takes the form  $f(t) \rightarrow h(t) = \int_a^b g(s)f(t-s)ds$ .

*Proof.* We compute

$$s(e^{2\pi i\nu t}) = \int_{-\infty}^{\infty} g(s)e^{2\pi i\nu(t-s)}ds = \int_{-\infty}^{\infty} g(s)e^{-2\pi i\nu s}dse^{2\pi i\nu t} = \lambda_s(f)e^{2\pi i\nu t},$$

which shows that  $s$  is a filter with the stated frequency response.  $\square$

The function  $g$  is arbitrary, so that this strategy leads to a wide class of analog filters. We may ask the question of whether the general analog filter always has this form. We will not go further into this, although one can find partially affirmative answers to this question.

### Exercise 1.39: Filters preserve sine- and cosine-series

An analog filter where  $\lambda_s(\nu) = \lambda_s(-\nu)$  is also called a *symmetric filter*.

- a) Prove that, if the input to a symmetric filter is a Fourier series which is a cosine series/sine-series, then the output also is a cosine/sine series.
- b) Show that  $s(f) = \int_{-a}^a g(s)f(t-s)ds$  is a symmetric filter whenever  $g$  is symmetric around 0 and supported on  $[-a, a]$ .

We saw that the symmetric extension of a function took the form of a cosine-series, and that this converged faster to the symmetric extension than the Fourier series did to the function. If a filter preserves cosine-series it will also preserve symmetric extensions, and therefore also map fast-converging Fourier series to fast-converging Fourier series.

## 1.6 Convergence of Fourier series\*

A major topic in harmonic analysis is finding conditions on  $f$  which secure convergence of its Fourier series. This turns out to be very difficult in general, and depends highly on the mode of convergence. We will cover some important result on this here in this section, which is a bit more technical than the remainder of the book. We will consider both

- *pointwise convergence*, i.e. that  $f_N(t) \rightarrow f(t)$ , and
- convergence in  $\|\cdot\|$ , i.e. that  $\|f_N - f\| \rightarrow 0$ .

The latter unfortunately does not imply the first, which is harder to prove. Although a general theorem about the pointwise convergence of the Fourier series for square-integrable functions exists, this result is way too hard to prove here. Instead we will restrict ourselves to the following class of functions, for which it is possible to state a proof for both modes of convergence. This class also contains most functions we encounter in the book, such as the square wave and the triangle wave:

**Definition 1.26.** *Piecewise continuous functions.*

A  $T$ -periodic function is said to be *piecewise continuous* if there exists a finite set of points

$$0 \leq a_0 < a_1 < \cdots < a_{n-1} < a_n < T$$

so that

1.  $f$  is continuous on each interval between adjacent such points,



2. the one-sided limits  $f(a_i^+) := \lim_{t \rightarrow a_i^+} f(t)$  and  $f(a_i^-) := \lim_{t \rightarrow a_i^-} f(t)$  exist, and
3.  $f(a_i) = \frac{f(a_i^+) + f(a_i^-)}{2}$  (i.e. the value at a "jump" is the average of the one-sided limits).

For piecewise continuous functions, convergence in  $\|\cdot\|$  for the Fourier series will follow from the following theorem.

**Theorem 1.27.** *Approximating piecewise continuous functions.*

Let  $f$  be piecewise continuous. We can find a sequence  $S_N \in V_{N,T}$ ,  $N \geq 1$  so that  $S_N(t) \rightarrow f(t)$  for all  $t$ . Also,  $S_N \rightarrow f$  uniformly as  $N \rightarrow \infty$  on any interval  $[a, b]$  where  $f$  is continuous.

The functions  $g_N$  are found in a constructive way in the proof of this theorem, but note that these are not the same as the Fourier series  $f_N$ ! Therefore, the theorem says nothing about the convergence of the Fourier series itself.

*Proof.* In the proof we will use the concept of a *summability kernel*, which is a sequence of functions  $k_N$  defined on  $[0, T]$  so that

- $\frac{1}{T} \int_0^T k_N(t) dt = 1$ ,
- there exists a constant  $C$  so that  $\frac{1}{T} \int_0^T |k_N(t)| dt \leq C$  for all  $N$ ,
- For all  $0 < \delta < T/2$ ,  $\lim_{n \rightarrow \infty} \frac{1}{T} \int_\delta^{T-\delta} |k_N(t)| dt = 0$ .

Note that if  $k_N$  is a trigonometric polynomial, then  $\frac{1}{T} \int_0^T k_N(\tau) f(t - \tau) d\tau$  is a trigonometric polynomial of the same degree (Make the substitution  $u = t - \tau$  to verify this). In Exercise 1.42 you are aided through the construction of one important such summability kernel, denoted  $F_N$  and called the *Fejer kernel*. The Fejer kernel has the following additional properties.

- $F_N$  is a trigonometric polynomial of degree  $N$ ,
- $0 \leq F_N(t) \leq \frac{T^2}{4(N+1)t^2}$ .

We now set  $S_N(t) = \frac{1}{T} \int_0^T F_N(u) f(t - u) du$ . Through the change of variables  $t = -u$  it easily follows that  $S_N(t) = \frac{1}{T} \int_0^T F_N(u) f(t + u) du$  as well. We can thus write

$$S_N(t) = \frac{1}{2T} \int_0^T (f(t + u) + f(t - u)) F_N(u) du.$$

Since also  $f(t) = \frac{1}{T} \int_0^T f(t) F_N(u) du$ , it follows that

$$S_N(t) - f(t) = \frac{1}{2T} \int_0^T (f(t + u) + f(t - u) - 2f(t)) F_N(u) du. \quad (1.24)$$

We have written  $S_N(t) - f(t)$  on this form since now the integrand is continuous at  $u = 0$  as a function of  $u$ : This is obvious if  $f$  is continuous at  $t$ . If on the other hand  $t$  is one of the discontinuities, we have that  $f(t+u) + f(t-u) \rightarrow 2f(t)$  as  $u \rightarrow 0$ . Given  $\epsilon > 0$ , we can therefore find a  $\delta > 0$  so that

$$|f(t+u) + f(t-u) - 2f(t)| < \epsilon$$

whenever  $|u| < \delta$ . Now, split the integral (1.24) in three:  $\int_0^T = \int_{-T/2}^{-\delta} + \int_{-\delta}^{\delta} + \int_{\delta}^{T/2}$ . For the second of these we have

$$\begin{aligned} & \frac{1}{2T} \int_{-\delta}^{\delta} |(f(t+u) + f(t-u) - 2f(t))F_N(u)| du \\ & \leq \frac{1}{2T} \int_{-\delta}^{\delta} \epsilon F_n(u) du \leq \frac{1}{2T} \int_{-T/2}^{T/2} \epsilon F_n(u) du = \frac{\epsilon}{2}. \end{aligned}$$

For the third of these we have

$$\begin{aligned} & \frac{1}{2T} \int_{\delta}^{T/2} |(f(t+u) + f(t-u) - 2f(t))F_N(u)| du \\ & \leq \frac{1}{2T} \int_{\delta}^{T/2} 4\|f\|_{\infty} \frac{T^2}{4(N+1)u^2} du \leq \frac{\|f\|_{\infty} T^2}{4(N+1)\delta^2}, \end{aligned}$$

where  $\|f\|_{\infty} = \max_{x \in [0, T]} |f(x)|$ . A similar calculation can be done for the first integral. Clearly then we can choose  $N$  so big that the sum of the first and third integrals are less than  $\epsilon/2$ , and we then get that  $|S_N(t) - f(t)| < \epsilon$ . This shows that  $S_N(t) \rightarrow f(t)$  as  $N \rightarrow \infty$  for any  $t$ . For the final statement, if  $[a, b]$  is an interval where  $f$  is continuous, choose the  $\delta$  above so small that  $[a - \delta, b + \delta]$  still contains no discontinuities. Since continuous functions are uniformly continuous on compact intervals, it is not too hard to see that the convergence of  $f$  to  $S_N$  on  $[a, b]$  is uniform. This completes the proof.  $\square$

Since  $S_N(t) = \frac{1}{T} \int_0^T f(t-u)F_N(u) du \in V_{N, T}$ , and  $f_N$  is a best approximation from  $V_{N, T}$ , we have that  $\|f_N - f\| \leq \|S_N - f\|$ . If  $f$  is continuous, the result says that  $\|f - S_N\|_{\infty} \rightarrow 0$ , which implies that  $\|f - S_N\| \rightarrow 0$ , so that  $\|f - f_N\| \rightarrow 0$ . Therefore, for  $f$  continuous, both  $\|f - S_n\|_{\infty} \rightarrow 0$  and  $\|f - f_N\| \rightarrow 0$  hold, so that we have established both modes of convergence. If  $f$  has a discontinuity, it is obvious that  $\|f - S_N\|_{\infty} \rightarrow 0$  can not hold, since  $S_N$  is continuous.  $\|f - f_N\| \rightarrow 0$  holds, however, even with discontinuities. The reason is that any function with only a finite number of discontinuities can be approximated arbitrarily well with continuous functions w.r.t.  $\|\cdot\|$ . The proof of this is left as an exercise.

Both the square wave and the triangle are piecewise continuous (at least if we redefined the value of the square wave at the discontinuity). Therefore both their Fourier series converge to  $f$  in  $\|\cdot\|$ . Since the triangle wave is continuous,  $S_N$  also converges uniformly to  $f_t$ .

The result above states that  $S_N$  converges pointwise to  $f$  - it does not say that  $f_N$  converges pointwise to  $f$ . This suggests that  $S_N$  may be better suited to approximate  $f$ . In Figure 1.12 we have plotted  $S_N(t)$  and  $f_N(t)$  for the square wave. Clearly the approximations are very different. The pointwise convergence of  $f_N(t)$  is more difficult to analyze, so we will make some additional assumptions.

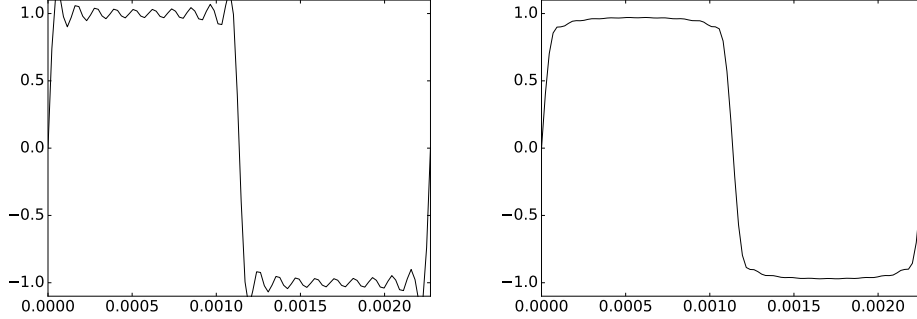


Figure 1.12:  $f_N(t)$  and  $S_N(t)$  for  $N = 20$ .

**Theorem 1.28.** *Pointwise convergence of Fourier series.*

Assume that  $f$  is piecewise continuous and that the one-sided limits

$$D_+f(t) = \lim_{h \rightarrow 0^+} \frac{f(t+h) - f(t^+)}{h} \quad D_-f(t) = \lim_{h \rightarrow 0^-} \frac{f(t+h) - f(t^-)}{h}$$

exist. Then  $\lim_{N \rightarrow \infty} f_N(t) = f(t)$ .

*Proof.* In Exercise 1.41 we construct another kernel  $D_N(t)$ , called the Dirichlet kernel. This satisfies only two of the properties of a summability kernel, but this will turn out to be enough for our purposes due to the additional assumption on the one-sided limits for the derivative. A formula similar to (1.24) can be easily proved using the same substitution  $v = -u$ :

$$f_N(t) - f(t) = \frac{1}{2T} \int_0^T (f(t+u) + f(t-u) - 2f(t)) D_N(u) du. \quad (1.25)$$

Substituting the expression for the Dirichlet kernel obtained in exercise 1.42, the integrand can be written as

$$\begin{aligned} & f(t+u) - f(t^+) + f(t-u) - f(t^-) D_N(u) \\ &= \left( \frac{f(t+u) - f(t^+)}{\sin(\pi u/T)} + \frac{f(t-u) - f(t^-)}{\sin(\pi u/T)} \right) \sin(\pi(2N+1)u/T) \\ &= h(u) \sin(\pi(2N+1)u/T). \end{aligned}$$

We have that

$$\begin{aligned} \frac{f(t+u) - f(t^+)}{\sin(\pi u/T)} &= \frac{f(t+u) - f(t^+)}{\pi u/T} \frac{\pi u/T}{\sin(\pi u/T)} \\ &\rightarrow \begin{cases} \frac{T}{\pi} D_+ f(t) & \text{when } u \rightarrow 0^+, \\ \frac{T}{\pi} D_- f(t) & \text{when } u \rightarrow 0^-, \end{cases} \end{aligned}$$

and similarly for  $\frac{f(t-u) - f(t^-)}{\sin(\pi u/T)}$ . It follows that the function  $h$  defined above is a piecewise continuous function in  $u$ . The proof will be done if we can show that  $\int_0^T h(u) \sin(\pi(2N+1)u/T) dt \rightarrow 0$  as  $N \rightarrow \infty$  for any piecewise continuous  $h$ . Since

$$\sin(\pi(2N+1)u/T) = \sin(2\pi Nu/T) \cos(\pi u/T) + \cos(2\pi Nu/T) \sin(\pi u/T),$$

and since  $h(u) \cos(\pi u/T)$  and  $h(u) \sin(\pi u/T)$  also are piecewise continuous, it is enough to show that  $\int h(u) \sin(2\pi Nu/T) du \rightarrow 0$  and  $\int h(u) \cos(2\pi Nu/T) du \rightarrow 0$ . These are simply the order  $N$  Fourier coefficients of  $h$ . Since  $h$  is in particular square integrable, it follows from Bessel's inequality (Theorem 1.14) that the Fourier coefficients of  $h$  go to zero, and the proof is done.  $\square$

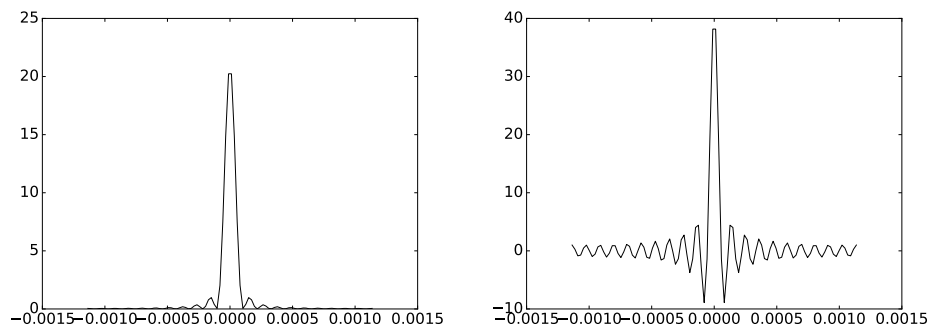
The requirement on the one-sided limits of the derivative above can be replaced by less strict conditions. This gives rise to what is known as *Dini's test*. One can also replace with the less strict requirement that  $f$  has a finite number of local minima and maxima. This is referred to as Dirichlet's theorem, after Dirichlet who proved it in 1829. There also exist much more general conditions that secure pointwise convergence of the Fourier series. The most general results require deep mathematical theory to prove.

Both the square wave and the triangle wave have one-sided limits for the derivative. Therefore both their Fourier series converge to  $f$  pointwise.

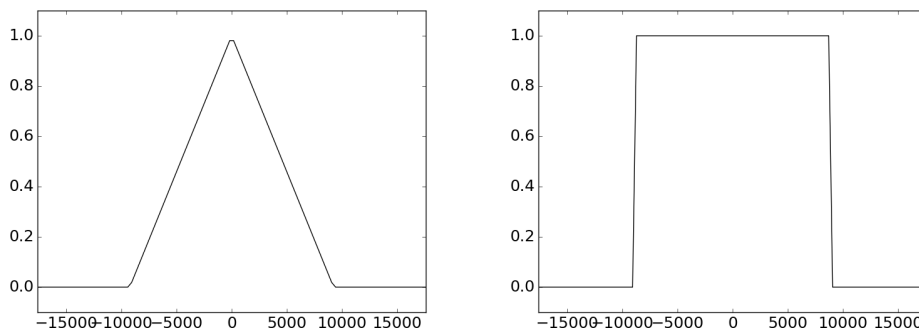
### 1.6.1 Interpretation in terms of filters

It is instructive to interpret Theorem 1.27 and 1.28 in terms of filters. There are filters at play here, and their kernels are the Fejer kernel and the Dirichlet kernel. The kernels are shown in Figure 1.13

For the Fejer kernel, we saw that  $S_N(t) = \frac{1}{T} \int_0^T F_N(u) f(t-u) du$ . So, if  $s_N$  is the filter with kernel  $F_N$ , then  $S_N = s_N(f)$ . It is shown in exercise 1.42 that  $S_N(t) = \frac{1}{N+1} \sum_{n=0}^N f_n(t)$ , also called the *Cesaro mean* of the Fourier series. Since the  $N$ 'th order Fourier series of  $e^{2\pi int/T}$  is 0 if  $|n| > |N|$ , and  $e^{2\pi int/T}$  if not, it follows that  $s_N(e^{2\pi int/T}) = \left(1 - \frac{|n|}{N+1}\right) e^{2\pi int/T}$ . In other words, the frequency response for filtering with the Fejer kernel  $F_N$  is given by the mapping  $n/T \rightarrow \left(1 - \frac{|n|}{N+1}\right)$ . On  $[-N/T, N/T]$ , this first increases linearly to 1, then decreases linearly back to 0. Outside  $[-N/T, N/T]$  we get zero.

Figure 1.13: The Fejer and Dirichlet kernels for  $N = 20$ .

For the Dirichlet kernel we saw that  $f_N(t) = \frac{1}{T} \int_0^T D_N(u)f(t-u)du$ . From this it follows in the same way that the frequency response corresponding to filtering with the Dirichlet kernel is given by the mapping  $n/T \rightarrow 1$ , i.e. it is one on  $[-N/T, N/T]$  and 0 elsewhere.

Figure 1.14: The frequency responses for the filters with Fejer and Dirichlet kernels,  $N = 20$ .

The two frequency responses are shown in Figure 1.14. Both filters above are what is called *lowpass filters*: They annihilate high frequencies. More precisely, if  $\nu > |N/T|$ , then the frequency response of  $\nu$  is zero. The lowest frequency  $\nu = 0$  is treated in the same way by the two filters, but the higher frequencies are differed: The Dirichlet kernel keeps them, while the Fejer kernel attenuates them, i.e. does not include all the frequency content at the higher frequencies. That filtering with the Fejer kernel gave something ( $S_N(t)$ ) with better convergence properties can be interpreted as follows: We should be careful when we include the contribution from the higher frequencies, as this may affect the convergence.

**Exercise 1.40: Approximation in norm with continuous functions**

Show that if  $f$  is a function with only a finite number of discontinuities, there exists a continuous function  $g$  so that  $\|f - g\| < \epsilon$ .

**Exercise 1.41: The Dirichlet kernel**

The Dirichlet kernel is defined as

$$D_N(t) = \sum_{n=-N}^N e^{2\pi i n t/T} = 1 + 2 \sum_{n=1}^N \cos(2\pi n t/T).$$

$D_N$  is clearly trigonometric, and of degree  $N$ .

- a) Show that  $D_N(t) = \frac{\sin(\pi(2N+1)t/T)}{\sin(\pi t/T)}$ .
- b) Show that  $f_N(t) = \frac{1}{T} \int_0^T f(t-u) D_N(u) du$ . Proving that  $\lim_{N \rightarrow \infty} f_N(t) = f(t)$  is thus equivalent to  $\lim_{N \rightarrow \infty} \frac{1}{T} \int_0^T f(t-u) D_N(u) du = f(t)$ .
- c) Prove that  $D_N(t)$  satisfies only two of the properties of a summability kernel.
- d) Write a function which takes  $N$  and  $T$  as arguments, and plots  $D_N(t)$  over  $[-T/2, T/2]$ .

**Exercise 1.42: The Fejer summability kernel**

The Fejer kernel is defined as

$$F_N(t) = \sum_{n=-N}^N \left(1 - \frac{|n|}{N+1}\right) e^{2\pi i n t/N}.$$

$F_N$  is clearly trigonometric, and of degree  $N$ .

- a) Show that  $F_N(t) = \frac{1}{N+1} \left(\frac{\sin(\pi(N+1)t/T)}{\sin(\pi t/T)}\right)^2$ , and conclude from this that  $0 \leq F_N(t) \leq \frac{T^2}{4(N+1)t^2}$ .

**Hint.** Use that  $\frac{2}{\pi}|u| \leq |\sin u|$  when  $u \in [-\pi/2, \pi/2]$ .

- b) Show that  $F_N(t)$  satisfies the three properties of a summability kernel.
- c) Show that  $\frac{1}{T} \int_0^T f(t-u) F_N(u) du = \frac{1}{N+1} \sum_{n=0}^N f_n$ .

**Hint.** Show that  $F_N(t) = \frac{1}{N+1} \sum_{n=0}^N D_n(t)$ , and use Exercise 1.41 b).

- d) Write a function which takes  $N$  and  $T$  as arguments, and plots  $F_N(t)$  over  $[-T/2, T/2]$ .

## 1.7 The MP3 standard

Digital audio first became commonly available when the CD was introduced in the early 1980s. As the storage capacity and processing speeds of computers increased, it became possible to transfer audio files to computers and both play and manipulate the data, in ways such as in the previous section. However, audio was represented by a large amount of data and an obvious challenge was how to reduce the storage requirements. Lossless coding techniques like Huffman and Lempel-Ziv coding were known and with these kinds of techniques the file size could be reduced to about half of that required by the CD format. However, by allowing the data to be altered a little bit it turned out that it was possible to reduce the file size down to about ten percent of the CD format, without much loss in quality. The MP3 audio format takes advantage of this.

MP3, or more precisely *MPEG-1 Audio Layer 3*, is part of an audio-visual standard called MPEG. MPEG has evolved over the years, from MPEG-1 to MPEG-2, and then to MPEG-4. The data on a DVD disc can be stored with either MPEG-1 or MPEG-2, while the data on a bluray-disc can be stored with either MPEG-2 or MPEG-4. MP3 was developed by Philips, CCETT (Centre commun d'études de television et telecommunications), IRT (Institut für Rundfunktechnik) and Fraunhofer Society, and became an international standard in 1991. Virtually all audio software and music players support this format. MP3 is just a sound format. It leaves a substantial amount of freedom in the encoder, so that different encoders can exploit properties of sound in various ways, in order to alter the sound in removing inaudible components therein. As a consequence there are many different MP3 encoders available, of varying quality. In particular, an encoder which works well for higher bit rates (high quality sound) may not work so well for lower bit rates.

With MP3, the sound is split into *frequency bands*, each band corresponding to a particular frequency range. In the simplest model, 32 frequency bands are used. A frequency analysis of the sound, based on what is called a *psycho-acoustic model*, is the basis for further transformation of these bands. The psycho-acoustic model computes the significance of each band for the human perception of the sound. When we hear a sound, there is a mechanical stimulation of the ear drum, and the amount of stimulus is directly related to the size of the sample values of the digital sound. The movement of the ear drum is then converted to electric impulses that travel to the brain where they are perceived as sound. The perception process uses a transformation of the sound so that a steady oscillation in air pressure is perceived as a sound with a fixed frequency. In this process certain kinds of perturbations of the sound are hardly noticed by the brain, and this is exploited in lossy audio compression.

More precisely, when the psycho-acoustic model is applied to the frequency content resulting from our frequency analysis, *scale factors* and *masking thresholds* are assigned for each band. The computed masking thresholds have to do with a phenomenon called *masking*. A simple example of this is that a loud sound will make a simultaneous low sound inaudible. For compression this means that if certain frequencies of a signal are very prominent, most of the other frequencies

can be removed, even when they are quite large. If the sounds are below the masking threshold, it is simply omitted by the encoder, since the model says that the sound should be inaudible.

Masking effects are just one example of what is called psycho-acoustic effects, and all such effects can be taken into account in a psycho-acoustic model. Another obvious such effect regards computing the scale factors: the human auditory system can only perceive frequencies in the range 20 Hz - 20 000 Hz. An obvious way to do compression is therefore to remove frequencies outside this range, although there are indications that these frequencies may influence the listening experience inaudibly. The computed scaling factors tell the encoder about the precision to be used for each frequency band: If the model decides that one band is very important for our perception of the sound, it assigns a big scale factor to it, so that more effort is put into encoding it by the encoder (i.e. it uses more bits to encode this band).

Using appropriate scale factors and masking thresholds provide compression, since bits used to encode the sound are spent on parts important for our perception. Developing a useful psycho-acoustic model requires detailed knowledge of human perception of sound. Different MP3 encoders use different such models, so they may produce very different results, worse or better.

The information remaining after frequency analysis and using a psycho-acoustic model is coded efficiently with (a variant of) Huffman coding. MP3 supports bit rates from 32 to 320 kb/s and the sampling rates 32, 44.1, and 48 kHz. The format also supports variable bit rates (the bit rate varies in different parts of the file). An MP3 encoder also stores metadata about the sound, such as the title of the audio piece, album and artist name and other relevant data.

MP3 too has evolved in the same way as MPEG, from MP1 to MP2, and to MP3, each one more sophisticated than the other, providing better compression. MP3 is not the latest development of audio coding in the MPEG family: AAC (Advanced Audio Coding) is presented as the successor of MP3 by its principal developer, Fraunhofer Society, and can achieve better quality than MP3 at the same bit rate, particularly for bit rates below 192 kb/s. AAC became well known in April 2003 when Apple introduced this format (at 128 kb/s) as the standard format for their iTunes Music Store and iPod music players. AAC is also supported by many other music players, including the most popular mobile phones.

The technologies behind AAC and MP3 are very similar. AAC supports more sample rates (from 8 kHz to 96 kHz) and up to 48 channels. AAC uses the same transformation as MP3, but AAC processes 1 024 samples at a time. AAC also uses much more sophisticated processing of frequencies above 16 kHz and has a number of other enhancements over MP3. AAC, as MP3, uses Huffman coding for efficient coding of the transformed values. Tests seem quite conclusive that AAC is better than MP3 for low bit rates (typically below 192 kb/s), but for higher rates it is not so easy to differentiate between the two formats. As for MP3 (and the other formats mentioned here), the quality of an AAC file depends crucially on the quality of the encoding program.



There are a number of variants of AAC, in particular AAC Low Delay (AAC-LD). This format was designed for use in two-way communication over a network,

for example the internet. For this kind of application, the encoding (and decoding) must be fast to avoid delays (a delay of at most 20 ms can be tolerated).

## 1.8 Summary

We defined digital sound, and demonstrated how we could perform simple operations on digital sound such as adding noise, playing at different rates e.t.c.. Digital sound could be obtained by sampling continuous sounds.

We discussed the basic question of what is sound is, and concluded that sound could be modeled as a sum of frequency components. If the function was periodic we could define its Fourier series, which can be thought of as an approximation scheme for periodic functions using finite-dimensional spaces of trigonometric functions. We established the basic properties of Fourier series, and some duality relationships between the function and its Fourier series. We have also computed the Fourier series of the square wave and the triangle wave, and we saw that we could speed up the convergence of the Fourier series by instead considering the symmetric extension of the function.

We also discussed the MP3 standard for compression of sound, and its relation to a psychoacoustic model which describes how the human auditory system perceives sound. There exist a wide variety of documents on this standard. In [33], an overview is given, which, although written in a signal processing friendly language and representing most relevant theory such as for the psychoacoustic model, does not dig into all the details.

we also defined analog filters, which were operations which operate on continuous sound, without any assumption on periodicity. In signal processing literature one defines the *Continuous-time Fourier transform*, or CTFT. We will not use this concept in this book. We have instead disguised this concept as the frequency response of an analog filter. To be more precise: in the literature, the CTFT of  $g$  is nothing but the frequency response of an analog filter with  $g$  as convolution kernel.

### What you should have learned in this chapter.

- Computer operations for reading, writing, and listening to sound.
- Construct sounds such as pure tones and the square wave, from mathematical formulas.
- The inner product which we use for function spaces.
- Definition of the Fourier spaces, and the orthogonality of the Fourier basis.
- Fourier series approximations as best approximations.

- Formulas for the Fourier coefficients.
- Using the computer to plot Fourier series, and comparing a sound with its Fourier series.
- For symmetric/antisymmetric functions, Fourier series are actually cosine/sine series.
- The complex Fourier basis and its orthonormality.
- Simple Fourier series pairs.
- Certain properties of Fourier series, for instance how delay of a function or multiplication with a complex exponential affect the Fourier coefficients.
- The convergence rate of a Fourier series depends on the regularity of the function. How this motivates the symmetric extension of a function.

## Chapter 2

# Digital sound and Discrete Fourier analysis

In Chapter 1 we saw how a periodic function can be decomposed into a linear combination of sines and cosines, or equivalently, a linear combination of complex exponential functions. This kind of decomposition is, however, not very convenient from a computational point of view. The coefficients are given by integrals that in most cases cannot be evaluated exactly, so some kind of numerical integration technique needs to be applied. In this chapter we will decompose vectors in terms of linear combinations of complex exponentials. As before it turns out that this is simplest when we assume that the values in the vector repeat periodically. Then a vector of finite dimension can be used to represent all sound values, and a transformation to the frequency domain, where operations which change the sound can easily be made, simply amounts to multiplying the vector by a matrix. This transformation is called the Discrete Fourier transform, and we will see how we can implement this efficiently. It turns out that these algorithms can also be used for computing approximations to the Fourier series, and for sampling a sound in order to create a vector of sound data.

### 2.1 Discrete Fourier analysis and the discrete Fourier transform

In this section we will parallel the developments we did for Fourier series, assuming instead that vectors (rather than functions) are involved. As with Fourier series we will assume that the vector is periodic. This means that we can represent it with the values from only the first period. In the following we will only work with these values, but we will remind ourselves from time to time that the values actually come from a periodic vector. As for functions, we will call denote the periodic vector as the periodic extension of the finite vector. To

illustrate this, we have in Figure 2.1 shown a vector  $\mathbf{x}$  and its periodic extension  $\mathbf{x}$ .

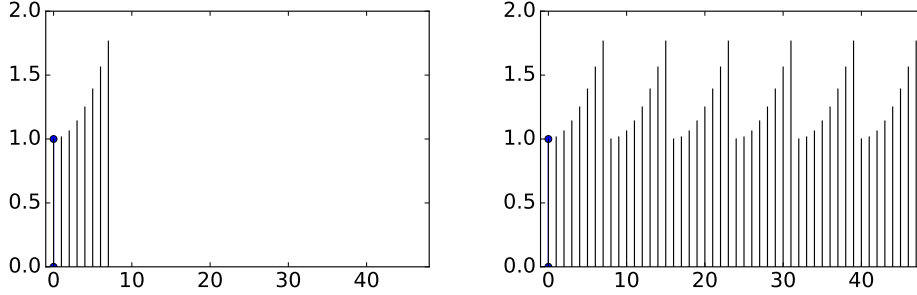


Figure 2.1: A vector and its periodic extension.

At the outset our vectors will have real components, but since we use complex exponentials we must be able to work with complex vectors also. We therefore first need to define the standard inner product and norm for complex vectors.

**Definition 2.1.** *Euclidean inner product.*

For complex vectors of length  $N$  the Euclidean inner product is given by

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{k=0}^{N-1} x_k \overline{y_k}. \quad (2.1)$$

The associated norm is

$$\|\mathbf{x}\| = \sqrt{\sum_{k=0}^{N-1} |x_k|^2}. \quad (2.2)$$

In the previous chapter we saw that, using a Fourier series, a function with period  $T$  could be approximated by linear combinations of the functions (the pure tones)  $\{e^{2\pi i n t/T}\}_{n=0}^N$ . This can be generalized to vectors (digital sounds), but then the pure tones must of course also be vectors.

**Definition 2.2.** *Discrete Fourier analysis.*

In Discrete Fourier analysis, a vector  $\mathbf{x} = (x_0, \dots, x_{N-1})$  is represented as a linear combination of the  $N$  vectors

$$\phi_n = \frac{1}{\sqrt{N}} \left( 1, e^{2\pi i n/N}, e^{2\pi i 2n/N}, \dots, e^{2\pi i k n/N}, \dots, e^{2\pi i n(N-1)/N} \right).$$

These vectors are called the normalised complex exponentials, or the pure digital tones of order  $N$ .  $n$  is also called frequency index. The whole collection  $\mathcal{F}_N = \{\phi_n\}_{n=0}^{N-1}$  is called the  $N$ -point Fourier basis.

Note that pure digital tones can be considered as samples of a pure tone, taken uniformly over one period: If  $f(t) = e^{2\pi i n t/T} / \sqrt{N}$  is the pure tone with

frequency  $n/T$ , then  $f(kT/N) = e^{2\pi i n(kT/N)/T} / \sqrt{N} = e^{2\pi i nk/N} / \sqrt{N} = \phi_n$ . When mapping a pure tone to a digital pure tone, the index  $n$  corresponds to frequency  $\nu = n/T$ , and  $N$  the number of samples taken over one period. Since  $Tf_s = N$ , where  $f_s$  is the sampling frequency, we have the following connection between frequency and frequency index:

$$\nu = \frac{nf_s}{N} \text{ and } n = \frac{\nu N}{f_s} \quad (2.3)$$

The following lemma shows that the vectors in the Fourier basis are orthonormal, so they do indeed form a basis.

**Lemma 2.3.** *Complex exponentials are an orthonormal basis.*

The normalized complex exponentials  $\{\phi_n\}_{n=0}^{N-1}$  of order  $N$  form an orthonormal basis in  $\mathbb{R}^N$ .

*Proof.* Let  $n_1$  and  $n_2$  be two distinct integers in the range  $[0, N-1]$ . The inner product of  $\phi_{n_1}$  and  $\phi_{n_2}$  is then given by

$$\begin{aligned} \langle \phi_{n_1}, \phi_{n_2} \rangle &= \frac{1}{N} \langle e^{2\pi i n_1 k/N}, e^{2\pi i n_2 k/N} \rangle \\ &= \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi i n_1 k/N} e^{-2\pi i n_2 k/N} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi i (n_1 - n_2) k/N} \\ &= \frac{1}{N} \frac{1 - e^{2\pi i (n_1 - n_2)}}{1 - e^{2\pi i (n_1 - n_2)/N}} \\ &= 0. \end{aligned}$$

In particular, this orthogonality means that the complex exponentials form a basis. Clearly also  $\langle \phi_n, \phi_n \rangle = 1$ , so that the  $N$ -point Fourier basis is in fact an orthonormal basis.  $\square$

Note that the normalizing factor  $\frac{1}{\sqrt{N}}$  was not present for pure tones in the previous chapter. Also, the normalizing factor  $\frac{1}{T}$  from the last chapter is not part of the definition of the inner product in this chapter. These are small differences which have to do with slightly different notation for functions and vectors, and which will not cause confusion in what follows.

The focus in Discrete Fourier analysis is to change coordinates from the standard basis to the Fourier basis, performing some operations on this “Fourier representation”, and then change coordinates back to the standard basis. Such operations are of crucial importance, and in this section we study some of their basic properties. We start with the following definition.

**Definition 2.4.** *Discrete Fourier Transform.*

We will denote the change of coordinates matrix from the standard basis of  $\mathbb{R}^N$  to the Fourier basis  $\mathcal{F}_N$  by  $F_N$ . We will also call this the ( $N$ -point) *Fourier matrix*.

The matrix  $\sqrt{N}F_N$  is also called the ( $N$ -point) *discrete Fourier transform*, or DFT. If  $\mathbf{x}$  is a vector in  $\mathbb{R}^N$ , then  $\mathbf{y} = \text{DFT}\mathbf{x}$  are called the DFT coefficients of  $\mathbf{x}$ . (the DFT coefficients are thus the coordinates in  $\mathcal{F}_N$ , scaled with  $\sqrt{N}$ ).  $\text{DFT}\mathbf{x}$  is sometimes written as  $\hat{\mathbf{x}}$ .

Note that we define the Fourier matrix and the DFT as two different matrices, the one being a scaled version of the other. The reason for this is that there are different traditions in different fields. In pure mathematics, the Fourier matrix is mostly used since it is, as we will see, a unitary matrix. In signal processing, the scaled version provided by the DFT is mostly used. We will normally write  $\mathbf{x}$  for the given vector in  $\mathbb{R}^N$ , and  $\mathbf{y}$  for its DFT. In applied fields, the Fourier basis vectors are also called *synthesis vectors*, since they can be used to “synthesize” the vector  $\mathbf{x}$ , with weights provided by the coordinates in the Fourier basis. To be more precise, we have that the change of coordinates performed by the Fourier matrix can be written as

$$\mathbf{x} = y_0\phi_0 + y_1\phi_1 + \cdots + y_{N-1}\phi_{N-1} = (\phi_0 \ \phi_1 \ \cdots \ \phi_{N-1})\mathbf{y} = F_N^{-1}\mathbf{y}, \quad (2.4)$$

where we have used the inverse of the defining relation  $\mathbf{y} = F_N\mathbf{x}$ , and that the  $\phi_n$  are the columns in  $F_N^{-1}$  (this follows from the fact that  $F_N^{-1}$  is the change of coordinates matrix from the Fourier basis to the standard basis, and the Fourier basis vectors are clearly the columns in this matrix). Equation (2.4) is also called the synthesis equation.

Let us find an expression for the matrix  $F_N$ . From Lemma 2.3 we know that the columns of  $F_N^{-1}$  are orthonormal. If the matrix was real, it would have been called orthogonal, and the inverse matrix could have been obtained by transposing.  $F_N^{-1}$  is complex, however, and it is easy to see that the conjugation present in the definition of the inner product (2.1), implies that the inverse of  $F_N$  can be obtained if we also conjugate, in addition to transpose, i.e.  $(F_N)^{-1} = (\overline{F_N})^T$ . We call  $(\overline{A})^T$  the *conjugate transpose* of  $A$ , and denote this by  $A^H$ . We thus have that  $(F_N)^{-1} = (F_N)^H$ . Matrices which satisfy  $A = A^H$  are called *unitary*. For complex matrices, this is the parallel to orthogonal matrices.

**Theorem 2.5.** *Fourier matrix is unitary.*

The Fourier matrix  $F_N$  is the unitary  $N \times N$ -matrix with entries given by

$$(F_N)_{nk} = \frac{1}{\sqrt{N}}e^{-2\pi ink/N},$$

for  $0 \leq n, k \leq N - 1$ .

Since the Fourier matrix is easily inverted, the DFT is also easily inverted. Note that, since  $(F_N)^T = F_N$ , we have that  $(F_N)^{-1} = \overline{F_N}$ . Let us make the following definition.

**Definition 2.6.** *IDFT.*

The matrix  $\overline{F_N}/\sqrt{N}$  is the inverse of the matrix  $\text{DFT} = \sqrt{N}F_N$ . We call this inverse matrix the *inverse discrete Fourier transform*, or IDFT.

We can thus also view the IDFT as a change of coordinates (this time from the Fourier basis to the standard basis), with a scaling of the coordinates by  $1/\sqrt{N}$  at the end. The IDFT is often called the *reverse DFT*. Similarly, the DFT is often called the *forward DFT*.

That  $\mathbf{y} = \text{DFT}\mathbf{x}$  and  $\mathbf{x} = \text{IDFT}\mathbf{y}$  can also be expressed in component form as

$$y_n = \sum_{k=0}^{N-1} x_k e^{-2\pi i n k / N} \quad x_k = \frac{1}{N} \sum_{n=0}^{N-1} y_n e^{2\pi i n k / N} \quad (2.5)$$

In applied fields such as signal processing, it is more common to state the DFT and IDFT in these component forms, rather than in the matrix forms  $\mathbf{y} = \text{DFT}\mathbf{x}$  and  $\mathbf{x} = \text{IDFT}\mathbf{y}$ .

Let us now see how these formulas work out in practice by considering some examples.

**Example 2.1: DFT of a cosine**

Let  $\mathbf{x}$  be the vector of length  $N$  defined by  $x_k = \cos(2\pi 5k/N)$ , and  $\mathbf{y}$  the vector of length  $N$  defined by  $y_k = \sin(2\pi 7k/N)$ . Let us see how we can compute  $F_N(2\mathbf{x} + 3\mathbf{y})$ . By the definition of the Fourier matrix as a change of coordinates,  $F_N(\phi_n) = \mathbf{e}_n$ . We therefore get

$$\begin{aligned} F_N(2\mathbf{x} + 3\mathbf{y}) &= F_N(2\cos(2\pi 5 \cdot /N) + 3\sin(2\pi 7 \cdot /N)) \\ &= F_N\left(2\frac{1}{2}(e^{2\pi i 5 \cdot /N} + e^{-2\pi i 5 \cdot /N}) + 3\frac{1}{2i}(e^{2\pi i 7 \cdot /N} - e^{-2\pi i 7 \cdot /N})\right) \\ &= F_N(\sqrt{N}\phi_5 + \sqrt{N}\phi_{N-5} - \frac{3i}{2}\sqrt{N}(\phi_7 - \phi_{N-7})) \\ &= \sqrt{N}(F_N(\phi_5) + F_N(\phi_{N-5}) - \frac{3i}{2}F_N\phi_7 + \frac{3i}{2}F_N\phi_{N-7}) \\ &= \sqrt{N}\mathbf{e}_5 + \sqrt{N}\mathbf{e}_{N-5} - \frac{3i}{2}\sqrt{N}\mathbf{e}_7 + \frac{3i}{2}\sqrt{N}\mathbf{e}_{N-7}. \end{aligned}$$

**Example 2.2: DFT on a square wave**

Let us attempt to apply the DFT to a signal  $\mathbf{x}$  which is 1 on indices close to 0, and 0 elsewhere. Assume that

$$x_{-L} = \dots = x_{-1} = x_0 = x_1 = \dots = x_L = 1,$$

while all other values are 0. This is similar to a square wave, with some modifications: First of all we assume symmetry around 0, while the square wave

of Example 1.4 assumes antisymmetry around 0. Secondly the values of the square wave are now 0 and 1, contrary to  $-1$  and  $1$  before. Finally, we have a different proportion of where the two values are assumed. Nevertheless, we will also refer to the current digital sound as a square wave.

Since indices with the DFT are between 0 and  $N-1$ , and since  $\mathbf{x}$  is assumed to have period  $N$ , the indices  $[-L, L]$  where our signal is 1 translates to the indices  $[0, L]$  and  $[N-L, N-1]$  (i.e., it is 1 on the first and last parts of the vector). Elsewhere our signal is zero. Since  $\sum_{k=N-L}^{N-1} e^{-2\pi ink/N} = \sum_{k=-L}^{-1} e^{-2\pi ink/N}$  (since  $e^{-2\pi ink/N}$  is periodic with period  $N$ ), the DFT of  $\mathbf{x}$  is

$$\begin{aligned} y_n &= \sum_{k=0}^L e^{-2\pi ink/N} + \sum_{k=N-L}^{N-1} e^{-2\pi ink/N} = \sum_{k=0}^L e^{-2\pi ink/N} + \sum_{k=-L}^{-1} e^{-2\pi ink/N} \\ &= \sum_{k=-L}^L e^{-2\pi ink/N} = e^{2\pi inL/N} \frac{1 - e^{-2\pi in(2L+1)/N}}{1 - e^{-2\pi in/N}} \\ &= e^{2\pi inL/N} e^{-\pi in(2L+1)/N} e^{\pi in/N} \frac{e^{\pi in(2L+1)/N} - e^{-\pi in(2L+1)/N}}{e^{\pi in/N} - e^{-\pi in/N}} \\ &= \frac{\sin(\pi n(2L+1)/N)}{\sin(\pi n/N)}. \end{aligned}$$

This computation does in fact also give us the IDFT of the same vector, since the IDFT just requires a change of sign in all the exponents, in addition to the  $1/N$  normalizing factor. From this example we see that, in order to represent  $\mathbf{x}$  in terms of frequency components, all components are actually needed. The situation would have been easier if only a few frequencies were needed.

### Example 2.3: Computing the DFT by hand

In most cases it is difficult to compute a DFT by hand, due to the entries  $e^{-2\pi ink/N}$  in the matrices, which typically can not be represented exactly. The DFT is therefore usually calculated on a computer only. However, in the case  $N = 4$  the calculations are quite simple. In this case the Fourier matrix takes the form

$$\text{DFT}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}.$$

We now can compute the DFT of a vector like  $(1, 2, 3, 4)^T$  simply as

$$\text{DFT}_4 \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 1+2+3+4 \\ 1-2i-3+4i \\ 1-2+3-4 \\ 1+2i-3-4i \end{pmatrix} = \begin{pmatrix} 10 \\ -2+2i \\ -2 \\ -2-2i \end{pmatrix}.$$



In general, computing the DFT implies using floating point multiplication. For  $N = 4$ , however, we see that there is no need for floating point multiplication at all, since  $\text{DFT}_4$  has unit entries which are either real or purely imaginary.

### Example 2.4: Direct implementation of the DFT

The DFT can be implemented very simply and directly by the code

```
def DFTImpl(x):
    y = zeros_like(x).astype(complex)
    N = len(x)
    for n in xrange(N):
        D = exp(-2*pi*n*1j*arange(float(N))/N)
        y[n] = dot(D, x)
    return y
```

In Exercise 2.13 we will extend this to a general implementation we will use later. Note that we do not allocate the entire matrix  $F_N$  in this code, as this quickly leads to out of memory situations, even for  $N$  of moderate size. Instead we construct one row of  $F_N$  at a time, and use this to compute one entry in the output. The method `dot` can be used here, since each entry in matrix multiplication can be viewed as an inner product. It is likely that the `dot` function is more efficient than using a `for`-loop, since Python may have an optimized way for computing this. Note that `dot` in Python does not conjugate any of the components, contrary to what we do in our definition of a complex inner product. This can be rewritten to a direct implementation of the IDFT also. We will look at this in the exercises, where we also make the method more general, so that the DFT can be applied to a series of vectors at a time (it can then be applied to all the channels in a sound in one call). Multiplying a full  $N \times N$  matrix by a vector requires roughly  $N^2$  arithmetic operations. The DFT algorithm above will therefore take a long time when  $N$  becomes moderately large. It turns out that a much more efficient algorithm exists for computing the DFT, which we will study at the end of this chapter. Python also has a built-in implementation of the DFT which uses such an efficient algorithm.

#### 2.1.1 Properties of the DFT

The DFT has properties which are very similar to those of Fourier series, as they were listed in Theorem 1.17. The following theorem sums this up:

**Theorem 2.7.** *Properties of the DFT.*

Let  $\mathbf{x}$  be a real vector of length  $N$ . The DFT has the following properties:

1.  $(\widehat{\mathbf{x}})_{N-n} = \overline{(\widehat{\mathbf{x}})_n}$  for  $0 \leq n \leq N - 1$ .
2. If  $x_k = x_{N-k}$  for all  $n$  (so  $\mathbf{x}$  is symmetric), then  $\widehat{\mathbf{x}}$  is a real vector.
3. If  $x_k = -x_{N-k}$  for all  $k$  (so  $\mathbf{x}$  is antisymmetric), then  $\widehat{\mathbf{x}}$  is a purely imaginary vector.

4. If  $d$  is an integer and  $\mathbf{z}$  is the vector with components  $z_k = x_{k-d}$  (the vector  $\mathbf{x}$  with its elements delayed by  $d$ ), then  $(\hat{\mathbf{z}})_n = e^{-2\pi idn/N} (\hat{\mathbf{x}})_n$ .
5. If  $d$  is an integer and  $\mathbf{z}$  is the vector with components  $z_k = e^{2\pi idk/N} x_k$ , then  $(\hat{\mathbf{z}})_n = (\hat{\mathbf{x}})_{n-d}$ .

*Proof.* The methods used in the proof are very similar to those used in the proof of Theorem 1.17. From the definition of the DFT we have

$$(\hat{\mathbf{x}})_{N-n} = \sum_{k=0}^{N-1} e^{-2\pi ik(N-n)/N} x_k = \sum_{k=0}^{N-1} e^{2\pi ikn/N} x_k = \overline{\sum_{k=0}^{N-1} e^{-2\pi ikn/N} x_k} = \overline{(\hat{\mathbf{x}})_n}$$

which proves property 1.

To prove property 2, we write

$$\begin{aligned} (\hat{\mathbf{z}})_n &= \sum_{k=0}^{N-1} z_k e^{-2\pi ikn/N} = \sum_{k=0}^{N-1} x_{N-k} e^{-2\pi ikn/N} = \sum_{u=1}^N x_u e^{-2\pi i(N-u)n/N} \\ &= \sum_{u=0}^{N-1} x_u e^{2\pi iun/N} = \overline{\sum_{u=0}^{N-1} x_u e^{-2\pi iun/N}} = \overline{(\hat{\mathbf{x}})_n}. \end{aligned}$$

If  $\mathbf{x}$  is symmetric it follows that  $\mathbf{z} = \mathbf{x}$ , so that  $(\hat{\mathbf{x}})_n = \overline{(\hat{\mathbf{x}})_n}$ . Therefore  $\mathbf{x}$  must be real. The case of antisymmetry in property 3 follows similarly.

To prove property 4 we observe that

$$\begin{aligned} (\hat{\mathbf{z}})_n &= \sum_{k=0}^{N-1} x_{k-d} e^{-2\pi ikn/N} = \sum_{k=0}^{N-1} x_k e^{-2\pi i(k+d)n/N} \\ &= e^{-2\pi idn/N} \sum_{k=0}^{N-1} x_k e^{-2\pi ikn/N} = e^{-2\pi idn/N} (\hat{\mathbf{x}})_n. \end{aligned}$$

For the proof of property 5 we note that the DFT of  $\mathbf{z}$  is

$$(\hat{\mathbf{z}})_n = \sum_{k=0}^{N-1} e^{2\pi idk/N} x_n e^{-2\pi ikn/N} = \sum_{k=0}^{N-1} x_n e^{-2\pi i(n-d)k/N} = (\hat{\mathbf{x}})_{n-d}.$$

This completes the proof.  $\square$

These properties have similar interpretations as the ones listed in Theorem 1.17 for Fourier series. Property 1 says that we need to store only about one half of the DFT coefficients, since the remaining coefficients can be obtained by conjugation. In particular, when  $N$  is even, we only need to store  $y_0, y_1, \dots, y_{N/2}$ . This also means that, if we plot the (absolute value) of the DFT of a real vector, we will see a symmetry around the index  $n = N/2$ . The theorem generalizes the properties from Theorem 1.17, except for the last property where the signal had a point of symmetry. We will delay the generalization of this property to later.

**Example 2.5: Computing the DFT when multiplying with a complex exponential**

To see how we can use the fourth property of Theorem 2.7, consider a vector  $\mathbf{x} = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$  with length  $N = 8$ , and assume that  $\mathbf{x}$  is so that  $F_8(\mathbf{x}) = (1, 2, 3, 4, 5, 6, 7, 8)$ . Consider the vector  $\mathbf{z}$  with components  $z_k = e^{2\pi i 2k/8} x_k$ . Let us compute  $F_8(\mathbf{z})$ . Since multiplication of  $\mathbf{x}$  with  $e^{2\pi i kd/N}$  delays the output  $\mathbf{y} = F_N(\mathbf{x})$  with  $d$  elements, setting  $d = 2$ , the  $F_8(\mathbf{z})$  can be obtained by delaying  $F_8(\mathbf{x})$  by two elements, so that  $F_8(\mathbf{z}) = (7, 8, 1, 2, 3, 4, 5, 6)$ . It is straightforward to compute this directly also:

$$\begin{aligned} (F_N \mathbf{z})_n &= \sum_{k=0}^{N-1} z_k e^{-2\pi i kn/N} = \sum_{k=0}^{N-1} e^{2\pi i 2k/N} x_k e^{-2\pi i kn/N} \\ &= \sum_{k=0}^{N-1} x_k e^{-2\pi i k(n-2)/N} = (F_N(\mathbf{x}))_{n-2}. \end{aligned}$$

**Exercise 2.6: Computing the DFT by hand**

Compute  $F_4 \mathbf{x}$  when  $\mathbf{x} = (2, 3, 4, 5)$ .

**Exercise 2.7: Exact form of low-order DFT matrix**

As in Example 2.3, state the exact cartesian form of the Fourier matrix for the cases  $N = 6$ ,  $N = 8$ , and  $N = 12$ .

**Exercise 2.8: DFT of a delayed vector**

We have a real vector  $\mathbf{x}$  with length  $N$ , and define the vector  $\mathbf{z}$  by delaying all elements in  $\mathbf{x}$  with 5 cyclically, i.e.  $z_5 = x_0$ ,  $z_6 = x_1, \dots, z_{N-1} = x_{N-6}$ , and  $z_0 = x_{N-5}, \dots, z_4 = x_{N-1}$ . For a given  $n$ , if  $|(F_N \mathbf{x})_n| = 2$ , what is then  $|(F_N \mathbf{z})_n|$ ? Justify the answer.

**Exercise 2.9: Using symmetry property**

Given a real vector  $\mathbf{x}$  of length 8 where  $(F_8(\mathbf{x}))_2 = 2 - i$ , what is  $(F_8(\mathbf{x}))_6$ ?

**Exercise 2.10: DFT of  $\cos^2(2\pi k/N)$** 

Let  $\mathbf{x}$  be the vector of length  $N$  where  $x_k = \cos^2(2\pi k/N)$ . What is then  $F_N \mathbf{x}$ ?

**Exercise 2.11: DFT of  $c^k \mathbf{x}$** 

Let  $\mathbf{x}$  be the vector with entries  $x_k = c^k$ . Show that the DFT of  $\mathbf{x}$  is given by the vector with components

$$y_n = \frac{1 - c^N}{1 - ce^{-2\pi in/N}}$$

for  $n = 0, \dots, N - 1$ .

### Exercise 2.12: Rewrite a complex DFT as real DFT's

If  $\mathbf{x}$  is complex, Write the DFT in terms of the DFT on real sequences.

**Hint.** Split into real and imaginary parts, and use linearity of the DFT.

### Exercise 2.13: DFT implementation

Extend the code for the function `DFTImpl` in Example 2.4 so that

- The function also takes a second parameter called `forward`. If this is true the DFT is applied. If it is false, the IDFT is applied. If this parameter is not present, then the forward transform should be assumed.
- If the input  $\mathbf{x}$  is two-dimensional (i.e. a matrix), the DFT/IDFT should be applied to each column of  $\mathbf{x}$ . This ensures that, in the case of sound, the FFT is applied to each channel in the sound when the entire sound is used as input, as we are used to when applying different operations to sound.

Also, write documentation for the code.

### Exercise 2.14: Symmetry

Assume that  $N$  is even.

- a) Show that, if  $x_{k+N/2} = x_k$  for all  $0 \leq k < N/2$ , then  $y_n = 0$  when  $n$  is odd.
- b) Show that, if  $x_{k+N/2} = -x_k$  for all  $0 \leq k < N/2$ , then  $y_n = 0$  when  $n$  is even.
- c) Show also the converse statements in a) and b).
- d) Also show the following:
  - $x_n = 0$  for all odd  $n$  if and only if  $y_{k+N/2} = y_k$  for all  $0 \leq k < N/2$ .
  - $x_n = 0$  for all even  $n$  if and only if  $y_{k+N/2} = -y_k$  for all  $0 \leq k < N/2$ .

**Exercise 2.15: DFT on complex and real data**

Let  $\mathbf{x}_1, \mathbf{x}_2$  be real vectors, and set  $\mathbf{x} = \mathbf{x}_1 + i\mathbf{x}_2$ . Use Theorem 2.7 to show that

$$\begin{aligned}(F_N(\mathbf{x}_1))_k &= \frac{1}{2} \left( (F_N(\mathbf{x}))_k + \overline{(F_N(\mathbf{x}))_{N-k}} \right) \\ (F_N(\mathbf{x}_2))_k &= \frac{1}{2i} \left( (F_N(\mathbf{x}))_k - \overline{(F_N(\mathbf{x}))_{N-k}} \right)\end{aligned}$$

This shows that we can compute two DFT's on real data from one DFT on complex data, and  $2N$  extra additions.

## 2.2 Connection between the DFT and Fourier series. Sampling and the sampling theorem

So far we have focused on the DFT as a tool to rewrite a vector in terms of the Fourier basis vectors. In practice, the given vector  $\mathbf{x}$  will often be sampled from some real data given by a function  $f(t)$ . We may then compare the frequency content of  $\mathbf{x}$  and  $f$ , and ask how they are related: What is the relationship between the Fourier coefficients of  $f$  and the DFT-coefficients of  $\mathbf{x}$ ?

In order to study this, assume for simplicity that  $f \in V_{M,T}$  for some  $M$ . This means that  $f$  equals its Fourier approximation  $f_M$ ,

$$f(t) = f_M(t) = \sum_{n=-M}^M z_n e^{2\pi i n t / T}, \text{ where } z_n = \frac{1}{T} \int_0^T f(t) e^{-2\pi i n t / T} dt. \quad (2.6)$$

We here have changed our notation for the Fourier coefficients from  $y_n$  to  $z_n$ , in order not to confuse them with the DFT coefficients. We recall that in order to represent the frequency  $n/T$  fully, we need the corresponding exponentials with both positive and negative arguments, i.e., both  $e^{2\pi i n t / T}$  and  $e^{-2\pi i n t / T}$ .

**Fact 2.8.** *frequency vs. Fourier coefficients.*

Suppose  $f$  is given by its Fourier series (2.6). Then the total frequency content for the frequency  $n/T$  is given by the two coefficients  $z_n$  and  $z_{-n}$ .

We have the following connection between the Fourier coefficients of  $f$  and the DFT of the samples of  $f$ .

**Proposition 2.9.** *Relation between Fourier coefficients and DFT coefficients.*

Let  $N > 2M$ ,  $f \in V_{M,T}$ , and let  $\mathbf{x} = \{f(kT/N)\}_{k=0}^{N-1}$  be  $N$  uniform samples from  $f$  over  $[0, T]$ . The Fourier coefficients  $z_n$  of  $f$  can be computed from

$$(z_0, z_1, \dots, z_M, \underbrace{0, \dots, 0}_{N-(2M+1)}, z_{-M}, z_{-M+1}, \dots, z_{-1}) = \frac{1}{N} \text{DFT}_N \mathbf{x}. \quad (2.7)$$

In particular, the total contribution in  $f$  from frequency  $n/T$ , for  $0 \leq n \leq M$ , is given by  $y_n$  and  $y_{N-n}$ , where  $\mathbf{y}$  is the DFT of  $\mathbf{x}$ .

*Proof.* Let  $\mathbf{x}$  and  $\mathbf{y}$  be as defined, so that

$$x_k = \frac{1}{N} \sum_{n=0}^{N-1} y_n e^{2\pi i n k / N}. \quad (2.8)$$

Inserting the sample points  $t = kT/N$  into the Fourier series, we must have that

$$\begin{aligned} x_k = f(kT/N) &= \sum_{n=-M}^M z_n e^{2\pi i n k / N} = \sum_{n=-M}^{-1} z_n e^{2\pi i n k / N} + \sum_{n=0}^M z_n e^{2\pi i n k / N} \\ &= \sum_{n=N-M}^{N-1} z_{n-N} e^{2\pi i (n-N)k / N} + \sum_{n=0}^M z_n e^{2\pi i n k / N} \\ &= \sum_{n=0}^M z_n e^{2\pi i n k / N} + \sum_{n=N-M}^{N-1} z_{n-N} e^{2\pi i n k / N}. \end{aligned}$$

This states that  $\mathbf{x} = \text{NIDFT}_N(z_0, z_1, \dots, z_M, \underbrace{0, \dots, 0}_{N-(2M+1)}, z_{-M}, z_{-M+1}, \dots, z_{-1})$ .

Equation (2.7) follows by applying the DFT to both sides. We also see that  $z_n = y_n/N$  and  $z_{-n} = y_{2M+1-n}/N = y_{N-n}/N$ , when  $\mathbf{y}$  is the DFT of  $\mathbf{x}$ . It now also follows immediately that the frequency content in  $f$  for the frequency  $n/T$  is given by  $y_n$  and  $y_{N-n}$ . This completes the proof.  $\square$

In Proposition 2.9 we take  $N$  samples over  $[0, T]$ , i.e. we sample at rate  $f_s = N/T$  samples per second. When  $|n| \leq M$ , a pure sound with frequency  $\nu = n/T$  is then seen to correspond to the DFT indices  $n$  and  $N - n$ . Since  $T = N/f_s$ ,  $\nu = n/T$  can also be written as  $\nu = n f_s / N$ . Moreover, the highest frequencies in Proposition 2.9 are those close to  $\nu = M/T$ , which correspond to DFT indices close to  $N - M$  and  $M$ , which are the nonzero frequencies closest to  $N/2$ . DFT index  $N/2$  corresponds to the frequency  $N/(2T) = f_s/2$ , which corresponds to the highest frequency we can reconstruct from samples for any  $M$ . Similarly, the lowest frequencies are those close to  $\nu = 0$ , which correspond to DFT indices close to 0 and  $N$ . Let us summarize this as follows.

**Observation 2.10.** *Connection between DFT index and frequency.*

Assume that  $\mathbf{x}$  are  $N$  samples of a sound taken at sampling rate  $f_s$  samples per second, and let  $\mathbf{y}$  be the DFT of  $\mathbf{x}$ . Then the DFT indices  $n$  and  $N - n$  give the frequency contribution at frequency  $\nu = n f_s / N$ . Moreover, the low frequencies in  $\mathbf{x}$  correspond to the  $y_n$  with  $n$  near 0 and  $N$ , while the high frequencies in  $\mathbf{x}$  correspond to the  $y_n$  with  $n$  near  $N/2$ .

The theorem says that any  $f \in V_{M,T}$  can be reconstructed from its samples (since we can write down its Fourier series), as long as  $N > 2M$ . That  $f \in V_{M,T}$

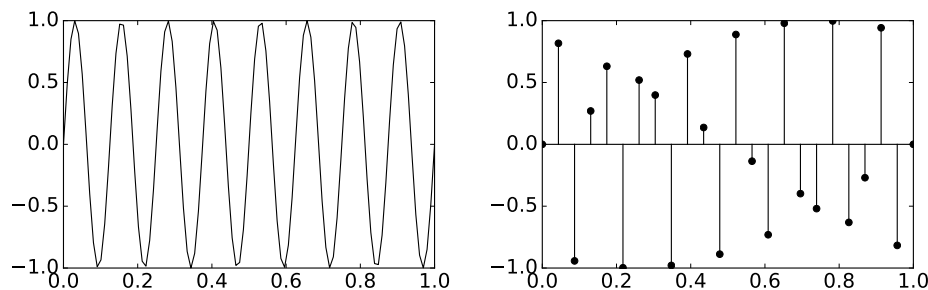


Figure 2.2: An example on how the samples are picked from an underlying continuous time function (left), and the samples on their own (right).

is important. From Figure 2.2 it is clear that information is lost in the right plot when we discard everything but the sample values from the left plot.

Here the function is  $f(t) = \sin(2\pi 8t) \in V_{8,1}$ , so that we need to choose  $N$  so that  $N > 2M = 16$  samples. Here  $N = 23$  samples were taken, so that reconstruction from the samples is possible. That the condition  $N < 2M$  is also necessary can easily be observed in Figure 2.3.

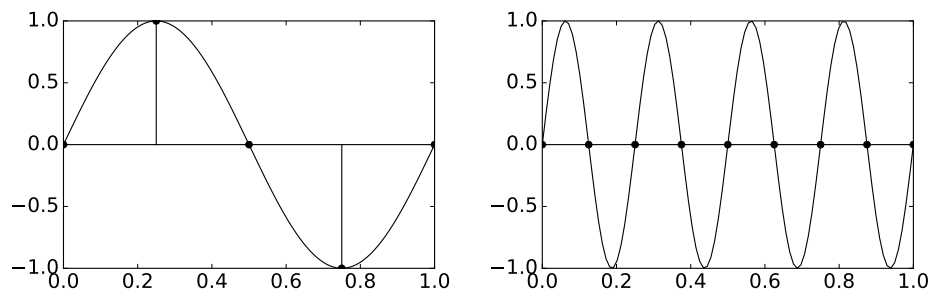


Figure 2.3: Sampling  $\sin(2\pi t)$  with two points (left), and sampling  $\sin(2\pi 4t)$  with eight points (right).

Right we have plotted  $\sin(2\pi 4t) \in V_{4,1}$ , with  $N = 8$  sample points taken uniformly from  $[0, 1]$ . Here  $M = 4$ , so that we require  $2M + 1 = 9$  sample points, according to Proposition 2.9. Clearly there is an infinite number of possible functions in  $V_{M,T}$  passing through the sample points (which are all zero): Any  $f(t) = c\sin(2\pi 4t)$  will do. Left we consider one period of  $\sin(2\pi t)$ . Since this is in  $V_{M,T} = V_{1,1}$ , reconstruction should be possible if we have  $N \geq 2M + 1 = 3$  samples. Four sample points, as seen left, is thus be enough to secure reconstruct.

The special case  $N = 2M + 1$  is interesting. No zeros are then inserted in the vector in Equation (2.7). Since the DFT is one-to-one, this means that there is a one-to-one correspondence between sample values and functions in  $V_{M,T}$  (i.e. Fourier series), i.e. we can always find a unique interpolant in  $V_{M,T}$  from  $N = 2M + 1$  samples. In Exercise 2.21 you will be asked to write code where you start with a given function  $f$ , Take  $N = 2M + 1$  samples, and plot the interpolant

from  $V_{M,T}$  against  $f$ . Increasing  $M$  should give an interpolant which is a better approximation to  $f$ , and if  $f$  itself resides in some  $V_{M,T}$  for some  $M$ , we should obtain equality when we choose  $M$  big enough. We have in elementary calculus courses seen how to determine a polynomial of degree  $N - 1$  that interpolates a set of  $N$  data points, and such polynomials are called interpolating polynomials. In mathematics many other classes than polynomials exist which are also useful for interpolation, and the Fourier basis is just one example.

Besides reconstructing a function from its samples, Proposition 2.9 also enables us to approximate functions in a simple way. To elaborate on this, recall that the Fourier series approximation  $f_M$  is a best approximation to  $f$  from  $V_{M,T}$ . We usually can't compute  $f_M$  exactly, however, since this requires us to compute the Fourier integrals. We could instead form the samples  $\mathbf{x}$  of  $f$ , and apply Proposition 2.9. If  $M$  is high,  $f_M$  is a good approximation to  $f$ , so that the samples of  $f_M$  are a good approximation to  $\mathbf{x}$ . By continuity of the DFT, it follows that  $\mathbf{y} = \text{DFT}_N \mathbf{x}$  is a good approximation to the DFT of the samples of  $f_M$ , so that

$$\tilde{f}(t) = \sum_{n=0}^{N-1} y_n e^{2\pi i n t / T} \quad (2.9)$$

is a good approximation to  $f_M$ , and therefore also to  $f$ . We have illustrated this in Figure 2.4.

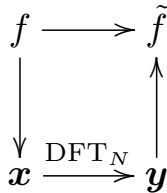


Figure 2.4: How we can interpolate  $f$  from  $V_{M,T}$  with help of the DFT. The left vertical arrow represents sampling. The right vertical arrow represents interpolation, i.e. computing Equation (2.9).

The new function  $\tilde{f}$  has the same values as  $f$  in the sample points. This is usually not the case for  $f_M$ , so that  $\tilde{f}$  and  $f_M$  are different approximations to  $f$ . Let us summarize as follows.

**Idea 2.11.**  $\tilde{f}$  as approximation to  $f$ .

The function  $\tilde{f}$  resulting from sampling, taking the DFT, and interpolation, as shown in Figure 2.4, also gives an approximation to  $f$ .  $\tilde{f}$  is a worse approximation in the mean square sense (since  $f_M$  is the best such), but it is much more useful since it avoids evaluation of the Fourier integrals, depends only on the samples, and is easily computed.

The condition  $N > 2M$  in Proposition 2.9 can also be written as  $N/T > 2M/T$ . The left side is now the sampling rate  $f_s$ , while the right side is the



double of the highest frequency in  $f$ . The result can therefore also be restated as follows

**Proposition 2.12.** *Reconstruction from samples.*

Any  $f \in V_{M,T}$  can be reconstructed uniquely from a uniform set of samples  $\{f(kT/N)\}_{k=0}^{N-1}$ , as long as  $f_s > 2|\nu|$ , where  $\nu$  denotes the highest frequency in  $f$ .

We also refer to  $f_s = 2|\nu|$  as the critical sampling rate, since it is the minimum sampling rate we need in order to reconstruct  $f$  from its samples. If  $f_s$  is substantially larger than  $2|\nu|$  we say that  $f$  is oversampled, since we have taken more samples than we really need. Similarly we say that  $f$  is undersampled if  $f_s$  is smaller than  $2|\nu|$ , since we have not taken enough samples in order to reconstruct  $f$ . Clearly Proposition 2.9 gives one formula for the reconstruction. In the literature another formula can be found, which we now will deduce. This alternative version of Theorem 2.9 is also called *the sampling theorem*. We start by substituting  $N = T/T_s$  (i.e.  $T = NT_s$ , with  $T_s$  being the sampling period) in the Fourier series for  $f$ :

$$f(kT_s) = \sum_{n=-M}^M z_n e^{2\pi i n k / N} \quad -M \leq k \leq M.$$

Equation (2.7) said that the Fourier coefficients could be found from the samples from

$$(z_0, z_1, \dots, z_M, \underbrace{0, \dots, 0}_{N-(2M+1)}, z_{-M}, z_{-M+1}, \dots, z_{-1}) = \frac{1}{N} \text{DFT}_N \mathbf{x}.$$

By delaying the  $n$  index with  $-M$ , this can also be written as

$$z_n = \frac{1}{N} \sum_{k=0}^{N-1} f(kT_s) e^{-2\pi i n k / N} = \frac{1}{N} \sum_{k=-M}^M f(kT_s) e^{-2\pi i n k / N}, \quad -M \leq n \leq M.$$

Inserting this in the reconstruction formula we get

$$\begin{aligned}
f(t) &= \frac{1}{N} \sum_{n=-M}^M \sum_{k=-M}^M f(kT_s) e^{-2\pi i n k/N} e^{2\pi i n t/T} \\
&= \sum_{k=-M}^M \frac{1}{N} \left( \sum_{n=-M}^M f(kT_s) e^{2\pi i n(t/T - k/N)} \right) \\
&= \sum_{k=-M}^M \frac{1}{N} e^{-2\pi i M(t/T - k/N)} \frac{1 - e^{2\pi i(2M+1)(t/T - k/N)}}{1 - e^{2\pi i(t/T - k/N)}} f(kT_s) \\
&= \sum_{k=-M}^M \frac{1}{N} \frac{\sin(\pi(t - kT_s)/T_s)}{\sin(\pi(t - kT_s)/T)} f(kT_s)
\end{aligned}$$

Let us summarize our findings as follows:

**Theorem 2.13.** *Sampling theorem and the ideal interpolation formula for periodic functions.*

Let  $f$  be a periodic function with period  $T$ , and assume that  $f$  has no frequencies higher than  $\nu$  Hz. Then  $f$  can be reconstructed exactly from its samples  $f(-MT_s), \dots, f(MT_s)$  (where  $T_s$  is the sampling period,  $N = \frac{T}{T_s}$  is the number of samples per period, and  $M = 2N + 1$ ) when the sampling rate  $f_s = \frac{1}{T_s}$  is bigger than  $2\nu$ . Moreover, the reconstruction can be performed through the formula

$$f(t) = \sum_{k=-M}^M f(kT_s) \frac{1}{N} \frac{\sin(\pi(t - kT_s)/T_s)}{\sin(\pi(t - kT_s)/T)}. \quad (2.10)$$

Formula (2.10) is also called the ideal interpolation formula for periodic functions. Such formulas, where one reconstructs a function based on a weighted sum of the sample values, are more generally called *interpolation formulas*. The function  $\frac{1}{N} \frac{\sin(\pi(t - kT_s)/T_s)}{\sin(\pi(t - kT_s)/T)}$  is also called an interpolation kernel. Note that  $f$  itself may not be equal to a finite Fourier series, and reconstruction is in general not possible then. The ideal interpolation formula can in such cases still be used, but the result we obtain may be different from  $f(t)$ .

In fact, the following more general result holds, which we will not prove. The result is also valid for functions which are not periodic, and is frequently stated in the literature:

**Theorem 2.14.** *Sampling theorem and the ideal interpolation formula, general version..*

Assume that  $f$  has no frequencies higher than  $\nu$  Hz. Then  $f$  can be reconstructed exactly from its samples  $\dots, f(-2T_s), f(-T_s), f(0), f(T_s), f(2T_s), \dots$  when the sampling rate is bigger than  $2\nu$ . Moreover, the reconstruction can be performed through the formula

$$f(t) = \sum_{k=-\infty}^{\infty} f(kT_s) \frac{\sin(\pi(t - kT_s)/T_s)}{\pi(t - kT_s)/T_s}. \quad (2.11)$$

When  $f$  is periodic, it is possible to deduce this partly from the interpolation formula for periodic functions. An ingredient in this is that  $x \approx \sin x$  for small  $x$ , so that there certainly is a connection between the terms in the two sums. When  $f$  is not periodic we require more tools from Fourier analysis, however.

The DFT coefficients represent the contribution in a sound at given frequencies. Due to this the DFT is extremely useful for performing operations on sound, and also for compression. For instance we can listen to either the lower or higher frequencies after performing a simple adjustment of the DFT coefficients. Observation 2.10 says that the  $2L + 1$  lowest frequencies correspond to the DFT-indices  $[0, L] \cup [N - L, N - 1]$ , while the  $2L + 1$  highest frequencies correspond to DFT-indices  $[N/2 - L, N/2 + L]$  (assuming that  $N$  is even). If we perform a DFT, eliminate these low or high frequencies, and perform an inverse DFT, we recover the sound signal where these frequencies have been eliminated. The function `forw_comp_rev_DFT()` in the module `forw_comp_rev` can perform these tasks for our audio sample file, as well as some other useful tasks that can be useful for compression. This function accepts named parameters `L` and `lower`, where the lowest frequencies are kept if `lower==1`, and the highest frequencies are kept if `lower==0`.

### Example 2.16: Using the DFT to adjust frequencies in sound

Let us test the function `forw_comp_rev_DFT` to listen to the lower frequencies in the audio sample file. For  $L = 13000$ , the resulting sound can be found in the file `castanetslowerfreq7.wav`. For  $L = 5000$ , the resulting sound can be found in the file `castanetslowerfreq3.wav`. With  $L = 13000$  you can hear the disturbance in the sound, but we have not lost that much even if about 90% of the DFT coefficients are dropped. The quality is much poorer when  $L = 5000$  (here we keep less than 5% of the DFT coefficients). However we can still recognize the song, and this suggests that most of the frequency information is contained in the lower frequencies.

Let us then listen to higher frequencies instead. For  $L = 140000$ , the resulting sound can be found in the file `this`. For  $L = 100000$  resulting sound can be found in the file `castanetshigherfreq3.wav`. Both sounds are quite unrecognizable.

We find that we need very high values of  $L$  to hear anything, suggesting again that most information is contained in the lowest frequencies.

#### 2.2.1 Change in frequency representation when windowing a signal

Note that there may be a problem in the previous example: when we restrict to the values in a given block, we actually look at a different signal. The new signal

repeats the values in the block in periods, while the old signal consists of one much bigger block. What are the differences in the frequency representations of the two signals?

Assume that the entire sound has length  $M$ . The frequency representation of this is computed as an  $M$ -point DFT (the signal is actually repeated with period  $M$ ), and we write the sound samples as a sum of frequencies:  $x_k = \frac{1}{M} \sum_{n=0}^{M-1} y_n e^{2\pi i k n / M}$ . Let us consider the effect of restricting to a block for each of the contributing pure tones  $e^{2\pi i k n_0 / M}$ ,  $0 \leq n_0 \leq M - 1$ . When we restrict this to a block of size  $N$ , we get the signal  $\{e^{2\pi i k n_0 / M}\}_{k=0}^{N-1}$ . Depending on  $n_0$ , this may not be a Fourier basis vector! Its  $N$ -point DFT gives us its frequency representation, and the absolute value of this is

$$\begin{aligned} |y_n| &= \left| \sum_{k=0}^{N-1} e^{2\pi i k n_0 / M} e^{-2\pi i k n / N} \right| = \left| \sum_{k=0}^{N-1} e^{2\pi i k (n_0 / M - n / N)} \right| \\ &= \left| \frac{1 - e^{2\pi i N (n_0 / M - n / N)}}{1 - e^{2\pi i (n_0 / M - n / N)}} \right| = \left| \frac{\sin(\pi N (n_0 / M - n / N))}{\sin(\pi (n_0 / M - n / N))} \right|. \end{aligned} \quad (2.12)$$

If  $n_0 = kM/N$ , this gives  $y_k = N$ , and  $y_n = 0$  when  $n \neq k$ . Thus, splitting the signal into blocks gives another pure tone when  $n_0$  is a multiple of  $M/N$ . When  $n_0$  is different from this the situation is different. Let us set  $M = 1000$ ,  $n_0 = 1$ , and experiment with different values of  $N$ . Figure 2.5 shows the  $y_n$  values for different values of  $N$ . We see that the frequency representation is now very different, and that many frequencies contribute.

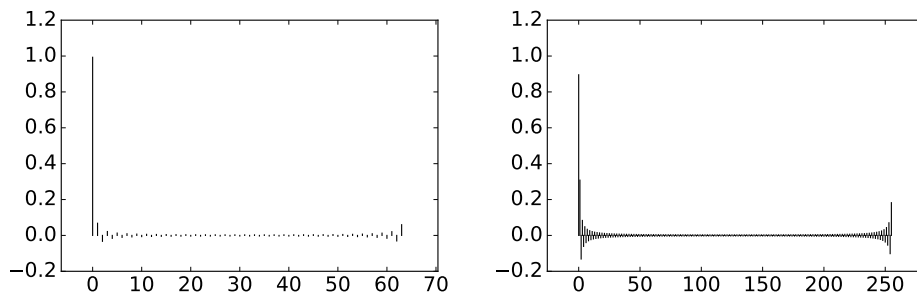


Figure 2.5: The frequency representation obtained when restricting to a block of size  $N$  of the signal, for  $N = 64$  (left), and  $N = 256$  (right)

The explanation is that the pure tone is not a pure tone when  $N = 64$  and  $N = 256$ , since at this scale such frequencies are too high to be represented exactly. The closest pure tone in frequency is  $n = 0$ , and we see that this has the biggest contribution, but other frequencies also contribute. The other frequencies contribute much more when  $N = 256$ , as can be seen from the peak in the closest frequency  $n = 0$ . In conclusion, when we split into blocks, the frequency representation may change in an undesirable way. This is a common

problem in signal processing theory, that one in practice needs to restrict to smaller segments of samples, but that this restriction may have undesired effects.

Another problem when we restrict to a shorter periodic signal is that we may obtain discontinuities at the boundaries between the new periods, even if there were no discontinuities in the original signal. And, as we know from the square wave, discontinuities introduce undesired frequencies. We have already mentioned that symmetric extensions may be used to remedy this.

The MP3 standard also applies a DFT to the sound data. In its simplest form it applies a 512 point DFT. There are some differences to how this is done when compared to Example 2.16, however. In our example we split the sound into disjoint blocks, and applied a DFT to each of them. The MP3 standard actually splits the sound into blocks which overlap, as this creates a more continuous frequency representation. Another difference is that the MP3 standard applies a *window* to the sound samples, and the effect of this is that the new signal has a frequency representation which is closer to the original one, when compared to the signal obtained by using the block values unchanged as above. We will go into details on this in the next chapter.

### Example 2.17: Compression by zeroing out small DFT coefficients

We can achieve compression of a sound by setting small DFT coefficients which to zero. The idea is that frequencies with small values at the corresponding frequency indices contribute little to our perception of the sound, so that they can be discarded. As a result we obtain a sound with less frequency components, which is thus more suitable for compression. To test this in practice, we first need to set a threshold, which decides which frequencies to keep. This can then be sent to the function `forw_comp_rev_DFT` by means of the named parameter `threshold`. The function will now also write to the display the percentage of the DFT coefficients which were zeroed out. If you run this function with `threshold` equal to 20, resulting sound can be found in the file `castanetsthreshold002.wav`, and the function says that about 68% of the DFT coefficients were set to zero. You can clearly hear the disturbance in the sound, but we have not lost that much. If we instead try `threshold` equal to 70, the resulting sound can be found in the file `castanetsthreshold01.wav`, and the function says that about 94% of the DFT coefficients were set to zero. The quality is much poorer now, even if we still can recognize the song. This suggests that most of the frequency information is contained in frequencies with the highest values.

In Figure 2.6 we have illustrated this principle for compression for 512 sound samples from a song. The samples of the sound and (the absolute value of) its DFT are shown at the top. At the bottom all values of the DFT with absolute value smaller than 0.02 are set to zero (52 values then remain), and the sound is reconstructed with the IDFT, and then shown in. The start and end signals look similar, even though the last signal can be represented with less than 10 % of the values from the first.

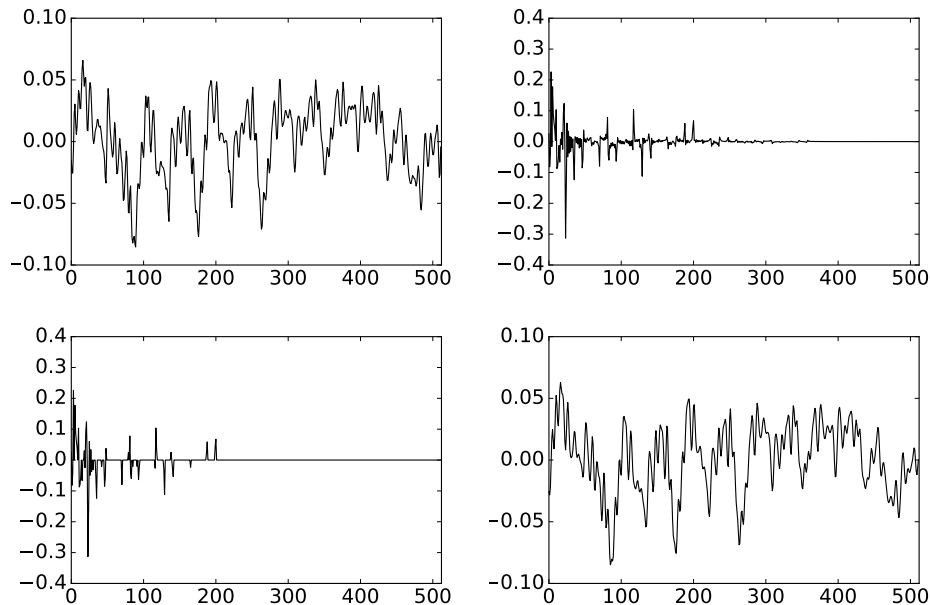


Figure 2.6: Experimenting with the DFT on a small part of a song.

Note that using a neglect threshold in this way is too simple in practice: The neglect threshold in general should depend on the frequency, since the human auditory system is more sensitive to certain frequencies.

### Example 2.18: Compression by quantizing DFT coefficients

The previous example is a rather simple procedure to obtain compression. The disadvantage is that it only affects frequencies with low contribution. A more neutral way to obtain compression is to let each DFT index occupy a certain number of bits. This is also called *quantization*, and provides us with compression if the number of bits is less than what actually is used to represent the sound. This is closer to what modern audio standards do. `forw_comp_rev_DFT` accepts a name parameter `n`. The effect of this is that a DFT coefficient with bit representation

$$\dots d_2 d_1 d_0 . d_{-1} d_{-2} d_{-3} \dots$$

is truncated so that the bits  $d_{n-1}$ ,  $d_{n-2}$ ,  $d_{n-3}$  are discarded. In other words, high values of  $n$  mean more rounding. If you run `forw_comp_rev_DFT` with `n` equal to 3, the resulting sound can be found in the file `castantesquantizedn3.wav`, with `n = 5` the resulting sound can be found in the file `castantesquantizedn5.wav`, and with `n = 7` resulting sound can be found in the file `castantesquantizedn7.wav`. You can hear that the sound degrades further when  $n$  is increased.

In practice this quantization procedure is also too simple, since the human auditory system is more sensitive to certain frequency information, and should thus allocate a higher number of bits for such frequencies. Modern audio standards take this into account, but we will not go into details on this.

### Exercise 2.19: Comment code

Explain what the code below does, line by line:

```
x = x[0:2**17]
y = fft.fft(x, axis=0)
y[(2**17/4):(3*2**17/4)] = 0
newx = abs(fft.ifft(y))
newx /= abs(newx).max()
play(newx, fs)
```

Comment in particular why we adjust the sound samples by dividing with the maximum value of the sound samples. What changes in the sound do you expect to hear?

### Exercise 2.20: Which frequency is changed?

In the code from the previous exercise it turns out that  $f_s = 44100\text{Hz}$ , and that the number of sound samples is  $N = 292570$ . Which frequencies in the sound file will be changed on the line where we zero out some of the DFT coefficients?

### Exercise 2.21: Implement interpolant

Implement code where you do the following:

- at the top you define the function  $f(x) = \cos^6(x)$ , and  $M = 3$ ,
- compute the unique interpolant from  $V_{M,T}$  (i.e. by taking  $N = 2M + 1$  samples over one period), as guaranteed by Proposition 2.9,
- plot the interpolant against  $f$  over one period.

Finally run the code also for  $M = 4$ ,  $M = 5$ , and  $M = 6$ . Explain why the plots coincide for  $M = 6$ , but not for  $M < 6$ . Does increasing  $M$  above  $M = 6$  have any effect on the plots?

## 2.3 The Fast Fourier Transform (FFT)

The main application of the DFT is as a tool to compute frequency information in large datasets. Since this is so useful in many areas, it is of vital importance that the DFT can be computed with efficient algorithms. The straightforward implementation of the DFT with matrix multiplication we looked at is not efficient for large data sets. However, it turns out that the DFT matrix may be

factored in a way that leads to much more efficient algorithms, and this is the topic of the present section. We will discuss the most widely used implementation of the DFT, usually referred to as the Fast Fourier Transform (FFT). The FFT has been stated as one of the ten most important inventions of the 20<sup>th</sup> century, and its invention made the DFT computationally feasible in many fields. The FFT is for instance used much in real time processing, such as processing and compression of sound, images, and video. The MP3 standard uses the FFT to find frequency components in sound, and matches this information with a psychoacoustic model, in order to find the best way to compress the data.

FFT-based functionality is collected in a module called `fft`.

Let us start with the most basic FFT algorithm, which applies for a general complex input vector  $\mathbf{x}$ , with length  $N$  being an even number.

**Theorem 2.15.** *FFT algorithm when  $N$  is even.*

Let  $\mathbf{y} = \text{DFT}_N \mathbf{x}$  be the  $N$ -point DFT of  $\mathbf{x}$ , with  $N$  an even number, and let  $D_{N/2}$  be the  $(N/2) \times (N/2)$ -diagonal matrix with entries  $(D_{N/2})_{n,n} = e^{-2\pi i n/N}$  for  $0 \leq n < N/2$ . Then we have that

$$(y_0, y_1, \dots, y_{N/2-1}) = \text{DFT}_{N/2} \mathbf{x}^{(e)} + D_{N/2} \text{DFT}_{N/2} \mathbf{x}^{(o)} \quad (2.13)$$

$$(y_{N/2}, y_{N/2+1}, \dots, y_{N-1}) = \text{DFT}_{N/2} \mathbf{x}^{(e)} - D_{N/2} \text{DFT}_{N/2} \mathbf{x}^{(o)} \quad (2.14)$$

where  $\mathbf{x}^{(e)}, \mathbf{x}^{(o)} \in \mathbb{R}^{N/2}$  consist of the even- and odd-indexed entries of  $\mathbf{x}$ , respectively, i.e.

$$\mathbf{x}^{(e)} = (x_0, x_2, \dots, x_{N-2}) \quad \mathbf{x}^{(o)} = (x_1, x_3, \dots, x_{N-1}).$$

Put differently, the formulas (2.13)-(2.14) reduce the computation of an  $N$ -point DFT to two  $N/2$ -point DFT's. It turns out that this is the basic fact which speeds up computations considerably. It is important to note that we first should compute that the same term  $D_{N/2} \text{DFT}_{N/2} \mathbf{x}^{(o)}$  appears in both formulas above. It is thus important that this is computed only once, and then inserted in both equations. Let us first check that these formulas are correct.

*Proof.* Suppose first that  $0 \leq n \leq N/2 - 1$ . We start by splitting the sum in the expression for the DFT into even and odd indices,

$$\begin{aligned} y_n &= \sum_{k=0}^{N-1} x_k e^{-2\pi i n k/N} = \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n 2k/N} + \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n (2k+1)/N} \\ &= \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n k/(N/2)} + e^{-2\pi i n/N} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n k/(N/2)} \\ &= \left( \text{DFT}_{N/2} \mathbf{x}^{(e)} \right)_n + e^{-2\pi i n/N} \left( \text{DFT}_{N/2} \mathbf{x}^{(o)} \right)_n, \end{aligned}$$



where we have substituted  $\mathbf{x}^{(e)}$  and  $\mathbf{x}^{(o)}$  as in the text of the theorem, and recognized the  $N/2$ -point DFT in two places. Assembling this for  $0 \leq n < N/2$  we obtain Equation (2.13). For the second half of the DFT coefficients, i.e.  $\{y_{N/2+n}\}_{0 \leq n \leq N/2-1}$ , we similarly have

$$\begin{aligned} y_{N/2+n} &= \sum_{k=0}^{N-1} x_k e^{-2\pi i(N/2+n)k/N} = \sum_{k=0}^{N-1} x_k e^{-\pi i k} e^{-2\pi i n k/N} \\ &= \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n 2k/N} - \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n(2k+1)/N} \\ &= \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n k/(N/2)} - e^{-2\pi i n/N} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n k/(N/2)} \\ &= \left( \text{DFT}_{N/2} \mathbf{x}^{(e)} \right)_n - e^{-2\pi i n/N} \left( \text{DFT}_{N/2} \mathbf{x}^{(o)} \right)_n. \end{aligned}$$

Equation (2.14) now follows similarly.  $\square$

Note that an algorithm for the IDFT can be deduced in exactly the same way. All we need to change is the sign in the exponents of the Fourier matrix. In addition we need to divide by  $1/N$  at the end. If we do this we get the following result, which we call the IFFT algorithm. Recall that we use the notation  $\bar{A}$  for the matrix where all the elements of  $A$  have been conjugated.

**Theorem 2.16.** *IFFT algorithm when  $N$  is even.*

Let  $N$  be an even number and let  $\tilde{\mathbf{x}} = \overline{\text{DFT}_N \mathbf{y}}$ . Then we have that

$$(\tilde{x}_0, \tilde{x}_1, \dots, \tilde{x}_{N/2-1}) = \overline{\text{DFT}_{N/2} \mathbf{y}^{(e)}} + \overline{D_{N/2} \text{DFT}_{N/2}} \mathbf{y}^{(o)} \quad (2.15)$$

$$(\tilde{x}_{N/2}, \tilde{x}_{N/2+1}, \dots, \tilde{x}_{N-1}) = \overline{\text{DFT}_{N/2} \mathbf{y}^{(e)}} - \overline{D_{N/2} \text{DFT}_{N/2}} \mathbf{y}^{(o)} \quad (2.16)$$

where  $\mathbf{y}^{(e)}, \mathbf{y}^{(o)} \in \mathbb{R}^{N/2}$  are the vectors

$$\mathbf{y}^{(e)} = (y_0, y_2, \dots, y_{N-2}) \quad \mathbf{y}^{(o)} = (y_1, y_3, \dots, y_{N-1}).$$

Moreover,  $\mathbf{x} = \text{IDFT}_N \mathbf{y}$  can be computed from  $\mathbf{x} = \tilde{\mathbf{x}}/N = \overline{\text{DFT}_N \mathbf{y}}/N$

It turns out that these theorems can be interpreted as matrix factorizations. For this we need to define the concept of a block matrix.

**Definition 2.17.** *Block matrix.*

Let  $m_0, \dots, m_{r-1}$  and  $n_0, \dots, n_{s-1}$  be integers, and let  $A^{(i,j)}$  be an  $m_i \times n_j$ -matrix for  $i = 0, \dots, r-1$  and  $j = 0, \dots, s-1$ . The notation

$$A = \begin{pmatrix} A^{(0,0)} & A^{(0,1)} & \dots & A^{(0,s-1)} \\ A^{(1,0)} & A^{(1,1)} & \dots & A^{(1,s-1)} \\ \vdots & \vdots & \ddots & \vdots \\ A^{(r-1,0)} & A^{(r-1,1)} & \dots & A^{(r-1,s-1)} \end{pmatrix}$$

denotes the  $(m_0 + m_1 + \dots + m_{r-1}) \times (n_0 + n_1 + \dots + n_{s-1})$ -matrix where the matrix entries occur as in the  $A^{(i,j)}$  matrices, in the way they are ordered. When  $A$  is written in this way it is referred to as a block matrix.

Clearly, using equations (2.13)-(2.14), the DFT matrix can be factorized using block matrix notation as

$$\begin{aligned} (y_0, y_1, \dots, y_{N/2-1}) &= (\text{DFT}_{N/2} \quad D_{N/2} \text{DFT}_{N/2}) \begin{pmatrix} \mathbf{x}^{(e)} \\ \mathbf{x}^{(o)} \end{pmatrix} \\ (y_{N/2}, y_{N/2+1}, \dots, y_{N-1}) &= (\text{DFT}_{N/2} \quad -D_{N/2} \text{DFT}_{N/2}) \begin{pmatrix} \mathbf{x}^{(e)} \\ \mathbf{x}^{(o)} \end{pmatrix}. \end{aligned}$$

Combining these, noting that

$$\begin{pmatrix} \text{DFT}_{N/2} & D_{N/2} \text{DFT}_{N/2} \\ \text{DFT}_{N/2} & -D_{N/2} \text{DFT}_{N/2} \end{pmatrix} = \begin{pmatrix} I & D_{N/2} \\ I & -D_{N/2} \end{pmatrix} \begin{pmatrix} \text{DFT}_{N/2} & \mathbf{0} \\ \mathbf{0} & \text{DFT}_{N/2} \end{pmatrix},$$

we obtain the following factorisations:

**Theorem 2.18.** *DFT and IDFT matrix factorizations.*

We have that

$$\begin{aligned} \text{DFT}_N \mathbf{x} &= \begin{pmatrix} I & D_{N/2} \\ I & -D_{N/2} \end{pmatrix} \begin{pmatrix} \text{DFT}_{N/2} & \mathbf{0} \\ \mathbf{0} & \text{DFT}_{N/2} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(e)} \\ \mathbf{x}^{(o)} \end{pmatrix} \\ \text{IDFT}_N \mathbf{y} &= \frac{1}{N} \overline{\begin{pmatrix} I & D_{N/2} \\ I & -D_{N/2} \end{pmatrix}} \overline{\begin{pmatrix} \text{DFT}_{N/2} & \mathbf{0} \\ \mathbf{0} & \text{DFT}_{N/2} \end{pmatrix}} \begin{pmatrix} \mathbf{y}^{(e)} \\ \mathbf{y}^{(o)} \end{pmatrix} \end{aligned} \quad (2.17)$$

We will shortly see why these factorizations reduce the number of arithmetic operations we need to do, but first let us consider how to implement them. First of all, note that we can apply the FFT factorizations again to  $F_{N/2}$  to obtain

$$\begin{aligned} \text{DFT}_N \mathbf{x} &= \begin{pmatrix} I & D_{N/2} \\ I & -D_{N/2} \end{pmatrix} \begin{pmatrix} I & D_{N/4} & \mathbf{0} & \mathbf{0} \\ I & -D_{N/4} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I & D_{N/4} \\ \mathbf{0} & \mathbf{0} & I & -D_{N/4} \end{pmatrix} \times \\ &\quad \begin{pmatrix} \text{DFT}_{N/4} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \text{DFT}_{N/4} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \text{DFT}_{N/4} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \text{DFT}_{N/4} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(ee)} \\ \mathbf{x}^{(eo)} \\ \mathbf{x}^{(oe)} \\ \mathbf{x}^{(oo)} \end{pmatrix} \end{aligned}$$

where the vectors  $\mathbf{x}^{(e)}$  and  $\mathbf{x}^{(o)}$  have been further split into even- and odd-indexed entries. Clearly, if this factorization is repeated, we obtain a factorization

$$\text{DFT}_N = \prod_{k=1}^{\log_2 N} \begin{pmatrix} I & D_{N/2^k} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ I & -D_{N/2^k} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I & D_{N/2^k} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I & -D_{N/2^k} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & I & D_{N/2^k} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & I & -D_{N/2^k} \end{pmatrix} P. \quad (2.18)$$

The factorization has been repeated until we have a final diagonal matrix with  $\text{DFT}_1$  on the diagonal, but clearly  $\text{DFT}_1 = 1$ , so we do not need any DFT-matrices in the final factor. Note that all matrices in this factorization are sparse. A factorization into a product of sparse matrices is the key to many efficient algorithms in linear algebra, such as the computation of eigenvalues and eigenvectors. When we later compute the number of arithmetic operations in this factorization, we will see that this is the case also here.

In Equation (2.18),  $P$  is a permutation matrix which secures that the even-indexed entries come first. Since the even-indexed entries have 0 as the last bit, this is the same as letting the last bit become the first bit. Since we here recursively place even-indexed entries first, it is not too difficult to see that  $P$  permutes the elements of  $\mathbf{x}$  by performing a *bit-reversal* of the indices, i.e.

$$P(\mathbf{e}_i) = \mathbf{e}_j \quad i = d_1 d_2 \dots d_n \quad j = d_n d_{n-1} \dots d_1,$$

where we have used the bit representations of  $i$  and  $j$ . Since  $P^2 = I$ , a bit-reversal can be computed very efficiently, and performed *in-place*, i.e. so that the result ends up in same vector  $\mathbf{x}$ , so that we do not need to allocate any memory in this operation. We will use an existing function called `bitreverse` to perform in-place bit-reversal. In Exercise 2.30 we will go through this implementation.

Matrix multiplication is usually not done in-place, i.e. when we compute  $\mathbf{y} = A\mathbf{x}$ , different memory is allocated for  $\mathbf{x}$  and  $\mathbf{y}$ . For certain simple matrices, however, matrix multiplication can also be done in-place, so that the output can be written into the same memory ( $\mathbf{x}$ ) used by the input. It turns out that the matrices in factorization (2.18) are of this kind, so that the entire FFT can be computed in-place. We will have more to say on this in the exercises.

In a practical algorithm, it is smart to perform the bit-reversal first, since the matrices in the factorization (2.18) are block diagonal, so that the different blocks in each matrix can be applied in parallel to  $P\mathbf{x}$  (the bit-reversed version of  $\mathbf{x}$ ). We can thus exploit the parallel processing capabilities of the computer. It turns out that this bit-reversal is useful for other similar factorizations of the DFT as well. We will also look at other such factorizations, and we will therefore split the computation of the DFT as follows: First a general function is applied,

which is responsible for the bit-reversal of the input vector  $\mathbf{x}$ . Then the matrices in the factorization (2.18) is applied in a “kernel FFT function” (and we will have many such kernels), which assumes that the input has been bit-reversed. A simple implementation of the general function can be as follows.

```
def FFTImpl(x, FFTKernel):
    bitreverse(x)
    FFTKernel(x)
```

A simple implementation of the kernel FFT function, based on the first FFT algorithm we stated, can be as follows.

```
def FFTKernelStandard(x):
    N = len(x)
    if N > 1:
        xe, xo = x[0:(N/2)], x[(N/2):]
        FFTKernelStandard(xe)
        FFTKernelStandard(xo)
        D = exp(-2*pi*1j*arange(float(N/2))/N)
        xo *= D
        x[:] = concatenate([xe + xo, xe - xo])
```

In Exercise 2.23 we will extend these to the general implementations we will use later. We can now run the FFT by combining the general function and the kernel as follows:

```
FFTImpl(x, FFTKernelStandard)
```

Note that `FFTKernelStandard` is recursive; it calls itself. If this is your first encounter with a recursive program, it is worth running through the code manually for a given value of  $N$ , such as  $N = 4$ .

Immediately we see from factorization (2.18) two possible implementations for a kernel. First, as we did, we can apply the FFT recursively. A second way is to, instead of using recursive function calls, use a for-loop where we at each stage in the loop compute the product with one matrix in factorization (2.18), from right to left. Inside this loop there must be another for-loop, where the different blocks in this matrix are applied. We will establish this non-recursive implementation in Exercise 2.28, and see that this leads to a more efficient algorithm.

Python has built-in functions for computing the DFT and the IDFT using the FFT algorithm. These reside in the module `numpy`. The functions are called `fft` and `ifft`. These functions make no assumption about the length of the vector, i.e. it may not be of even length. The implementation may however check if the length of the vector is  $2^r$ , and in those cases variants of the algorithm discussed here can be used. In general, fast algorithms exist when the vector length  $N$  can be factored as a product of small integers.

### 2.3.1 Reduction in the number of arithmetic operations

Now we will explain why the FFT and IFFT factorizations reduce the number of arithmetic operations when compared to direct DFT and IDFT implementations.

We will assume that  $\mathbf{x} \in \mathbb{R}^N$  with  $N$  a power of 2, so that the FFT algorithm can be used recursively, all the way down to vectors of length 1. In many settings this power of 2 assumption can be done. As an example, in compression of sound, one restricts processing to a certain block of the sound data, since the entire sound is too big to be processed in one piece. One then has a freedom to how big these blocks are made, and for optimal speed one often uses blocks of length  $2^r$  with  $r$  some integer in the range 5–10. At the end of this section we will explain how the more general FFT can be computed when  $N$  is not a power of 2.

We first need some terminology for how we count the number of operations of a given type in an algorithm. In particular we are interested in the limiting behaviour when  $N$  becomes large, which is the motivation for the following definition.

**Definition 2.19.** *Order of an algorithm.*

Let  $R_N$  be the number of operations of a given type (such as multiplication or addition) in an algorithm, where  $N$  describes the dimension of the data (such as the size of the matrix or length of the vector), and let  $f$  be a positive function. The algorithm is said to be of order  $f(N)$ , also written  $O(f(N))$ , if the number of operations grows as  $f(N)$  for large  $N$ , or more precisely, if

$$\lim_{N \rightarrow \infty} \frac{R_N}{f(N)} = 1.$$

In some situations we may count the number of operations exactly, but we will also see that it may be easier to obtain the order of the algorithm, since the number of operations may have a simpler expression in the limit. Let us see how we can use this terminology to describe the complexity of the FFT algorithm. Let  $M_N$  and  $A_N$  denote the number of real multiplications and real additions, respectively, required by the FFT algorithm. Once the FFT's of order  $N/2$  have been computed ( $M_{N/2}$  real multiplications and  $A_{N/2}$  real additions are needed for each), it is clear from equations (2.13)-(2.14) that an additional  $N$  complex additions, and an additional  $N/2$  complex multiplications, are required. Since one complex multiplication requires 4 real multiplications and 2 real additions, and one complex addition requires two real additions, we see that we require an additional  $2N$  real multiplications, and  $2N + N = 3N$  real additions. This means that we have the difference equations

$$M_N = 2M_{N/2} + 2N \qquad A_N = 2A_{N/2} + 3N. \qquad (2.19)$$

Note that  $e^{-2\pi i/N}$  may be computed once and for all and outside the algorithm, and this is the reason why we have not counted these operations.

The following example shows how the difference equations (2.19) can be solved. It is not too difficult to argue that  $M_N = O(2N \log_2 N)$  and  $A_N = O(3N \log_2 N)$ , by noting that there are  $\log_2 N$  levels in the FFT, with  $2N$  real multiplications and real  $3N$  additions at each level. But for  $N = 2$  and  $N = 4$  we may actually

avoid some multiplications, so we should solve these equations by stating initial conditions carefully, in order to obtain exact operation counts. In practice, and as we will see later, one often has more involved equations than (2.19), for which the solution can not be seen directly, so that one needs to apply systematic mathematical methods instead. Below is shown an example of this.

**Solving for the number of operations.** To use standard solution methods for difference equations to equations (2.19), we first need to write them in a standard form. Assuming that  $A_N$  and  $M_N$  are powers of 2, we set  $N = 2^r$  and  $x_r = M_{2^r}$ , or  $x_r = A_{2^r}$ . The difference equations can then be rewritten as  $x_r = 2x_{r-1} + 2 \cdot 2^r$  for multiplications, and  $x_r = 2x_{r-1} + 3 \cdot 2^r$  for additions, and again be rewritten in the standard forms

$$x_{r+1} - 2x_r = 4 \cdot 2^r \qquad x_{r+1} - 2x_r = 6 \cdot 2^r.$$

The homogeneous equation  $x_{r+1} - 2x_r = 0$  has the general solution  $x_r^h = C2^r$ . Since the base in the power on the right hand side equals the root in the homogeneous equation, we should in each case guess for a particular solution on the form  $(x_p)_r = Ar2^r$ . If we do this we find that the first equation has particular solution  $(x_p)_r = 2r2^r$ , while the second has particular solution  $(x_p)_r = 3r2^r$ . The general solutions are thus on the form  $x_r = 2r2^r + C2^r$ , for multiplications, and  $x_r = 3r2^r + C2^r$  for additions.

Now let us state initial conditions for the number of additions and multiplications. Example 2.3 showed that floating point multiplication can be avoided completely for  $N = 4$ . We can therefore use  $M_4 = x_2 = 0$  as an initial value. This gives,  $x_r = 2r2^r - 4 \cdot 2^r$ , so that  $M_N = 2N \log_2 N - 4N$ .

For additions we can use  $A_2 = x_1 = 4$  as initial value (since  $\text{DFT}_2(x_1, x_2) = (x_1 + x_2, x_1 - x_2)$ ), which gives  $x_r = 3r2^r$ , so that  $A_N = 3N \log_2 N - N$ . Our FFT algorithm thus requires slightly more additions than multiplications. FFT algorithms are often characterized by their *operation count*, i.e. the total number of real additions and real multiplications, i.e.  $R_N = M_N + A_N$ . We see that  $R_N = 5N \log_2 N - 5N$ . The order of the operation count of our algorithm can thus be written as  $O(5N \log_2 N)$ , since  $\lim_{N \rightarrow \infty} \frac{5N \log_2 N - 5N}{5N \log_2 N} = 1$ .

In practice one can reduce the number of multiplications further, since  $e^{-2\pi in/N}$  take the simple values  $1, -1, -i, i$  for some  $n$ . One can also use that  $e^{-2\pi in/N}$  can take the simple values  $\pm 1/\sqrt{2} \pm 1/\sqrt{2}i = 1/\sqrt{2}(\pm 1 \pm i)$ , which also saves some floating point multiplication, due to that we can factor out  $1/\sqrt{2}$ . These observations do not give big reductions in the arithmetic complexity, however, and one can show that the operation count is still  $O(5N \log_2 N)$  after using these observations.

It is straightforward to show that the IFFT implementation requires the same operation count as the FFT algorithm.

In contrast, the direct implementation of the DFT requires  $N^2$  complex multiplications and  $N(N-1)$  complex additions. This results in  $4N^2$  real multiplications and  $2N^2 + 2N(N-1) = 4N^2 - 2N$  real additions. The total

operation count is thus  $8N^2 - 2N$ . In other words, the FFT and IFFT significantly reduce the number of arithmetic operations. In Exercise 2.29 we present another algorithm, called the Split-radix algorithm, which reduces the number of operations even further. We will see, however, the reduction obtained with the split-radix algorithm is about 20%. Let us summarize our findings as follows.

**Theorem 2.20.** *Number of operations in the FFT and IFFT algorithms.*

The  $N$ -point FFT and IFFT algorithms we have gone through both require  $O(2N \log_2 N)$  real multiplications and  $O(3N \log_2 N)$  real additions. In comparison, the number of real multiplications and real additions required by direct implementations of the  $N$ -point DFT and IDFT are  $O(8N^2)$ .

Often we apply the DFT for real data, so we would like to have FFT-algorithms tailored to this, with reduced complexity (since real data has half the dimension of general complex data). By some it has been argued that one can find improved FFT algorithms when one assumes that the data is real. In Exercise 2.27 we address this issue, and conclude that there is little to gain from assuming real input: The general algorithm for complex input can be tailored for real input so that it uses half the number of operations, which harmonizes with the fact that real data has half the dimension of complex data.

Another reason why the FFT is efficient is that, since the FFT splits the calculation of the DFT into computing two DFT's of half the size, the FFT is well suited for parallel computing: the two smaller FFT's can be performed independently of one another, for instance in two different computing cores on the same computer. Besides reducing the number of arithmetic operations, FFT implementation can also apply several programming tricks to speed up computation, see for instance <http://cnx.org/content/m12021/latest/> for an overview.

### 2.3.2 The FFT when $N$ is not a power of 2

Applying an FFT to a vector of length  $2^n$  is by far the most common thing to do. It turns out, however, that the idea behind the algorithm easily carries over to the case when  $N$  is any composite number, i.e. when  $N = N_1 N_2$ . This makes the FFT useful also in settings where we have a dictated number of elements in  $\mathbf{x}$ , which is not an even number. The approach we will present in this section will help us as long as  $N$  is not a prime number. The case when  $N$  is a prime number needs other techniques.

So, assume that  $N = N_1 N_2$ . Any time-index  $k$  can be written uniquely on the form  $N_1 k + p$ , with  $0 \leq k < N_2$ , and  $0 \leq p < N_1$ . We will make the following definition.

**Definition 2.21.** *Polyphase components of a vector.*

Let  $\mathbf{x} \in \mathbb{R}^{N_1 N_2}$ . We denote by  $\mathbf{x}^{(p)}$  the vector in  $\mathbb{R}^{N_2}$  with entries  $(\mathbf{x}^{(p)})_k = x_{N_1 k + p}$ .  $\mathbf{x}^{(p)}$  is also called the  $p$ 'th polyphase component of  $\mathbf{x}$ .

The previous vectors  $\mathbf{x}^{(e)}$  and  $\mathbf{x}^{(o)}$  can be seen as special cases of polyphase components. Polyphase components will also be useful later. Using the polyphase notation, we can write

$$\begin{aligned} \text{DFT}_N \mathbf{x} &= \sum_{k=0}^{N-1} x_k e^{-2\pi i n k / N} = \sum_{p=0}^{N_1-1} \sum_{k=0}^{N_2-1} (\mathbf{x}^{(p)})_k e^{-2\pi i n (N_1 k + p) / N} \\ &= \sum_{p=0}^{N_1-1} e^{-2\pi i n p / N} \sum_{k=0}^{N_2-1} (\mathbf{x}^{(p)})_k e^{-2\pi i n k / N_2} \end{aligned}$$

Similarly, any frequency index  $n$  can be written uniquely on the form  $N_2 q + n$ , with  $0 \leq q < N_1$ , and  $0 \leq n < N_2$ , so that the DFT can also be written as

$$\begin{aligned} &\sum_{p=0}^{N_1-1} e^{-2\pi i (N_2 q + n) p / N} \sum_{k=0}^{N_2-1} (\mathbf{x}^{(p)})_k e^{-2\pi i (N_2 q + n) k / N_2} \\ &= \sum_{p=0}^{N_1-1} e^{-2\pi i q p / N_1} e^{-2\pi i n p / N} \sum_{k=0}^{N_2-1} (\mathbf{x}^{(p)})_k e^{-2\pi i n k / N_2}. \end{aligned}$$

Now, if  $X$  is the  $N_2 \times N_1$ -matrix  $X$  where the  $p$ 'th column is  $\mathbf{x}^{(p)}$ , we recognize the inner sum  $\sum_{k=0}^{N_2-1} (\mathbf{x}^{(p)})_k e^{-2\pi i n k / N_2}$  as matrix multiplication with  $\text{DFT}_{N_2}$  and  $X$ , so that this can be written as  $(\text{DFT}_{N_2} X)_{n,p}$ . The entire sum can thus be written as

$$\sum_{p=0}^{N_1-1} e^{-2\pi i q p / N_1} e^{-2\pi i n p / N} (\text{DFT}_{N_2} X)_{n,p}.$$

Now, define  $Y$  as the matrix where  $X$  is multiplied component-wise with the matrix with  $(n, p)$ -component  $e^{-2\pi i n p / N}$ . The entire sum can then be written as

$$\sum_{p=0}^{N_1-1} e^{-2\pi i q p / N_1} Y_{n,p} = (Y F_{N_1})_{n,q}$$

This means that the sum can be written as component  $(n, q)$  in the matrix  $Y F_{N_1}$ . Clearly  $Y F_{N_1}$  is the matrix where the DFT is applied to all rows of  $Y$ . We have thus shown that component  $N_2 q + n$  of  $F_N \mathbf{x}$  equals  $(Y F_{N_1})_{n,q}$ . This means that  $F_N \mathbf{x}$  can be obtained by stacking the columns of  $Y F_{N_1}$  on top of one-another. We can thus summarize our procedure as follows, which gives a recipe for splitting an FFT into smaller FFT's when  $N$  is not a prime number.

**Theorem 2.22.** *FFT algorithm when  $N$  is composite.*

When  $N = N_1 N_2$ , the FFT of a vector  $\mathbf{x}$  can be computed as follows

- Form the  $N_2 \times N_1$ -matrix  $X$ , where the  $p$ 'th column is  $\mathbf{x}^{(p)}$ .



- Perform the DFT on all the columns in  $X$ , i.e. compute  $F_{N_2}X$ .
- Multiply element  $(n, p)$  in the resulting matrix with  $e^{-2\pi inp/N}$  (these are called *twiddle factors*), to obtain matrix  $Y$ .
- Perform the DFT on all the rows in the resulting matrix, i.e. compute  $YF_{N_1}$ .
- Form the vector where the columns of the resulting matrix are stacked on top of one-another.

From the algorithm one easily deduces how the IDFT can be computed also: All steps are invertible, and can be performed by IFFT or multiplication. We thus only need to perform the inverse steps in reverse order.

### Exercise 2.22: Extra results for the FFT when $N = N_1N_2$

When  $N$  is composite, there are a couple of results we can state regarding polyphase components.

a) Assume that  $N = N_1N_2$ , and that  $\mathbf{x} \in \mathbb{R}^N$  satisfies  $x_{k+rN_1} = x_k$  for all  $k, r$ , i.e.  $\mathbf{x}$  has period  $N_1$ . Show that  $y_n = 0$  for all  $n$  which are not a multiple of  $N_2$ .

b) Assume that  $N = N_1N_2$ , and that  $\mathbf{x}^{(p)} = \mathbf{0}$  for  $p \neq 0$ . Show that the polyphase components  $\mathbf{y}^{(p)}$  of  $\mathbf{y} = \text{DFT}_N \mathbf{x}$  are constant vectors for all  $p$ .

But what about the case when  $N$  is a prime number? Rader's algorithm [38] handles this case by expressing a DFT with  $N$  a prime number in terms of DFT's of length  $N - 1$  (which is not a prime number). Our previous scenario can then be followed, but stops quickly again if  $N - 1$  has prime factors of high order. Since there are some computational penalties in applying Rader's algorithm, it may be inefficient some cases. Winograd's FFT algorithm [50] extends Rader's algorithm to work for the case when  $N = p^r$ . This algorithm tends to reduce the number of multiplications, at the price of an increased number of additions. It is difficult to program, and is rarely used in practice.

### Exercise 2.23: Extend implementation

Recall that, in Exercise 2.13, we extended the direct DFT implementation so that it accepted a second parameter telling us if the forward or reverse transform should be applied. Extend the general function and the standard kernel in the same way. Again, the forward transform should be used if the `forward` parameter is not present. Assume also that the kernel accepts only one-dimensional data, and that the general function applies the kernel to each column in the input if the input is two-dimensional (so that the FFT can be applied to all channels in a sound with only one call). The signatures for our methods should thus be changed as follows:

```
def FFTImpl(x, FFTKernel, forward = True):
def FFTKernelStandard(x, forward):
```

It should be straightforward to make the modifications for the reverse transform by consulting the second part of Theorem 2.18. For simplicity, let `FFTImpl` take care of the additional division with  $N$  we need to do in case of the IDFT. In the following we will assume these signatures for the FFT implementation and the corresponding kernels.

### Exercise 2.24: Compare execution time

In this exercise we will compare execution times for the different methods for computing the DFT.

a) Write code which compares the execution times for an  $N$ -point DFT for the following three cases: Direct implementation of the DFT (as in Example 2.4), the FFT implementation used in this chapter, and the built-in `fft`-function. Your code should use the sample audio file `castanets.wav`, apply the different DFT implementations to the first  $N = 2^r$  samples of the file for  $r = 3$  to  $r = 15$ , store the execution times in a vector, and plot these. You can use the function `time()` in the `time` module to measure the execution time.

b) A problem for large  $N$  is that there is such a big difference in the execution times between the two implementations. We can address this by using a loglog-plot instead. Plot  $N$  against execution times using the function `loglog`. How should the fact that the number of arithmetic operations are  $8N^2$  and  $5N \log_2 N$  be reflected in the plot?

c) It seems that the built-in FFT is much faster than our own FFT implementation, even though they may use similar algorithms. Try to explain what can be the cause of this.

### Exercise 2.25: Combine two FFT's

Let  $\mathbf{x}_1 = (1, 3, 5, 7)$  and  $\mathbf{x}_2 = (2, 4, 6, 8)$ . Compute  $\text{DFT}_4 \mathbf{x}_1$  and  $\text{DFT}_4 \mathbf{x}_2$ . Explain how you can compute  $\text{DFT}_8(1, 2, 3, 4, 5, 6, 7, 8)$  based on these computations (you don't need to perform the actual computation). What are the benefits of this approach?

### Exercise 2.26: FFT operation count

When we wrote down the difference equation for the number of multiplications in the FFT algorithm, you could argue that some multiplications were not counted. Which multiplications in the FFT algorithm were not counted when writing down this difference equation? Do you have a suggestion to why these multiplications were not counted?

**Exercise 2.27: FFT algorithm adapted to real data**

In this exercise we will look at how we can adapt an FFT algorithm to real input. Since  $y_{N-n} = \overline{y_n}$  for real input, there is no additional complexity in computing the second half of the DFT coefficients, once the first half has been computed. We will now rewrite Equation (2.13) for indices  $n$  and  $N/2 - n$  as

$$\begin{aligned} y_n &= (\text{DFT}_{N/2} \mathbf{x}^{(e)})_n + e^{-2\pi i n/N} (\text{DFT}_{N/2} \mathbf{x}^{(o)})_n \\ y_{N/2-n} &= (\text{DFT}_{N/2} \mathbf{x}^{(e)})_{N/2-n} + e^{-2\pi i (N/2-n)/N} (\text{DFT}_{N/2} \mathbf{x}^{(o)})_{N/2-n} \\ &= \overline{(\text{DFT}_{N/2} \mathbf{x}^{(e)})_n} - e^{2\pi i n/N} \overline{(\text{DFT}_{N/2} \mathbf{x}^{(o)})_n} \\ &= \overline{(\text{DFT}_{N/2} \mathbf{x}^{(e)})_n} - \overline{e^{-2\pi i n/N} (\text{DFT}_{N/2} \mathbf{x}^{(o)})_n}. \end{aligned}$$

We see here that, if we already have computed  $\text{DFT}_{N/2} \mathbf{x}^{(e)}$  and  $\text{DFT}_{N/2} \mathbf{x}^{(o)}$ , we need one additional complex multiplication for each  $y_n$  with  $0 \leq n < N/4$  (since  $e^{-2\pi i n/N}$  and  $(\text{DFT}_{N/2} \mathbf{x}^{(o)})_n$  are complex). No further multiplications are needed in order to compute  $y_{N/2-n}$ , since we simply conjugate terms before adding them. Again  $y_{N/2}$  must be handled explicitly with this approach. For this we can use the formula

$$y_{N/2} = (\text{DFT}_{N/2} \mathbf{x}^{(e)})_0 - (D_{N/2} \text{DFT}_{N/2} \mathbf{x}^{(o)})_0$$

instead.

**a)** Conclude from this that an FFT algorithm adapted to real data at each step requires  $N/4$  complex additions and  $N/2$  complex additions. Conclude from this as before that an algorithm based on real data requires  $M_N = O(N \log_2 N)$  real multiplications and  $A_N = O(\frac{3}{2} N \log_2 N)$  real additions (i.e. half the operation count for complex input).

**b)** Find an IFFT algorithm adapted to vectors with conjugate symmetry, which has the same operation count as this FFT algorithm adapted to real data.

**Hint.** Consider the vectors  $\mathbf{z}$ ,  $\mathbf{w}$  with entries  $z_n = y_n + \overline{y_{N/2-n}} \in \mathbb{R}^{N/2}$  and  $w_n = e^{2\pi i n/N} (y_n - \overline{y_{N/2-n}}) \in \mathbb{R}^{N/2}$ . From the equations above, how can these be used in an IFFT?

**Exercise 2.28: Non-recursive FFT algorithm**

Use the factorization in (2.18) to write a kernel function `FFTKernelNonrec` for a non-recursive FFT implementation. In your code, perform the matrix multiplications in Equation (2.18) from right to left in an (outer) for-loop. For each matrix loop through the different blocks on the diagonal in an (inner) for-loop. Make sure you have the right number of blocks on the diagonal, each block being on the form

$$\begin{pmatrix} I & D_{N/2^k} \\ I & -D_{N/2^k} \end{pmatrix}.$$

It may be a good idea to start by implementing multiplication with such a simple matrix first as these are the building blocks in the algorithm (also attempt to do this so that everything is computed in-place). Also compare the execution times with our original FFT algorithm, as we did in Exercise 2.24, and try to explain what you see in this comparison.

### Exercise 2.29: The Split-radix FFT algorithm

In this exercise we will develop a variant of the FFT algorithm called the *split-radix FFT algorithm*, which until recently held the record for the lowest operation count for any FFT algorithm.

We start by splitting the rightmost  $\text{DFT}_{N/2}$  in Equation (2.17) by using this equation again, to obtain

$$\text{DFT}_N \mathbf{x} = \begin{pmatrix} \text{DFT}_{N/2} & D_{N/2} \begin{pmatrix} \text{DFT}_{N/4} & D_{N/4} \text{DFT}_{N/4} \\ \text{DFT}_{N/4} & -D_{N/4} \text{DFT}_{N/4} \end{pmatrix} \\ \text{DFT}_{N/2} & -D_{N/2} \begin{pmatrix} \text{DFT}_{N/4} & D_{N/4} \text{DFT}_{N/4} \\ \text{DFT}_{N/4} & -D_{N/4} \text{DFT}_{N/4} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(e)} \\ \mathbf{x}^{(oe)} \\ \mathbf{x}^{(oo)} \end{pmatrix}. \quad (2.20)$$

The term *radix* describes how an FFT is split into FFT's of smaller sizes, i.e. how the sum in an FFT is split into smaller sums. The FFT algorithm we started this section with is called a *radix 2* algorithm, since it splits an FFT of length  $N$  into FFT's of length  $N/2$ . If an algorithm instead splits into FFT's of length  $N/4$ , it is called a *radix 4* FFT algorithm. The algorithm we go through here is called the *split radix* algorithm, since it uses FFT's of both length  $N/2$  and  $N/4$ .

**a)** Let  $G_{N/4}$  be the  $(N/4) \times (N/4)$  diagonal matrix with  $e^{-2\pi i n/N}$  on the diagonal. Show that  $D_{N/2} = \begin{pmatrix} G_{N/4} & \mathbf{0} \\ \mathbf{0} & -iG_{N/4} \end{pmatrix}$ .

**b)** Let  $H_{N/4}$  be the  $(N/4) \times (N/4)$  diagonal matrix  $G_{D/4} D_{N/4}$ . Verify the following rewriting of Equation (2.20):

$$\begin{aligned}
 \text{DFT}_N \mathbf{x} &= \begin{pmatrix} \text{DFT}_{N/2} & \begin{pmatrix} G_{N/4} \text{DFT}_{N/4} & H_{N/4} \text{DFT}_{N/4} \\ -iG_{N/4} \text{DFT}_{N/4} & iH_{N/4} \text{DFT}_{N/4} \end{pmatrix} \\ \text{DFT}_{N/2} & \begin{pmatrix} -G_{N/4} \text{DFT}_{N/4} & -H_{N/4} \text{DFT}_{N/4} \\ iG_{N/4} \text{DFT}_{N/4} & -iH_{N/4} \text{DFT}_{N/4} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(e)} \\ \mathbf{x}^{(oe)} \\ \mathbf{x}^{(oo)} \end{pmatrix} \\
 &= \begin{pmatrix} I & \mathbf{0} & G_{N/4} & H_{N/4} \\ \mathbf{0} & I & -iG_{N/4} & iH_{N/4} \\ I & \mathbf{0} & -G_{N/4} & -H_{N/4} \\ \mathbf{0} & I & iG_{N/4} & -iH_{N/4} \end{pmatrix} \begin{pmatrix} \text{DFT}_{N/2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \text{DFT}_{N/4} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \text{DFT}_{N/4} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(e)} \\ \mathbf{x}^{(oe)} \\ \mathbf{x}^{(oo)} \end{pmatrix} \\
 &= \begin{pmatrix} I & \begin{pmatrix} G_{N/4} & H_{N/4} \\ -iG_{N/4} & iH_{N/4} \end{pmatrix} \\ I & -\begin{pmatrix} G_{N/4} & H_{N/4} \\ -iG_{N/4} & iH_{N/4} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \text{DFT}_{N/2} \mathbf{x}^{(e)} \\ \text{DFT}_{N/4} \mathbf{x}^{(oe)} \\ \text{DFT}_{N/4} \mathbf{x}^{(oo)} \end{pmatrix} \\
 &= \begin{pmatrix} \text{DFT}_{N/2} \mathbf{x}^{(e)} + \begin{pmatrix} G_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oe)} + H_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oo)} \\ -i(G_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oe)} - H_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oo)}) \end{pmatrix} \\ \text{DFT}_{N/2} \mathbf{x}^{(e)} - \begin{pmatrix} G_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oe)} + H_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oo)} \\ -i(G_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oe)} - H_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oo)}) \end{pmatrix} \end{pmatrix}
 \end{aligned}$$

c) Explain from the above expression why, once the three FFT's above have been computed, the rest can be computed with  $N/2$  complex multiplications, and  $2 \times N/4 + N = 3N/2$  complex additions. This is equivalent to  $2N$  real multiplications and  $N + 3N = 4N$  real additions.

**Hint.** It is important that  $G_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oe)}$  and  $H_{N/4} \text{DFT}_{N/4} \mathbf{x}^{(oo)}$  are computed first, and the sum and difference of these two afterwards.

d) Due to what we just showed, our new algorithm leads to real multiplication and addition counts which satisfy

$$M_N = M_{N/2} + 2M_{N/4} + 2N \quad A_N = A_{N/2} + 2A_{N/4} + 4N$$

Find the general solutions to these difference equations and conclude from these that  $M_N = O(\frac{4}{3}N \log_2 N)$ , and  $A_N = O(\frac{8}{3}N \log_2 N)$ . The operation count is thus  $O(4N \log_2 N)$ , which is a reduction of  $N \log_2 N$  from the FFT algorithm.

e) Write an FFT kernel function `FFTKernelSplitradix` for the split-radix algorithm (again this should handle both the forward and reverse transforms). Are there more or less recursive function calls in this function than in the original FFT algorithm? Also compare the execution times with our original FFT algorithm, as we did in Exercise 2.24. Try to explain what you see in this comparison.

By carefully examining the algorithm we have developed, one can reduce the operation count to  $4N \log_2 N - 6N + 8$ . This does not reduce the order of the algorithm, but for small  $N$  (which often is the case in applications) this

reduces the number of operations considerably, since  $6N$  is large compared to  $4N \log_2 N$  for small  $N$ . In addition to having a lower number of operations than the FFT algorithm of Theorem 2.15, a bigger percentage of the operations are additions for our new algorithm: there are now twice as many additions than multiplications. Since multiplications may be more time-consuming than additions (depending on how the CPU computes floating-point arithmetic), this can be a big advantage.

### Exercise 2.30: Bit-reversal

In this exercise we will make some considerations which will help us explain the code for bit-reversal. This is perhaps not a mathematically challenging exercise, but nevertheless a good exercise in how to think when developing an efficient algorithm. We will use the notation  $i$  for an index, and  $j$  for its bit-reverse. If we bit-reverse  $k$  bits, we will write  $N = 2^k$  for the number of possible indices.

a) Consider the following code

```

j = 0
for i in range(N-1):
    print j
    m = N/2
    while (m >= 1 and j >= m):
        j -= m
        m /= 2
    j += m

```

Explain that the code prints all numbers in  $[0, N - 1]$  in bit-reversed order (i.e.  $j$ ). Verify this by running the program, and writing down the bits for all numbers for, say  $N = 16$ . In particular explain the decrements and increments made to the variable  $j$ . The code above thus produces pairs of numbers  $(i, j)$ , where  $j$  is the bit-reverse of  $i$ . As can be seen, `bitreverse` applies similar code, and then swaps the values  $x_i$  and  $x_j$  in  $\mathbf{x}$ , as it should.

Since bit-reverse is its own inverse (i.e.  $P^2 = I$ ), it can be performed by swapping elements  $i$  and  $j$ . One way to secure that bit-reverse is done only once, is to perform it only when  $j > i$ . You see that `bitreverse` includes this check.

b) Explain that  $N - j - 1$  is the bit-reverse of  $N - i - 1$ . Due to this, when  $i, j < N/2$ , we have that  $N - i - 1, N - j - 1 \geq N/2$ , and that `bitreversal` can swap them. Moreover, all swaps where  $i, j \geq N/2$  can be performed immediately when pairs where  $i, j < N/2$  are encountered. Explain also that  $j < N/2$  if and only if  $i$  is even. In the code you can see that the swaps  $(i, j)$  and  $(N - i - 1, N - j - 1)$  are performed together when  $i$  is even, due to this.

c) Assume that  $i < N/2$  is odd. Explain that  $j \geq N/2$ , so that  $j > i$ . This says that when  $i < N/2$  is odd, we can always swap  $i$  and  $j$  (this is the last swap performed in the code). All swaps where  $0 \leq j < N/2$  and  $N/2 \leq j < N$  can be performed in this way.

In `bitreversal`, you can see that the bit-reversal of  $2r$  and  $2r+1$  are handled together (i.e.  $i$  is increased with 2 in the `for`-loop). The effect of this is that the number of `if`-tests can be reduced, due to the observations from b) and c).

## 2.4 Summary

We considered the analog of Fourier series for digital sound, which is called the Discrete Fourier Transform, and looked at its properties and its relation to Fourier series. We also saw that the sampling theorem guaranteed that there is no loss in considering the samples of a function, as long as the sampling rate is high enough compared to the highest frequency in the sound.

We obtained an implementation of the DFT, called the FFT, which is more efficient in terms of the number of arithmetic operations than a direct implementation of the DFT. The FFT has been cited as one of the ten most important algorithms of the 20<sup>th</sup> century [5]. The original paper [8] by Cooley and Tukey dates back to 1965, and handles the case when  $N$  is composite. In the literature, one has been interested in the FFT algorithms where the number of (real) additions and multiplications (combined) is as low as possible. This number is also called the *flop count*. The presentation in this book thus differs from the literature in that we mostly count only the number of multiplications. The split-radix algorithm [51, 14], which we reviewed in Exercise 2.3.29, held the record for the lowest flop count until quite recently. In [22], Frigo and Johnson showed that the operation count can be reduced to  $O(34N \log_2(N)/9)$ , which clearly is less than the  $O(4N \log_2 N)$  we obtained for the split-radix algorithm. It may seem strange that the total number of additions and multiplications are considered: Aren't multiplications more time-consuming than additions? When you consider how this is done mechanically, this is certainly the case: In fact, floating point multiplication can be considered as a combination of many floating point additions. Due to this, one can find many places in the literature where expressions are rewritten so that the multiplication count is reduced, at the cost of a higher addition count. Winograd's algorithm [50] is an example of this, where the number of additions is much higher than the number of multiplications. However, most modern CPU's have more complex hardware dedicated to computing multiplications, which can result in that one floating point multiplication can be performed in one cycle, just as one addition can. Another thing is that modern CPU's typically can perform many additions and multiplications in parallel, and the higher complexity in the multiplication hardware may result in that the CPU can run less multiplications in parallel, compared to additions. In other words, if we run test program on a computer, it may be difficult to detect any differences in performance between addition and multiplication, even though complex big-scale computing should in theory show some differences. There are also other important aspects of the FFT, besides the flop count. Another is memory use. It is possible to implement the FFT so that the output is computed into the same memory as the input, so that the

FFT algorithm does not require extra memory besides the input buffer. Clearly, one should bit-reverse the input buffer in order to achieve this.

We have now defined two types of transforms to the frequency domain: Fourier series for continuous, periodic functions, and the DFT, for periodic vectors. In the literature there are two other transforms also: The Continuous time Fourier transform (CTFT) we have already mentioned at the end of Chapter 1. We also have the Discrete time Fourier transform (DTFT) for vectors which are not periodic [37]. In this book we will deliberately avoid the DTFT as well, since it assumes that the signal to transform is of infinite duration, while we in practice analyze signals with a limited time scope.

The sampling theorem is also one of the most important results of the last century. It was discovered by Harry Nyquist and Claude Shannon [42], but also by others independently. One can show that the sampling theorem holds also for functions which are not periodic, as long as we have the same bound on the highest frequency. This is more common in the literature. In fact, the proof seen here where we restrict to periodic functions is not common. The advantage of the proof seen here is that we remain in a finite dimensional setting, and that we only need the DFT. More generally, proofs of the sampling theorem in the literature use the DTFT and the CTFT.

### What you should have learned in this chapter.

- The definition of the Fourier basis and its orthonormality.
- The definition of the Discrete Fourier Transform as a change of coordinates to the Fourier basis, its inverse, and its unitarity.
- How to apply the DFT to a sum of sinusoids.
- Properties of the DFT, such as conjugate symmetry when the vector is real, how it treats delayed vectors, or vectors multiplied with a complex exponential.
- Translation between DFT index and frequency. In particular DFT indices for high and low frequencies.
- How one can use the DFT to adjust frequencies in sound.
- How the FFT algorithm works by splitting into two FFT's of half the length.
- Simple FFT implementation.
- Reduction in the number of operations with the FFT.



## Chapter 3

# Operations on digital sound: digital filters

In Section 1.5 we defined filters as operations on continuous sound which preserved different frequencies. Such operations are important since they can change the frequency content in many ways. They are difficult to use computationally, however, since they are defined for all instances in time. As when we defined the DFT to make Fourier series computable, we would like to define *digital filters*, in order to make filters computable. It will turn out that such digital filters can be computed by the following procedure:

$$z_n = \frac{1}{4}(x_{n-1} + 2x_n + x_{n+1}), \quad \text{for } n = 0, 1, \dots, N-1. \quad (3.1)$$

Here  $\mathbf{x}$  denotes the *input vector*, and  $\mathbf{z}$  the *output vector*. In other words, the output of a digital filter is constructed by combining several input elements linearly. The concrete filter defined by Equation (3.1) is called a *smoothing filter*, as we will demonstrate that it smooths the variations in the sound. We will start this chapter by looking at matrix representations for operations as given by Equation (3.1). Then we will formally define digital filters in terms of preservation of frequencies as we did for the filters in Chapter 1, and show that this is equivalent to operations on the form (3.1).

### 3.1 Matrix representations of filters

Let us consider Equation (3.1) in some more detail to get more intuition about filters. As before we assume that the input vector is periodic with period  $N$ , so that  $x_{n+N} = x_n$ . Our first observation is that the output vector  $\mathbf{z}$  is also periodic with period  $N$  since

$$z_{n+N} = \frac{1}{4}(x_{n+N-1} + 2x_{n+N} + x_{n+N+1}) = \frac{1}{4}(x_{n-1} + 2x_n + x_{n+1}) = z_n.$$

The filter is also clearly a linear transformation and may therefore be represented by an  $N \times N$  matrix  $S$  that maps the vector  $\mathbf{x} = (x_0, x_1, \dots, x_{N-1})$  to the vector  $\mathbf{z} = (z_0, z_1, \dots, z_{N-1})$ , i.e., we have  $\mathbf{z} = S\mathbf{x}$ . To find  $S$ , for  $1 \leq n \leq N - 2$  it is clear from Equation (3.1) that row  $n$  has the value  $1/4$  in column  $n - 1$ , the value  $1/2$  in column  $n$ , and the value  $1/4$  in column  $n + 1$ . For row 0 we must be a bit more careful, since the index  $-1$  is outside the legal range of the indices. This is where the periodicity helps us out so that

$$z_0 = \frac{1}{4}(x_{-1} + 2x_0 + x_1) = \frac{1}{4}(x_{N-1} + 2x_0 + x_1) = \frac{1}{4}(2x_0 + x_1 + x_{N-1}).$$

From this we see that row 0 has the value  $1/4$  in columns 1 and  $N - 1$ , and the value  $1/2$  in column 0. In exactly the same way we can show that row  $N - 1$  has the entry  $1/4$  in columns 0 and  $N - 2$ , and the entry  $1/2$  in column  $N - 1$ . In summary, the matrix of the smoothing filter is given by

$$S = \frac{1}{4} \begin{pmatrix} 2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 2 \end{pmatrix}. \quad (3.2)$$

A matrix on this form is called a Toeplitz matrix. The general definition is as follows and may seem complicated, but is in fact quite straightforward:

**Definition 3.1.** *Toeplitz matrices.*

An  $N \times N$ -matrix  $S$  is called a Toeplitz matrix if its elements are constant along each diagonal. More formally,  $S_{k,l} = S_{k+s,l+s}$  for all nonnegative integers  $k, l$ , and  $s$  such that both  $k + s$  and  $l + s$  lie in the interval  $[0, N - 1]$ . A Toeplitz matrix is said to be circulant if in addition

$$S_{(k+s) \bmod N, (l+s) \bmod N} = S_{k,l}$$

for all integers  $k, l$  in the interval  $[0, N - 1]$ , and all  $s$  (Here mod denotes the remainder modulo  $N$ ).

Toeplitz matrices are very popular in the literature and have many applications. A Toeplitz matrix is constant along each diagonal, while the additional property of being circulant means that each row and column of the matrix 'wraps over' at the edges. Clearly the matrix given by Equation (3.2) satisfies Definition 3.1 and is a circulant Toeplitz matrix. A Toeplitz matrix is uniquely identified by the values on its nonzero diagonals, and a circulant Toeplitz matrix is uniquely identified by the values on the main diagonal, and on the diagonals above (or under) it. Toeplitz matrices show up here in the context of filters, but they will also show up later in the context of wavelets.

Equation (3.1) leads us to the more general expression

$$z_n = \sum_k t_k x_{n-k}. \quad (3.3)$$

If  $\mathbf{t}$  has infinitely many nonzero entries, the sum is an infinite one, and may diverge. We will, however, mostly assume that  $\mathbf{t}$  has a finite number of nonzero entries. This general expression opens up for defining many types of operations. The values  $t_k$  will be called *filter coefficients*. The range of  $k$  is not specified, but is typically an interval around 0, since  $z_n$  usually is calculated by combining  $x_k$ 's with indices close to  $n$ . Both positive and negative indices are allowed. As an example, for formula (3.1)  $k$  ranges over  $-1, 0$ , and  $1$ , and we have that  $t_{-1} = t_1 = 1/4$ , and  $t_0 = 1/2$ . Since Equation (3.3) needs to be computed for each  $n$ , if only  $t_0, \dots, t_{kmax}$  are nonzero, we need to go through the following for-loop to compute  $z_{kmax}, \dots, z_{N-1}$ :

```
z = zeros_like(x)
for n in range(kmax, N):
    for k in range(kmax + 1):
        z[n] += t[k]*x[n - k]
```

It is clearly possible to vectorize the inner loop here, since it takes the form of a dot product. Another possible way to vectorize is to first change the order of summation, and then vectorize as follows

```
z = zeros_like(x)
for k in range(kmax + 1):
    z[kmax:N] += t[k]*x[(kmax-k):(N-k)]
```

Depending on how vectorization is supported, this code will in general execute faster, and is to prefer. The drawback, however, is that a filter often is applied in real time, with the output computed only when enough input is available, with the input becoming available continuously. This second approach then clearly fails, since it computes nothing before all input is available. In the exercise we will compare the computation times for the two approaches above, and compare them with a built-in function which computes the same.

Note that above we did not consider the first entries in  $\mathbf{z}$ , since this is where the circulation occurs. Taken this into account, the first filter we considered in this chapter can be implemented in vectorized form simply as

```
z[0]      = x[1]/4. + x[0]/2. + x[N-1]/4.
z[1:(N-1)] = x[2:N]/4. + x[1:(N-1)]/2. + x[0:(N-2)]/4.
z[N-1]    = x[0]/4. + x[N-1]/2. + x[N-2]/4.
```

In the following we will avoid such implementations, since for-loops can be very slow. We will see that an efficient built-in function exists for computing this, and use this instead.

By following the same argument as above, the following is clear:

**Proposition 3.2.** *Filters as matrices.*

Any operation defined by Equation (3.3) is a linear transformation which transforms a vector of period  $N$  to another of period  $N$ . It may therefore be represented by an  $N \times N$  matrix  $S$  that maps the vector  $\mathbf{x} = (x_0, x_1, \dots, x_{N-1})$  to the vector  $\mathbf{z} = (z_0, z_1, \dots, z_{N-1})$ , i.e., we have  $\mathbf{z} = S\mathbf{x}$ . Moreover, the matrix  $S$  is a circulant Toeplitz matrix, and the first column  $\mathbf{s}$  of this matrix is given by

$$s_k = \begin{cases} t_k, & \text{if } 0 \leq k < N/2; \\ t_{k-N}, & \text{if } N/2 \leq k \leq N-1. \end{cases} \quad (3.4)$$

In other words, the first column of  $S$  can be obtained by placing the coefficients in (3.3) with positive indices at the beginning of  $\mathbf{s}$ , and the coefficients with negative indices at the end of  $\mathbf{s}$ .

This proposition will be useful for us, since it explains how to pass from the form (3.3), which is most common in practice, to the matrix form  $S$ . Since the filter coefficients  $t_k$  uniquely define any  $N \times N$ -circulant Toeplitz matrix, we will establish the following shorthand notation for the filter matrix for a given set of filter coefficients. We will use this notation only when we have a finite set of nonzero filter coefficients (note however that many interesting filters in signal processing have infinitely many nonzero filter coefficients, see Section 3.5) Note also that we always choose  $N$  so large that the placement of the filter coefficients in the first column, as dictated by Proposition 3.2, never collide (as happens when  $N$  is smaller than the number of filter coefficients).

**Definition 3.3.** *Compact notation for filters.*

Let  $k_{\min}$ ,  $k_{\max}$  be the smallest and biggest index of a filter coefficient in Equation (3.3) so that  $t_k \neq 0$  (if no such values exist, let  $k_{\min} = k_{\max} = 0$ ), i.e.

$$z_n = \sum_{k=k_{\min}}^{k_{\max}} t_k x_{n-k}. \quad (3.5)$$

We will use the following compact notation for  $S$ :

$$S = \{t_{k_{\min}}, \dots, t_{-1}, \underline{t_0}, t_1, \dots, t_{k_{\max}}\}.$$

In other words, the entry with index 0 has been underlined, and only the nonzero  $t_k$ 's are listed.  $k_{\max}$  and  $k_{\min}$  are also called the start and end indices of  $S$ . By the length of  $S$ , denoted  $l(S)$ , we mean the number  $k_{\max} - k_{\min}$ .

One seldom writes out the matrix of a filter, but rather uses this compact notation.

### Example 3.1: Finding the matrix elements from the filter coefficients

Let us apply Proposition 3.2 to the operation defined by formula (3.1):

- for  $k = 0$  Equation (3.4) gives  $s_0 = t_0 = 1/2$ .

- For  $k = 1$  Equation (3.4) gives  $s_1 = t_1 = 1/4$ .
- For  $k = N - 1$  Equation (3.4) gives  $s_{N-1} = t_{-1} = 1/4$ .

For all  $k$  different from 0, 1, and  $N - 1$ , we have that  $s_k = 0$ . Clearly this gives the matrix in Equation (3.2).

### Example 3.2: Finding the filter coefficients from the matrix

Proposition 3.2 is also useful when we have a circulant Toeplitz matrix  $S$ , and we want to find filter coefficients  $t_k$  so that  $\mathbf{z} = S\mathbf{x}$  can be written on the form (3.3). Consider the matrix

$$S = \begin{pmatrix} 2 & 1 & 0 & 3 \\ 3 & 2 & 1 & 0 \\ 0 & 3 & 2 & 1 \\ 1 & 0 & 3 & 2 \end{pmatrix}.$$

This is a circulant Toeplitz matrix with  $N = 4$ , and we see that  $s_0 = 2$ ,  $s_1 = 3$ ,  $s_2 = 0$ , and  $s_3 = 1$ . The first equation in (3.4) gives that  $t_0 = s_0 = 2$ , and  $t_1 = s_1 = 3$ . The second equation in (3.4) gives that  $t_{-2} = s_2 = 0$ , and  $t_{-1} = s_3 = 1$ . By including only the  $t_k$  which are nonzero, the operation can be written as

$$z_n = t_{-1}x_{n-(-1)} + t_0x_n + t_1x_{n-1} + t_2x_{n-2} = x_{n+1} + 2x_0 + 3x_{n-1}.$$

### Example 3.3: Writing down compact filter notation

Using the compact notation for a filter, we would write  $S = \{1/4, \underline{1/2}, 1/4\}$  for the filter given by formula (3.1)). For the filter

$$z_n = x_{n+1} + 2x_0 + 3x_{n-1}$$

from Example 3.2, we would write  $S = \{1, \underline{2}, 3\}$ .

#### 3.1.1 Convolution

Applying a filter to a vector  $\mathbf{x}$  is also called taking the *convolution* of the two vectors  $\mathbf{t}$  and  $\mathbf{x}$ . Convolution is usually defined without the assumption that the input vector is periodic, and without any assumption on the vector lengths (i.e. they may be sequences of infinite length). The case where both vectors  $\mathbf{t}$  and  $\mathbf{x}$  have a finite number of nonzero elements deserves extra attention. Assume that  $t_0, \dots, t_{M-1}$  and  $x_0, \dots, x_{N-1}$  are the only nonzero elements in  $\mathbf{t}$  and  $\mathbf{x}$  (i.e. we can view them as vectors in  $\mathbb{R}^M$  and  $\mathbb{R}^N$ , respectively). It is clear from the expression  $z_n = \sum t_k x_{n-k}$  that only  $z_0, \dots, z_{M+N-2}$  can be nonzero. This motivates the following definition.

**Definition 3.4.** *Convolution of vectors.*

By the *convolution* of two vectors  $\mathbf{t} \in \mathbb{R}^M$  and  $\mathbf{x} \in \mathbb{R}^N$  we mean the vector  $\mathbf{t} * \mathbf{x} \in \mathbb{R}^{M+N-1}$  defined by

$$(\mathbf{t} * \mathbf{x})_n = \sum_k t_k x_{n-k}, \quad (3.6)$$

where we only sum over  $k$  so that  $0 \leq k < M$ ,  $0 \leq n - k < N$ .

Note that convolution in the literature usually assumes infinite vectors. Python has the built-in function `convolve` for computing  $\mathbf{t} * \mathbf{x}$ . As we shall see in the exercises this function is highly optimized, and is therefore much used in practice. Since convolution is not exactly the same as our definition of a filter (since we assume that a vector is repeated periodically), it would be a good idea to express our definition of filters in terms of convolution. This can be achieved with the next proposition, which is formulated for the case with equally many filter coefficients with negative and positive indices. The result is thus directly applicable for symmetric filters, which is the type of filters we will mostly concentrate on. It is a simple exercise to generalize the result to other filters, however.

**Proposition 3.5.** *Using convolution to compute filters.*

Assume that  $S$  is a filter on the form

$$S = \{t_{-L}, \dots, t_0, \dots, t_L\}.$$

If  $\mathbf{x} \in \mathbb{R}^N$ , then  $S\mathbf{x}$  can be computed as follows:

- Form the vector  $\tilde{\mathbf{x}} = (x_{N-L}, \dots, x_{N-1}, x_0, \dots, x_{N-1}, x_0, \dots, x_{L-1}) \in \mathbb{R}^{N+2L}$ .
- Use the `convolve` function to compute  $\tilde{\mathbf{z}} = \mathbf{t} * \tilde{\mathbf{x}} \in \mathbb{R}^{M+N+2L-1}$ .
- We have that  $S\mathbf{x} = (\tilde{z}_{2L}, \dots, \tilde{z}_{M+N-2})$ .

We will consider an implementation of this result using the `convolve` function in the exercises.

*Proof.* When  $\mathbf{x} \in \mathbb{R}^N$ , the operation  $\mathbf{x} \rightarrow \mathbf{t} * \mathbf{x}$  can be represented by an  $(M + N - 1) \times N$  matrix. It is easy to see that this matrix has element  $(i + s, i)$  equal to  $t_s$ , for  $0 \leq i < M$ ,  $0 \leq s < N$ . In the left part of Figure 3.1 such a matrix is shown for  $M = 5$ . The (constant) nonzero diagonals are shown as diagonal lines.

Now, form the vector  $\tilde{\mathbf{x}} \in \mathbb{R}^{N+2L}$  as in the text of the theorem. Convolution  $(t_{-L}, \dots, t_L)$  with vectors in  $\mathbb{R}^{N+2L}$  can similarly be represented by an  $(M + N + 2L - 1) \times (N + 2L)$ -matrix. The rows from  $2L$  up to and including  $M + N - 2$  in this matrix (we have marked these with horizontal lines above) make up a new matrix  $\tilde{S}$ , shown in the right part of Figure 3.1 ( $\tilde{S}$  is an  $N \times (N + 2L)$  matrix).

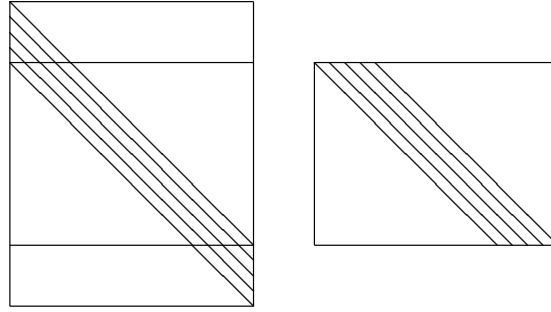


Figure 3.1: Matrix for the operation  $\mathbf{x} \rightarrow \mathbf{t} * \mathbf{x}$  (left), as well as this matrix with the first and last  $2L$  rows dropped (right).

We need to show that  $S\mathbf{x} = \tilde{S}\tilde{\mathbf{x}}$ . We have that  $\tilde{S}\tilde{\mathbf{x}}$  equals the matrix shown in the left part of Figure 3.2 multiplied with  $(x_{N-L}, \dots, x_{N-1}, x_0, \dots, x_{N-1}, x_0, \dots, x_{L-1})$  (we inserted extra vertical lines in the matrix where circulation occurs), which equals the matrix shown in the right part of Figure 3.2 multiplied with  $(x_0, \dots, x_{N-1})$ . We see that this is  $S\mathbf{x}$ , and the proof is complete.

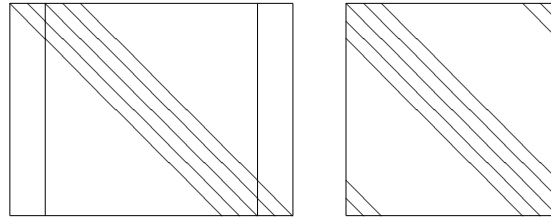


Figure 3.2: The matrix we multiply with  $(x_{N-L}, \dots, x_{N-1}, x_0, \dots, x_{N-1}, x_0, \dots, x_{L-1})$  (left), and the matrix we multiply with  $(x_0, \dots, x_{N-1})$  (right).

□

There is also a very nice connection between convolution and polynomials:

**Proposition 3.6.** *Convolution and polynomials.*

Assume that  $p(x) = a_N x^N + a_{N-1} x_{N-1} + \dots, a_1 x + a_0$  and  $q(x) = b_M x^M + b_{M-1} x_{M-1} + \dots, b_1 x + b_0$  are polynomials of degree  $N$  and  $M$  respectively. Then the coefficients of the polynomial  $pq$  can be obtained by computing  $\text{convolve}(\mathbf{a}, \mathbf{b})$ .

We can thus interpret a filter as a polynomial. In this setting, clearly the length  $l(S)$  of the filter can be interpreted as the degree of the polynomial. If  $\mathbf{t} \in \mathbb{R}^M$  and  $\mathbf{x} \in \mathbb{R}^N$ , then they can be associated with polynomials of degree  $M - 1$  and  $N - 1$ , respectively. Also, their convolution, which is in  $\mathbb{R}^{M+N-1}$ , can be associated with a polynomial of degree  $M + N - 2$ , which is the sum of the

degrees of the individual polynomials. Of course we can make the same addition of degrees when we multiply polynomials. Clearly the polynomial associated with  $\mathbf{t}$  is the frequency response, when we insert  $x = e^{-i\omega}$ . Also, applying two filters in succession is equivalent to applying the convolution of the filters, so that two filtering operations can be combined to one.

Since the number of nonzero filter coefficients is typically much less than  $N$  (the period of the input vector), the matrix  $S$  have many entries which are zero. Multiplication with such matrices requires less additions and multiplications than for other matrices: If  $S$  has  $k$  nonzero filter coefficients,  $S$  has  $Nk$  nonzero entries, so that  $kN$  multiplications and  $(k-1)N$  additions are needed to compute  $S\mathbf{x}$ . This is much less than the  $N^2$  multiplications and  $(N-1)N$  additions needed in the general case. Perhaps more important is that we need not form the entire matrix, we can perform the matrix multiplication directly in a loop. For large  $N$  we risk running into out of memory situations if we had to form the entire matrix.

### Exercise 3.4: Finding the filter coefficients and the matrix

Assume that the filter  $S$  is defined by the formula

$$z_n = \frac{1}{4}x_{n+1} + \frac{1}{4}x_n + \frac{1}{4}x_{n-1} + \frac{1}{4}x_{n-2}.$$

Write down the filter coefficients  $t_k$ , and the matrix for  $S$  when  $N = 8$ .

### Exercise 3.5: Finding the filter coefficients from the matrix

Given the circulant Toeplitz matrix

$$S = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix},$$

write down the filter coefficients  $t_k$ .

### Exercise 3.6: Convolution and polynomials

Compute the convolution of  $\{1, 2, 1\}$  with itself. interpret the result in terms of two polynomials.

### Exercise 3.7: Implementation of convolution

Implement code which computes  $\mathbf{t} * \mathbf{x}$  in the two ways described after Equation (3.3), i.e. as a double for loop, and as a simple for loop in `k`, with `n` vectorized. As your  $\mathbf{t}$ , take  $k$  randomly generated numbers. Compare execution times for these two methods and the `convolve` function, for different values of  $k$ . Present the result as a plot where  $k$  runs along the  $x$ -axis, and execution times run along the  $y$ -axis. Your result will depend on how Python performs vectorization.



**Exercise 3.8: Filters with a different number of coefficients with positive and negative indices**

Assume that  $S = \{t_{-E}, \dots, t_0, \dots, t_F\}$ . Formulate a generalization of Proposition 3.5 for such filters, i.e. to filters where there may be a different number of filter coefficients with positive and negative indices. You should only need to make some small changes to the proof of Proposition 3.5 to achieve this.

**Exercise 3.9: Implementing filtering with convolution**

Implement a function `filterS` which uses Proposition 3.5 and the `convolve` function  $S\mathbf{x}$  when  $S = \{t_{-L}, \dots, t_0, \dots, t_L\}$ . The function should take the vectors  $(t_{-L}, \dots, t_0, \dots, t_L)$  and  $\mathbf{x}$  as input.

**3.2 Formal definition of filters and the vector frequency response**

Let us now define digital filters formally, and establish their relationship to Toeplitz matrices. We have seen that a sound can be decomposed into different frequency components, and we would like to define filters as operations which adjust these frequency components in a predictable way. One such example is provided in Example 2.16, where we simply set some of the frequency components to 0. The natural starting point is to require for a filter that the output of a pure tone is a pure tone with the same frequency.

**Definition 3.7.** *Digital filters and vector frequency response.*

A linear transformation  $S : \mathbb{R}^N \mapsto \mathbb{R}^N$  is said to be a digital filter, or simply a filter, if, for any integer  $n$  in the range  $0 \leq n \leq N - 1$  there exists a value  $\lambda_{S,n}$  so that

$$S(\phi_n) = \lambda_{S,n}\phi_n, \quad (3.7)$$

i.e., the  $N$  Fourier vectors are the eigenvectors of  $S$ . The vector of (eigen)values  $\lambda_S = (\lambda_{S,n})_{n=0}^{N-1}$  is often referred to as the (*vector*) *frequency response* of  $S$ .

Since the Fourier basis vectors are orthogonal vectors,  $S$  is clearly orthogonally diagonalizable. Since also the Fourier basis vectors are the columns in  $(F_N)^H$ , we have that

$$S = F_N^H D F_N \quad (3.8)$$

whenever  $S$  is a digital filter, where  $D$  has the frequency response (i.e. the eigenvalues) on the diagonal<sup>1</sup>. We could also use  $\text{DFT}_N$  to diagonalize filters, but it is customary to use an orthogonal matrix (i.e.  $F_N$ ) when the matrix is

<sup>1</sup>Recall that the orthogonal diagonalization of  $S$  takes the form  $S = PDP^T$ , where  $P$  contains as columns an orthonormal set of eigenvectors, and  $D$  is diagonal with the eigenvectors listed on the diagonal (see Section 7.1 in [25]).

orthogonally diagonalizable. In particular, if  $S_1$  and  $S_2$  are digital filters, we can write  $S_1 = F_N^H D_1 F_N$  and  $S_2 = F_N^H D_2 F_N$ , so that

$$S_1 S_2 = F_N^H D_1 F_N F_N^H D_2 F_N = F_N^H D_1 D_2 F_N.$$

Since  $D_1 D_2 = D_2 D_1$  for any diagonal matrices, we get the following corollary:

**Corollary 3.8.** *The product of two filters is a filter.*

The product of two digital filters is again a digital filter. Moreover, all digital filters commute, i.e. if  $S_1$  and  $S_2$  are digital filters,  $S_1 S_2 = S_2 S_1$ .

Clearly also  $S_1 + S_2$  is a filter when  $S_1$  and  $S_2$  are. The set of all filters is thus a vector space, which also is closed under multiplication. Such a space is called an *algebra*. Since all filters commute, this algebra is also called a *commutative algebra*.

### 3.2.1 Time delay

*Time delay* with  $d$  elements is defined by  $E_d(\mathbf{x} = \mathbf{z}$ , with  $z_k = x_{k-d}$ . From this it follows that the vector with components  $e^{2\pi i k n / N}$  is sent to the vector with components  $e^{2\pi i (k-d) n / N} = e^{-2\pi i d n / N} e^{2\pi i k n / N}$ . This means that  $\phi_n$  is an eigenvector, so that time delay with  $d$  elements is a digital filter. Since  $e^{-2\pi i d n / N}$  is the corresponding eigenvalue for  $\phi_n$ , this also gives us the frequency response. Since all  $|\lambda_{S,n}| = 1$ , time-delay does not change the amplitude of frequencies in sounds.

The next result states three equivalent characterizations of a digital filter. The first one is simply the definition in terms of having the Fourier basis as eigenvectors. The second is that the matrix is circulant Toeplitz, i.e. that the operations we started this chapter with actually are filters. The third characterization is in terms of a new concept which we now define.

**Definition 3.9.** *Time-invariance.*

Assume that  $S$  is a linear transformation from  $\mathbb{R}^N$  to  $\mathbb{R}^N$ . Let  $\mathbf{x}$  be input to  $S$ , and  $\mathbf{y} = S\mathbf{x}$  the corresponding output. Let also  $\mathbf{z}$ ,  $\mathbf{w}$  be delays of  $\mathbf{x}$ ,  $\mathbf{y}$  with  $d$  elements (i.e.  $\mathbf{z} = E_d \mathbf{x}$ ,  $\mathbf{w} = E_d \mathbf{y}$ ).  $S$  is said to be *time-invariant* if, for any  $d$  and  $\mathbf{x}$ ,  $S\mathbf{z} = \mathbf{w}$  (i.e.  $S$  sends the delayed input vector to the delayed output vector, equivalently  $SE_d = E_d S$ ).

Clearly time delay is time-invariant, since  $E_{d_1} E_{d_2} = E_{d_2} E_{d_1} = E_{d_1+d_2}$  for any  $d_1$  and  $d_2$ .

**Theorem 3.10.** *Characterizations of digital filters.*

The following are equivalent characterizations of a digital filter:

- $S = (F_N)^H D F_N$  for a diagonal matrix  $D$ , i.e. the Fourier basis is a basis of eigenvectors for  $S$ .
- $S$  is a circulant Toeplitz matrix.

- $S$  is linear and time-invariant.

*Proof.* If  $S$  is a filter, then  $SE_d = E_dS$  for all  $d$  since all filters commute, so that  $S$  is time-invariant. This proves 1.  $\rightarrow$  3..

Assume that  $S$  is time-invariant. Note that  $E_d\mathbf{e}_0 = \mathbf{e}_d$ , and since  $SE_d\mathbf{e}_0 = E_dS\mathbf{e}_0$  we have that  $S\mathbf{e}_d = E_d\mathbf{s}$ , where  $\mathbf{s}$  is the first column of  $S$ . This also says that column  $d$  of  $S$  can be obtained by delaying the first column of  $S$  with  $d$  elements. But then  $S$  is a circulant Toeplitz matrix. This proves 3.  $\rightarrow$  2..

Finally, any circulant Toeplitz matrix can be written on the form  $\sum_{d=0}^{N-1} s_d E_d$  (by splitting the matrix into a sum of its diagonals). Since all  $E_d$  are filters, it is clear that any circulant Toeplitz matrix is a filter. This proves 2.  $\rightarrow$  1..  $\square$

Due to this result, filters are also called *LTI filters*, LTI standing for Linear, Time-Invariant. Also, operations defined by (3.3) are digital filters, when restricted to vectors with period  $N$ . The following result enables us to compute the eigenvalues/frequency response easily through the DFT, so that we do not need to form the characteristic polynomial and find its roots:

**Theorem 3.11.** *Connection between frequency response and the matrix.*

Any digital filter is uniquely characterized by the values in the first column of its matrix. Moreover, if  $\mathbf{s}$  is the first column in  $S$ , the frequency response of  $S$  is given by

$$\boldsymbol{\lambda}_S = \text{DFT}_N \mathbf{s}. \quad (3.9)$$

Conversely, if we know the frequency response  $\boldsymbol{\lambda}_S$ , the first column  $\mathbf{s}$  of  $S$  is given by

$$\mathbf{s} = \text{IDFT}_N \boldsymbol{\lambda}_S. \quad (3.10)$$

*Proof.* If we replace  $S$  by  $(F_N)^H D F_N$  we find that

$$\begin{aligned} \text{DFT}_N \mathbf{s} &= \sqrt{N} F_N \mathbf{s} = \sqrt{N} F_N S \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \sqrt{N} F_N F_N^H D F_N \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\ &= \sqrt{N} D F_N \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = D \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \boldsymbol{\lambda}_S, \end{aligned}$$

where we have used that the first column in  $F_N$  has all entries equal to  $1/\sqrt{N}$ , and that the diagonal matrix  $D$  has all the eigenvalues of  $S$  on its diagonal, so that the last expression is the vector of eigenvalues  $\boldsymbol{\lambda}_S$ . This proves (3.9). Equation (3.10) follows directly by applying the inverse DFT to (3.9).  $\square$

The first column  $\mathbf{s}$ , which thus characterizes the filter, is also called the *impulse response*. This name stems from the fact that we can write  $\mathbf{s} = S\mathbf{e}_0$ , i.e. the vector  $\mathbf{s}$  is the output (often called response) to the vector  $\mathbf{e}_0$  (often called an impulse). Equation (3.9) states that the frequency response can be written as

$$\lambda_{S,n} = \sum_{k=0}^{N-1} s_k e^{-2\pi ink/N}, \quad \text{for } n = 0, 1, \dots, N-1, \quad (3.11)$$

where  $s_k$  are the components of  $\mathbf{s}$ .

The identity matrix is a digital filter since  $I = (F_N)^H I F_N$ . Since  $\mathbf{e}_0$  is the first column, it has impulse response  $\mathbf{s} = \mathbf{e}_0$ . Its frequency response has 1 in all components and therefore preserves all frequencies, as expected.

In signal processing, the frequency content of a vector (i.e., its DFT) is also referred to as its spectrum. This may be somewhat confusing from a linear algebra perspective, because in this context the term spectrum is used to denote the eigenvalues of a matrix. But because of Theorem 3.11 this is not so confusing after all if we interpret the spectrum of a vector (in signal processing terms) as the spectrum of the corresponding digital filter (in linear algebra terms).

### Example 3.10: Frequency response of a simple filter

When only few of the coefficients  $s_k$  are nonzero, it is possible to obtain nice expressions for the frequency response. To see this, let us compute the frequency response of the filter defined from formula (3.1). We saw that the first column of the corresponding Toeplitz matrix satisfied  $s_0 = 1/2$ , and  $s_{N-1} = s_1 = 1/4$ . The frequency response is thus

$$\begin{aligned} \lambda_{S,n} &= \frac{1}{2}e^0 + \frac{1}{4}e^{-2\pi in/N} + \frac{1}{4}e^{-2\pi in(N-1)/N} \\ &= \frac{1}{2}e^0 + \frac{1}{4}e^{-2\pi in/N} + \frac{1}{4}e^{2\pi in/N} = \frac{1}{2} + \frac{1}{2}\cos(2\pi n/N). \end{aligned}$$

### Example 3.11: Matrix form

We have seen that the DFT can be used to spare us the tedious calculation of eigenvectors and eigenvalues we are used to, at least for circulant Toeplitz matrices. Let us compare the two approaches for a simple matrix.

$$S = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}.$$

It is straightforward to compute the eigenvalues and eigenvectors of this matrix the way you learned in your first course in linear algebra. However, this matrix is also a circulant Toeplitz matrix, so that we can use the results in this section to compute the eigenvalues and eigenvectors. Since here  $N = 2$ , we have that  $e^{2\pi ink/N} = e^{\pi ink} = (-1)^{nk}$ . This means that the Fourier basis vectors are

$(1, 1)/\sqrt{2}$  and  $(1, -1)/\sqrt{2}$ , which also are the eigenvectors of  $S$ . The eigenvalues are the frequency response of  $S$ , which can be obtained as

$$\sqrt{N}F_N \mathbf{s} = \sqrt{2} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$$

The eigenvalues are thus 3 and 5. You could have obtained the same result with your computer. Note that the computer may not return the eigenvectors exactly as the Fourier basis vectors, since the eigenvectors are not unique (the multiple of an eigenvector is also an eigenvector). The computer may for instance switch the signs of the eigenvectors. We have no control over what the computer chooses to do, since some underlying numerical algorithm for computing eigenvectors is used, which we can't influence.

### Example 3.12: Computing the output of a filter

Certain vectors are easy to express in terms of the Fourier basis. This enables us to compute the output of such vectors from a digital filter easily.

Let us consider the filter  $S$  defined by  $z_n = \frac{1}{6}(x_{n+2} + 4x_{n+1} + 6x_n + 4x_{n-1} + x_{n-2})$ , and see how we can compute  $S\mathbf{x}$  when

$$\mathbf{x} = (\cos(2\pi 5 \cdot 0/N), \cos(2\pi 5 \cdot 1/N), \dots, \cos(2\pi 5 \cdot (N-1)/N)),$$

where  $N$  is the length of the vector. We note first that

$$\begin{aligned} \sqrt{N}\phi_5 &= (e^{2\pi i 5 \cdot 0/N}, e^{2\pi i 5 \cdot 1/N}, \dots, e^{2\pi i 5 \cdot (N-1)/N}) \\ \sqrt{N}\phi_{N-5} &= (e^{-2\pi i 5 \cdot 0/N}, e^{-2\pi i 5 \cdot 1/N}, \dots, e^{-2\pi i 5 \cdot (N-1)/N}), \end{aligned}$$

Since  $e^{2\pi i 5k/N} + e^{-2\pi i 5k/N} = 2\cos(2\pi 5k/N)$ , we get by adding the two vectors that  $\mathbf{x} = \frac{1}{2}\sqrt{N}(\phi_5 + \phi_{N-5})$ . Since the  $\phi_n$  are eigenvectors, we have expressed  $\mathbf{x}$  as a sum of eigenvectors. The corresponding eigenvalues are given by the vector frequency response, so let us compute this. If  $N = 8$ , computing  $S\mathbf{x}$  means to multiply with the  $8 \times 8$  circulant Toeplitz matrix

$$\frac{1}{6} \begin{pmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 1 & 4 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 1 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 1 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 4 & 1 & 0 & 0 & 0 & 1 & 4 & 6 \end{pmatrix}$$

We now see that

$$\begin{aligned}
\lambda_{S,n} &= \frac{1}{6}(6 + 4e^{-2\pi in/N} + e^{-2\pi i2n/N} + e^{-2\pi i(N-2)n/N} + 4e^{-2\pi i(N-1)n/N}) \\
&= \frac{1}{6}(6 + 4e^{2\pi in/N} + 4e^{-2\pi in/N} + e^{2\pi i2n/N} + e^{-2\pi i2n/N}) \\
&= 1 + \frac{4}{3} \cos(2\pi n/N) + \frac{1}{3} \cos(4\pi n/N).
\end{aligned}$$

The two values of this we need are

$$\begin{aligned}
\lambda_{S,5} &= 1 + \frac{4}{3} \cos(2\pi 5/N) + \frac{1}{3} \cos(4\pi 5/N) \\
\lambda_{S,N-5} &= 1 + \frac{4}{3} \cos(2\pi(N-5)/N) + \frac{1}{3} \cos(4\pi(N-5)/N) \\
&= 1 + \frac{4}{3} \cos(2\pi 5/N) + \frac{1}{3} \cos(4\pi 5/N).
\end{aligned}$$

Since these are equal,  $\mathbf{x}$  is a sum of eigenvectors with equal eigenvalues. This means that  $\mathbf{x}$  itself also is an eigenvector, with the same eigenvalue, so that

$$S\mathbf{x} = \left(1 + \frac{4}{3} \cos(2\pi 5/N) + \frac{1}{3} \cos(4\pi 5/N)\right) \mathbf{x}.$$

### 3.2.2 Using digital filters to approximate filters

The formal definition of digital filters resembles that of filters from Chapter 1, the difference being that the Fourier basis now is discrete. Let us try to connect the two. In doing so, let us differ them by calling the ones from Chapter 1 simply filters (without *digital* in front). We have the following result.

**Theorem 3.12.** *Connection with the frequency response.*

Let  $s$  be a filter with frequency response  $\lambda_s(f)$ , and assume that  $f \in V_{M,T}$  (so that also  $s(f) \in V_{M,T}$ ). Let

$$\begin{aligned}
\mathbf{x} &= (f(0 \cdot T/N), f(1 \cdot T/N), \dots, f((N-1)T/N)) \\
\mathbf{z} &= (s(f)(0 \cdot T/N), s(f)(1 \cdot T/N), \dots, s(f)((N-1)T/N))
\end{aligned}$$

be vectors of  $N = 2M + 1$  uniform samples from  $f$  and  $s(f)$ . Then the operation  $S : \mathbf{x} \rightarrow \mathbf{z}$  (i.e. the operation which sends the samples of the input to the samples of the output) is well-defined on  $\mathbb{R}^N$ , and is an  $N \times N$ -digital filter with vector frequency response  $\lambda_{S,n} = \lambda_s(n/T)$ .

*Proof.* With  $N = 2M + 1$  we know that  $f \in V_{M,T}$  is uniquely determined from  $\mathbf{x}$ . This means that  $s(f)$  also is uniquely determined from  $\mathbf{x}$ , so that  $\mathbf{z}$  also is uniquely determined from  $\mathbf{x}$ . The operation  $S : \mathbf{x} \rightarrow \mathbf{z}$  is therefore well-defined on  $\mathbb{R}^N$ .

Clearly also  $s(e^{2\pi int/T}) = \lambda_s(n/T)e^{2\pi int/T}$ . Since the samples of  $e^{2\pi int/T}$  is the vector  $e^{2\pi ikn/N}$ , and the samples of  $\lambda_s(n/T)e^{2\pi int/T}$  is  $\lambda_s(n/T)e^{2\pi ikn/N}$ , the vector  $e^{2\pi ikn/N}$  is an eigenvector of  $S$  with eigenvalue  $\lambda_s(n/T)$ . Clearly then  $S$  is a digital filter with frequency response  $\lambda_{S,n} = \lambda_s(n/T)$ .  $\square$

It is interesting that the vector frequency response above is obtained by sampling the frequency response. In this way we also see that it is easy to realize any digital filter as the restriction of a filter: an filter  $s$  where the frequency response has the values  $\lambda_{S,n}$  at the points  $n/T$  will do. In the theorem it is essential that  $f \in V_{M,T}$ . There are many functions with the same samples, but where the samples of the output from the filter are different. When we restrict to  $V_{M,T}$ , however, the output samples are always determined from the input samples.

Theorem 3.12 explains how digital filters can occur in practice. In the real world, a signal is modeled as a continuous function  $f(t)$ , and an operation on signals as a filter  $s$ . We can't compute the entire output  $s(f)$  of the filter, but it is possible to apply the digital filter from Theorem 3.12 to the samples  $\mathbf{x}$  of  $f$ . In general  $f(t)$  may not lie in  $V_{M,T}$ , but we can denote by  $\tilde{f}$  the unique function in  $V_{M,T}$  with the same samples as  $f$  (as in Section 2.2). By definition,  $S\mathbf{x}$  are the samples of  $s(\tilde{f}) \in V_{M,T}$ .  $s(\tilde{f})$  can finally be found from these samples by using the procedure from Figure 2.4 for finding  $s(\tilde{f})$ . This procedure for finding  $s(\tilde{f})$  is illustrated in Figure 3.3.

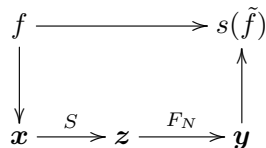


Figure 3.3: The connections between filters and digital filters, sampling and interpolation, provided by Theorem 3.12. The left vertical arrow represents sampling, the right vertical arrow represents interpolation.

Clearly,  $s(\tilde{f})$  is an approximation to  $s(f)$ , since  $\tilde{f}$  is an approximation to  $f$ , and since  $s$  is continuous. Let us summarize this as follows:

**Idea 3.13.** *Approximating a filter.*

A filter  $s$  can be approximated through sampling, a digital filter, the DFT, and interpolation, as illustrated in Figure 3.3.  $S$  is the digital filter with frequency response  $\lambda_{S,n} = \lambda_s(n/T)$ . When  $f \in V_{M,T}$ , this approximation equals  $s(f)$ . When we increase the number of sample points/the size of the filter, the approximation becomes better. If there is a bound on the highest frequency in  $f$ , there exists an  $N$  so that when sampling of that size, the approximation equals  $s(f)$ .

Let us comment on why the last statements here are true. That the approximation equals  $s(f)$  when  $f \in V_{M,T}$  is obvious, since both  $f$  and  $s(f) \in V_{M,T}$

are determined from their samples then. If there is a bound on the highest frequency in  $f$ , then  $f$  lies in  $V_{M,T}$  for large enough  $M$ , so that we recover  $s(f)$  as our approximation using  $N = 2M + 1$ . Finally, what happens when there is no bound on the highest frequency? We know that  $s(f_N) = (s(f))_N$ . Since  $f_N$  is a good approximation to  $f$ , the samples  $\mathbf{x}$  of  $f$  are close to the samples of  $f_N$ . By continuity of the digital filter,  $\mathbf{z} = S\mathbf{x}$  will also be close to the samples of  $(s(f))_N = s(f_N)$ , so that (also by continuity) interpolating with  $\mathbf{z}$  gives a good approximation to  $(s(f))_N$ , which is again a good approximation to  $s(f)$ . From this it follows that the digital filter is a better approximation when  $N$  is high.

### Exercise 3.13: Time reversal is not a filter

In Example 1.2 we looked at time reversal as an operation on digital sound. In  $\mathbb{R}^N$  this can be defined as the linear mapping which sends the vector  $\mathbf{e}_k$  to  $\mathbf{e}_{N-1-k}$  for all  $0 \leq k \leq N - 1$ .

- Write down the matrix for the time reversal linear mapping, and explain from this why time reversal is not a digital filter.
- Prove directly that time reversal is not a time-invariant operation.

### Exercise 3.14: When is a filter symmetric?

Let  $S$  be a digital filter. Show that  $S$  is symmetric if and only if the frequency response satisfies  $\lambda_{S,n} = \lambda_{S,N-n}$  for all  $n$ .

### Exercise 3.15: Eigenvectors and eigenvalues

Consider the matrix

$$S = \begin{pmatrix} 4 & 1 & 3 & 1 \\ 1 & 4 & 1 & 3 \\ 3 & 1 & 4 & 1 \\ 1 & 3 & 1 & 4 \end{pmatrix}.$$

- Compute the eigenvalues and eigenvectors of  $S$  using the results of this section. You should only need to perform one DFT in order to achieve this.
- Verify the result from a) by computing the eigenvectors and eigenvalues the way you taught in your first course in linear algebra. This should be a much more tedious task.
- Use a computer to compute the eigenvectors and eigenvalues of  $S$  also. For some reason some of the eigenvectors seem to be different from the Fourier basis vectors, which you would expect from the theory in this section. Try to find an explanation for this.



**Exercise 3.16: Composing filters**

Assume that  $S_1$  and  $S_2$  are two circulant Toeplitz matrices.

- a) How can you express the eigenvalues of  $S_1 + S_2$  in terms of the eigenvalues of  $S_1$  and  $S_2$ ?
- b) How can you express the eigenvalues of  $S_1 S_2$  in terms of the eigenvalues of  $S_1$  and  $S_2$ ?
- c) If  $A$  and  $B$  are general matrices, can you find a formula which expresses the eigenvalues of  $A + B$  and  $AB$  in terms of those of  $A$  and  $B$ ? If not, can you find a counterexample to what you found in a) and b)?

**Exercise 3.17: Keeping every second component**

Consider the linear mapping  $S$  which keeps every second component in  $\mathbb{R}^N$ , i.e.  $S(e_{2k}) = e_{2k}$ , and  $S(e_{2k-1}) = \mathbf{0}$ . Is  $S$  a digital filter?

**3.3 The continuous frequency response and properties**

If we make the substitution  $\omega = 2\pi n/N$  in the formula for  $\lambda_{S,n}$ , we may interpret the frequency response as the values on a continuous function on  $[0, 2\pi)$ .

**Theorem 3.14.** *Connection between vector- and continuous frequency response.*

The function  $\lambda_S(\omega)$  defined on  $[0, 2\pi)$  by

$$\lambda_S(\omega) = \sum_k t_k e^{-ik\omega}, \quad (3.12)$$

where  $t_k$  are the filter coefficients of  $S$ , satisfies

$$\lambda_{S,n} = \lambda_S(2\pi n/N) \text{ for } n = 0, 1, \dots, N-1$$

for any  $N$ . In other words, regardless of  $N$ , the vector frequency response lies on the curve  $\lambda_S$ .  $\lambda_S(\omega)$  is called the *continuous frequency response* of  $S$ , and  $\omega$  is called *angular frequency*.

The difference between the vector- and continuous frequency response lies in that one uses the filter coefficients  $t_k$ , while the other uses the impulse response  $s_k$ . These contain the same values, but they are ordered differently. The result shows that, at the points  $2\pi n/N$ , they are equal.

*Proof.* For any  $N$  we have that

$$\begin{aligned}
\lambda_{S,n} &= \sum_{k=0}^{N-1} s_k e^{-2\pi i n k / N} = \sum_{0 \leq k < N/2} s_k e^{-2\pi i n k / N} + \sum_{N/2 \leq k \leq N-1} s_k e^{-2\pi i n k / N} \\
&= \sum_{0 \leq k < N/2} t_k e^{-2\pi i n k / N} + \sum_{N/2 \leq k \leq N-1} t_{k-N} e^{-2\pi i n k / N} \\
&= \sum_{0 \leq k < N/2} t_k e^{-2\pi i n k / N} + \sum_{-N/2 \leq k \leq -1} t_k e^{-2\pi i n (k+N) / N} \\
&= \sum_{0 \leq k < N/2} t_k e^{-2\pi i n k / N} + \sum_{-N/2 \leq k \leq -1} t_k e^{-2\pi i n k / N} \\
&= \sum_{-N/2 \leq k < N/2} t_k e^{-2\pi i n k / N} = \lambda_S(2\pi n / N).
\end{aligned}$$

where we have used Equation (3.4). □

If  $\mathbf{t}$  is the set of filter coefficients of  $S$ , we can combine Theorem 3.14 with Equation (3.9), and use the fact that time delay does not affect the absolute value of the DFT (Theorem 2.7), in order to plot the frequency response of  $S$  as follows:

```

omega = 2*pi*arange(0,N)/float(N)
s = concatenate([t, zeros(N - len(t))])
plot(omega, abs(fft.fft(s)))

```

With this procedure we avoid computing the frequency response by hand. We will have use for this several places later.

Note that  $\sum_{k=0}^{N-1} s_k e^{-\pi i \omega k}$  typically will not converge when  $N \rightarrow \infty$  (although it gives the right values at all points  $\omega = 2\pi n / N$  for all  $N$ )! The filter coefficients avoid this convergence problem, however, since we assume that only  $t_k$  with  $|k|$  small are nonzero. In other words, filter coefficients are used in the definition of the continuous frequency response so that we can find a continuous curve where we can find the vector frequency response values for all  $N$ .

The frequency response contains the important characteristics of a filter, since it says how it behaves for the different frequencies. When analyzing a filter, we therefore often plot the frequency response. Often we plot only the absolute value (or the magnitude) of the frequency response, since this is what explains how each frequency is amplified or attenuated. Since  $\lambda_S$  is clearly periodic with period  $2\pi$ , we may restrict angular frequency to the interval  $[0, 2\pi)$ . The conclusion in Observation 2.10 was that the low frequencies in a vector correspond to DFT indices close to 0 and  $N - 1$ , and high frequencies correspond to DFT indices close to  $N/2$ . This observation is easily translated to a statement about angular frequencies:

**Observation 3.15.** *Plotting the frequency response.*

When plotting the frequency response on  $[0, 2\pi)$ , angular frequencies near 0 and  $2\pi$  correspond to low frequencies, angular frequencies near  $\pi$  correspond to high frequencies

$\lambda_S$  may also be viewed as a function defined on the interval  $[-\pi, \pi)$ . Plotting on  $[-\pi, \pi]$  is often done in practice, since it makes clearer what corresponds to lower frequencies, and what corresponds to higher frequencies:

**Observation 3.16.** *Higher and lower frequencies.*

When plotting the frequency response on  $[-\pi, \pi)$ , angular frequencies near 0 correspond to low frequencies, angular frequencies near  $\pm\pi$  correspond to high frequencies.

The following holds:

**Theorem 3.17.** *Connection between analog and digital filters.*

Assume that  $s$  is an analog filter, and that we sample a periodic function at rate  $f_s$  over one period, and denote the corresponding digital filter by  $S$ . The analog and digital frequency responses are related by  $\lambda_s(f) = \lambda_S(2\pi f f_s)$ .

To see this, note first that  $S$  has frequency response  $\lambda_{S,n} = \lambda_s(n/T) = \lambda_s(f)$ , where  $f = n/T$ . We then rewrite  $\lambda_{S,n} = \lambda_S(2\pi n/N) = \lambda_S(2\pi f T/N) = \lambda_S(2\pi f f_s)$ .

Since the frequency response is essentially a DFT, it inherits several properties from Theorem 2.7. We will mostly use the continuous frequency response to express these properties.

**Theorem 3.18.** *Properties of the frequency response.*

We have that

- The continuous frequency response satisfies  $\lambda_S(-\omega) = \overline{\lambda_S(\omega)}$ .
- If  $S$  is a digital filter,  $S^T$  is also a digital filter. Moreover, if the frequency response of  $S$  is  $\lambda_S(\omega)$ , then the frequency response of  $S^T$  is  $\overline{\lambda_S(\omega)}$ .
- If  $S$  is symmetric,  $\lambda_S$  is real. Also, if  $S$  is antisymmetric (the element on the opposite side of the diagonal is the same, but with opposite sign),  $\lambda_S$  is purely imaginary.
- A digital filter  $S$  is invertible if and only if  $\lambda_{S,n} \neq 0$  for all  $n$ . In that case  $S^{-1}$  is also a digital filter, and  $\lambda_{S^{-1},n} = 1/\lambda_{S,n}$ .
- If  $S_1$  and  $S_2$  are digital filters, then  $S_1 S_2$  also is a digital filter, and  $\lambda_{S_1 S_2}(\omega) = \lambda_{S_1}(\omega) \lambda_{S_2}(\omega)$ .

*Proof.* Property 1. and 3. follow directly from Theorem 2.7. Transposing a matrix corresponds to reversing the first column of the matrix and thus also the filter coefficients. Due to this Property 2. also follows from Theorem 2.7. If  $S = (F_N)^H D F_N$ , and all  $\lambda_{S,n} \neq 0$ , we have that  $S^{-1} = (F_N)^H D^{-1} F_N$ , where

$D^{-1}$  is a diagonal matrix with the values  $1/\lambda_{S,n}$  on the diagonal. Clearly then  $S^{-1}$  is also a digital filter, and its frequency response is  $\lambda_{S^{-1},n} = 1/\lambda_{S,n}$ , which proves 4. The last property follows in the same way as we showed that filters commute:

$$S_1 S_2 = (F_N)^H D_1 F_N (F_N)^H D_2 F_N = (F_N)^H D_1 D_2 F_N.$$

The frequency response of  $S_1 S_2$  is thus obtained by multiplying the frequency responses of  $S_1$  and  $S_2$ .  $\square$

In particular the frequency response may not be real, although this was the case in the first example of this section. Theorem 3.18 applies also for the vector frequency response. Since the vector frequency response are the eigenvalues of the filter, the last property above says that, for filters, multiplication of matrices corresponds to multiplication of eigenvalues. Clearly this is an important property which is shared with all other matrices which have the same eigenvectors.

### Example 3.18: Plotting a simple frequency response

In Example 3.10 we computed the vector frequency response of the filter defined in formula (3.1). The filter coefficients are here  $t_{-1} = 1/4$ ,  $t_0 = 1/2$ , and  $t_1 = 1/4$ . The continuous frequency response is thus

$$\lambda_S(\omega) = \frac{1}{4}e^{i\omega} + \frac{1}{2} + \frac{1}{4}e^{-i\omega} = \frac{1}{2} + \frac{1}{2}\cos\omega.$$

Clearly this matches the computation from Example 3.10. Figure 3.4 shows plots of this frequency response, plotted on the intervals  $[0, 2\pi]$  and  $[-\pi, \pi]$ .

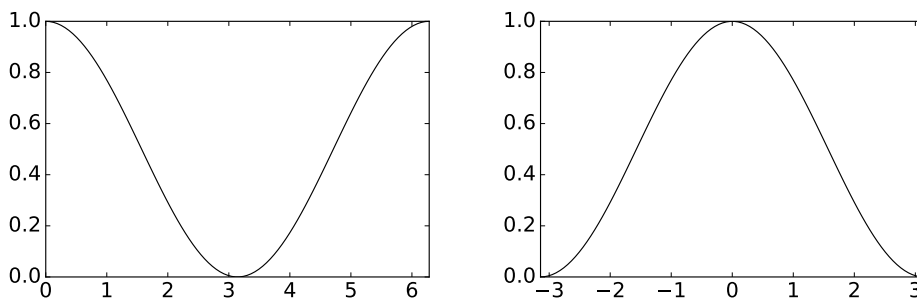


Figure 3.4:  $|\lambda_S(\omega)|$  of the moving average filter of Formula (3.1), plotted over  $[0, 2\pi]$  and  $[-\pi, \pi]$ .

Both the continuous frequency response and the vector frequency response for  $N = 51$  are shown. The right part shows clearly how the high frequencies are softened by the filter.

**Example 3.19: Computing a composite filter**

Assume that the filters  $S_1$  and  $S_2$  have the frequency responses  $\lambda_{S_1}(\omega) = \cos(2\omega)$ ,  $\lambda_{S_2}(\omega) = 1 + 3\cos\omega$ . Let us see how we can use Theorem 3.18 to compute the filter coefficients and the matrix of the filter  $S = S_1S_2$ . We first notice that, since both frequency responses are real, all  $S_1$ ,  $S_2$ , and  $S = S_1S_2$  are symmetric. We rewrite the frequency responses as

$$\begin{aligned}\lambda_{S_1}(\omega) &= \frac{1}{2}(e^{2i\omega} + e^{-2i\omega}) = \frac{1}{2}e^{2i\omega} + \frac{1}{2}e^{-2i\omega} \\ \lambda_{S_2}(\omega) &= 1 + \frac{3}{2}(e^{i\omega} + e^{-i\omega}) = \frac{3}{2}e^{i\omega} + 1 + \frac{3}{2}e^{-i\omega}.\end{aligned}$$

We now get that

$$\begin{aligned}\lambda_{S_1S_2}(\omega) &= \lambda_{S_1}(\omega)\lambda_{S_2}(\omega) = \left(\frac{1}{2}e^{2i\omega} + \frac{1}{2}e^{-2i\omega}\right) \left(\frac{3}{2}e^{i\omega} + 1 + \frac{3}{2}e^{-i\omega}\right) \\ &= \frac{3}{4}e^{3i\omega} + \frac{1}{2}e^{2i\omega} + \frac{3}{4}e^{i\omega} + \frac{3}{4}e^{-i\omega} + \frac{1}{2}e^{-2i\omega} + \frac{3}{4}e^{-3i\omega}\end{aligned}$$

From this expression we see that the filter coefficients of  $S$  are  $t_{\pm 1} = 3/4$ ,  $t_{\pm 2} = 1/2$ ,  $t_{\pm 3} = 3/4$ . All other filter coefficients are 0. Using Theorem 3.2, we get that  $s_1 = 3/4$ ,  $s_2 = 1/2$ , and  $s_3 = 3/4$ , while  $s_{N-1} = 3/4$ ,  $s_{N-2} = 1/2$ , and  $s_{N-3} = 3/4$  (all other  $s_k$  are 0). This gives us the matrix representation of  $S$ .

**3.3.1 Windowing operations**

In this section we will take a look at a very important, and perhaps surprising, application of the continuous frequency response. Let us return to the computations from Example 2.16. There we saw that, when we restricted to a block of the signal, this affected the frequency representation. If we substitute with the angular frequencies  $\omega = 2\pi n/N$  and  $\omega_0 = 2\pi n_0/M$  in Equation (2.12), we get

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} e^{ik\omega_0} e^{-ik\omega} = \frac{1}{N} \sum_{k=0}^{N-1} e^{-ik(\omega - \omega_0)}$$

(here  $y_n$  were the DFT components of the sound after we had restricted to a block). This expression states that, when we restrict to a block of length  $N$  in the signal by discarding the other samples, a pure tone of angular frequency  $\omega_0$  suddenly gets a frequency contribution at angular frequency  $\omega$  also, and the contribution is given by this formula. The expression is seen to be the same as the frequency response of the filter  $\frac{1}{N}\{1, 1, \dots, 1\}$  (where 1 is repeated  $N$  times), evaluated at  $\omega - \omega_0$ . This filter is nothing but a (delayed) moving average filter. The frequency response of a moving average filter thus governs how the different frequencies pollute when we limit ourselves to a block of the signal. Since this

frequency response has its peak at 0, angular frequencies  $\omega$  close to  $\omega_0$  have biggest values, so that the pollution is mostly from frequencies close to  $\omega_0$ . But unfortunately, other frequencies also pollute.

One can also ask the question if there are better ways to restrict to a block of size  $N$  of the signal. We formulate the following idea.

**Idea 3.19.** *Windows.*

Let  $\mathbf{x} = (x_0, \dots, x_M)$  be a sound of length  $M$ . We would like to find values  $\mathbf{w} = \{w_0, \dots, w_{N-1}\}$  so that the new sound  $(w_0x_0, \dots, w_{N-1}x_{N-1})$  of length  $N < M$  has a frequency representation similar to that of  $\mathbf{x}$ .  $\mathbf{w}$  is called a *window* of length  $N$ , and the new sound is called the *windowed signal*.

Above we encountered the window  $\mathbf{w} = \{\underline{1}, 1, \dots, 1\}$ . This is called the rectangular window. To see how we can find a good window, note first that the DFT values in the windowed signal of length  $N$  is

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} w_k e^{ik\omega_0} e^{-ik\omega} = \frac{1}{N} \sum_{k=0}^{N-1} w_k e^{-ik(\omega - \omega_0)}.$$

This is the frequency response of  $\frac{1}{N}\mathbf{w}$ . In order to limit the pollution from other frequencies, we thus need to construct a window with a frequency response with smaller values than that of the rectangular window away from 0. Let us summarize our findings as follows:

**Observation 3.20.** *Constructing a window.*

Assume that we would like to construct a window of length  $N$ . It is desirable that the frequency response of the window has small values away from zero.

We will not go into techniques for how such frequency responses can be constructed, but only consider one example different from the rectangular window. We define the Hamming window by

$$w_n = 2(0.54 - 0.46 \cos(2\pi n/(N - 1))). \quad (3.13)$$

The frequency responses of the rectangular window and the Hamming window are compared in Figure 3.5 for  $N = 32$ .

We see that the Hamming window has much smaller values away from 0, so that it is better suited as a window. However, the width of the “main lobe” (i.e. the main structure at the center), seems to be bigger. The window coefficients themselves are shown in Figure 3.6. It is seen that the frequency response of the Hamming window attenuates more and more as we get close to the boundaries.

Many other windows are used in the literature. The concrete window from Exercise 3.29 is for instance used in the MP3 standard. It is applied to the sound, and after this an FFT is applied to the windowed sound in order to make a frequency analysis of that part of the sound. The effect of the window is that there is smaller loss in the frequency representation of the sound when we restrict to a block of sound samples. This is a very important part of the psychoacoustic model used in the MP3 encoder, since it has to make compression decisions based on the frequency information in the sound.

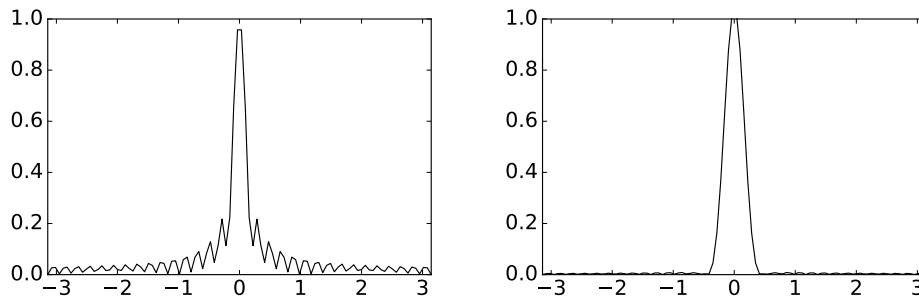


Figure 3.5: The frequency responses of the rectangular and Hamming windows, which we considered for restricting to a block of the signal.

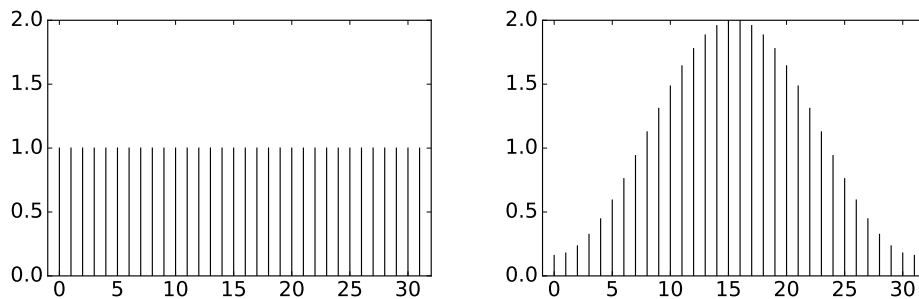


Figure 3.6: The coefficients of the rectangular and Hamming windows, which we considered for restricting to a block of the signal.

### Exercise 3.20: Plotting a simple frequency response

Let again  $S$  be the filter defined by the equation

$$z_n = \frac{1}{4}x_{n+1} + \frac{1}{4}x_n + \frac{1}{4}x_{n-1} + \frac{1}{4}x_{n-2},$$

as in Exercise 3.4. Compute and plot (the magnitude of)  $\lambda_S(\omega)$ .

### Exercise 3.21: Low-pass and high-pass filters

A filter  $S$  is defined by the equation

$$z_n = \frac{1}{3}(x_n + 3x_{n-1} + 3x_{n-2} + x_{n-3}).$$

a) Compute and plot the (magnitude of the continuous) frequency response of the filter, i.e.  $|\lambda_S(\omega)|$ . Is the filter a low-pass filter or a high-pass filter?

b) Find an expression for the vector frequency response  $\lambda_{S,2}$ . What is  $S\mathbf{x}$  when  $\mathbf{x}$  is the vector of length  $N$  with components  $e^{2\pi i 2k/N}$ ?

**Exercise 3.22: Circulant matrices**

A filter  $S_1$  is defined by the equation

$$z_n = \frac{1}{16}(x_{n+2} + 4x_{n+1} + 6x_n + 4x_{n-1} + x_{n-2}).$$

- Write down an  $8 \times 8$  circulant Toeplitz matrix which corresponds to applying  $S_1$  on a periodic signal with period  $N = 8$ .
- Compute and plot (the continuous) frequency response of the filter. Is the filter a low-pass filter or a high-pass filter?
- Another filter  $S_2$  has (continuous) frequency response  $\lambda_{S_2}(\omega) = (e^{i\omega} + 2 + e^{-i\omega})/4$ . Write down the filter coefficients for the filter  $S_1 S_2$ .

**Exercise 3.23: Composite filters**

Assume that the filters  $S_1$  and  $S_2$  have the frequency responses  $\lambda_{S_1}(\omega) = 2 + 4 \cos(\omega)$ ,  $\lambda_{S_2}(\omega) = 3 \sin(2\omega)$ .

- Compute and plot the frequency response of the filter  $S_1 S_2$ .
- Write down the filter coefficients  $t_k$  and the impulse response  $\mathbf{s}$  for the filter  $S_1 S_2$ .

**Exercise 3.24: Maximum and minimum**

Compute and plot the continuous frequency response of the filter  $S = \{1/4, 1/2, 1/4\}$ . Where does the frequency response achieve its maximum and minimum value, and what are these values?

**Exercise 3.25: Plotting a simple frequency response**

Plot the continuous frequency response of the filter  $T = \{1/4, -1/2, 1/4\}$ . Where does the frequency response achieve its maximum and minimum value, and what are these values? Can you write down a connection between this frequency response and that from Exercise 3.24?

**Exercise 3.26: Continuous- and vector frequency responses**

Define the filter  $S$  by  $S = \{1, 2, 3, 4, 5, 6\}$ . Write down the matrix for  $S$  when  $N = 8$ . Plot (the magnitude of)  $\lambda_S(\omega)$ , and indicate the values  $\lambda_{S,n}$  for  $N = 8$  in this plot.



**Exercise 3.27: Starting with circulant matrices**

Given the circulant Toeplitz matrix

$$S = \frac{1}{5} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 0 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}$$

Write down the compact notation for this filter. Compute and plot (the magnitude) of  $\lambda_S(\omega)$ .

**Exercise 3.28: When the filter coefficients are powers**

Assume that  $S = \{\underline{1}, c, c^2, \dots, c^k\}$ . Compute and plot  $\lambda_S(\omega)$  when  $k = 4$  and  $k = 8$ . How does the choice of  $k$  influence the frequency response? How does the choice of  $c$  influence the frequency response?

**Exercise 3.29: The Hanning window**

The Hanning window is defined by  $w_n = 1 - \cos(2\pi n/(N-1))$ . Compute and plot the window coefficients and the continuous frequency response of this window for  $N = 32$ , and compare with the window coefficients and the frequency responses for the rectangular- and the Hamming window.

**3.4 Some examples of filters**

We have now established the basic theory of filters, and it is time to study some specific examples. Some of the filters have the desirable property that they favor certain frequencies, while annihilating others. Such filters have their own names.

**Definition 3.21.** *Lowpass and high-pass filters.*

A filter  $S$  is called

- a *low-pass filter* if  $\lambda_S(\omega)$  is large when  $\omega$  is close to 0, and  $\lambda_S(\omega) \approx 0$  when  $\omega$  is close to  $\pi$  (i.e.  $S$  keeps low frequencies and annihilates high frequencies),
- a *high-pass filter* if  $\lambda_S(\omega)$  is large when  $\omega$  is close to  $\pi$ , and  $\lambda_S(\omega) \approx 0$  when  $\omega$  is close to 0 (i.e.  $S$  keeps high frequencies and annihilates low frequencies),
- a *bandpass filter* if  $\lambda_S(\omega)$  is large within some interval  $[a, b] \subset [0, 2\pi]$ , and  $\lambda_S(\omega) \approx 0$  outside this interval.

This definition should be considered rather vague when it comes to what we mean by “ $\omega$  close to  $0, \pi$ ”, and “ $\lambda_S(\omega)$  is large”: in practice, when we talk about low-pass and high-pass filters, it may be that the frequency responses are still quite far from what is commonly referred to as *ideal low-pass or high-pass filters*, where the frequency response only assumes the values 0 and 1 near 0 and  $\pi$ .

One common application of low-pass filters is to reduce treble in sound, which is a common option in audio systems. The treble in a sound is generated by the fast oscillations (high frequencies) in the signal. Another option in audio systems is to reduce the bass. This corresponds to reducing the low frequencies in the sound, and high-pass filters are suitable for this purpose. It turns out that there is a simple way to jump between low-pass and high-pass filters:

**Observation 3.22.** *Passing between low-pass- and high-pass filters.*

Assume that  $S_2$  is obtained by adding an alternating sign to the filter coefficients of  $S_1$ . If  $S_1$  is a low-pass filter, then  $S_2$  is a high-pass filter, and vice versa.

To see why this is the case, let  $S_1$  be a filter with filter coefficients  $t_k$ , and let us consider the filter  $S_2$  with filter coefficient  $(-1)^k t_k$ . The frequency response of  $S_2$  is

$$\begin{aligned}\lambda_{S_2}(\omega) &= \sum_k (-1)^k t_k e^{-i\omega k} = \sum_k (e^{-i\pi})^k t_k e^{-i\omega k} \\ &= \sum_k e^{-i\pi k} t_k e^{-i\omega k} = \sum_k t_k e^{-i(\omega+\pi)k} = \lambda_{S_1}(\omega + \pi).\end{aligned}$$

where we have set  $e^{-i\pi} = -1$  (note that this is nothing but Property 4. in Theorem 2.7, with  $d = N/2$ ). Now, for a low-pass filter  $S_1$ ,  $\lambda_{S_1}(\omega)$  has large values when  $\omega$  is close to 0 (the low frequencies), and values near 0 when  $\omega$  is close to  $\pi$  (the high frequencies). For a high-pass filter  $S_2$ ,  $\lambda_{S_2}(\omega)$  has values near 0 when  $\omega$  is close to 0 (the low frequencies), and large values when  $\omega$  is close to  $\pi$  (the high frequencies). Therefore, the relation  $\lambda_{S_2}(\omega) = \lambda_{S_1}(\omega + \pi)$  says that  $S_1$  is low-pass when  $S_2$  is high-pass, and vice versa.

### Example 3.30: Adding echo

An echo is a copy of the sound that is delayed and softer than the original sound. If  $x$  is the sound, the sound  $z$  with samples given by

```
N, nchannels = shape(x)
z = zeros((N, nchannels))
z[0:d] = x[0:d] # No echo at the beginning of the signal
z[d:N] = x[d:N] + c*x[0:(N-d)]
z /= abs(z).max()
```

will include an echo of the original sound. This is an example of a filtering operation where each output element is constructed from two input elements.

$d$  is an integer which represents the delay in samples. If you need a delay in  $t$  seconds, simply multiply this with the sample rate to obtain the delay  $d$  in samples ( $d$  needs to be rounded to the nearest integer).  $c$  is a constant called the damping factor. Since an echo is usually weaker than the original sound, we usually have that  $c < 1$ . If we choose  $c > 1$ , the echo will dominate in the sound.

The sample file with echo added with  $d = 10000$  and  $c = 0.5$  can be found in the file [castanetsecho.wav](#).

Using our compact filter notation, the filter which adds echo can be written as

$$S = \{1, 0, \dots, 0, c\},$$

where the damping factor  $c$  appears after the delay  $d$ . The frequency response of this is  $\lambda_S(\omega) = 1 + ce^{-id\omega}$ , which is not real, so that the filter is not symmetric.

In Figure 3.7 we have plotted the magnitude of this frequency response with  $c = 0.1$  and  $d = 10$ .

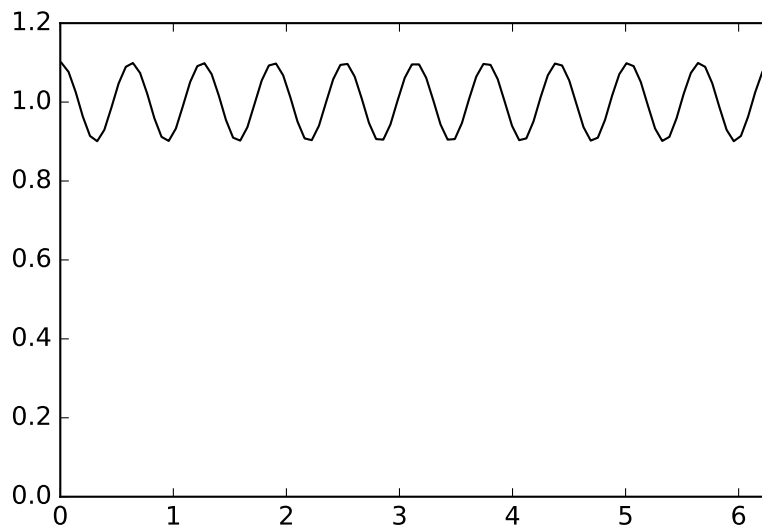


Figure 3.7: The frequency response of a filter which adds an echo with damping factor  $c = 0.1$  and delay  $d = 10$ .

We see that the response varies between 0.9 and 1.1. The deviation from 1 is controlled by the damping factor  $c$ , and the oscillation is controlled by the delay  $d$ .

**Example 3.31: Reducing the treble with moving average filters**

Let us now take a look at filters which adjust the treble in sound. The fact that the filters are useful for these purposes will be clear when we plot the frequency response. A general way of reducing variations in a sequence of numbers is to replace one number by the average of itself and its neighbors, and this is easily done with a digital sound signal. If  $\mathbf{z} = (z_i)_{i=0}^{N-1}$  is the sound signal produced by taking the average of three successive samples, we have that

$$z_n = \frac{1}{3}(x_{n+1} + x_n + x_{n-1}),$$

i.e.  $S = \{1/3, 1/3, 1/3\}$ . This filter is also called a *moving average filter* (with three elements), and it can be written in compact form as. If we set  $N = 4$ , the corresponding circulant Toeplitz matrix for the filter is

$$S = \frac{1}{3} \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

The frequency response is

$$\lambda_S(\omega) = (e^{i\omega} + 1 + e^{-i\omega})/3 = (1 + 2 \cos(\omega))/3.$$

More generally we can construct the moving average filter of  $2L + 1$  elements, which is  $S = \{1, \dots, 1, \dots, 1\}/(2L + 1)$ , where there is symmetry around 0. Clearly then the first column of  $S$  is  $\mathbf{s} = (\underbrace{1, \dots, 1}_{L+1 \text{ times}}, 0, \dots, 0, \underbrace{1, \dots, 1}_{L \text{ times}})/(2L + 1)$ . In

Example 2.2 we computed that the DFT of the vector  $\mathbf{x} = (\underbrace{1, \dots, 1}_{L+1 \text{ times}}, 0, \dots, 0, \underbrace{1, \dots, 1}_{L \text{ times}})$  had components

$$y_n = \frac{\sin(\pi n(2L + 1)/N)}{\sin(\pi n/N)}.$$

Since  $\mathbf{s} = \mathbf{x}/(2L + 1)$  and  $\lambda_S = \text{DFT}_N \mathbf{s}$ , the frequency response of  $S$  is

$$\lambda_{S,n} = \frac{1}{2L + 1} \frac{\sin(\pi n(2L + 1)/N)}{\sin(\pi n/N)},$$

so that

$$\lambda_S(\omega) = \frac{1}{2L + 1} \frac{\sin((2L + 1)\omega/2)}{\sin(\omega/2)}.$$

We clearly have

$$0 \leq \frac{1}{2L + 1} \frac{\sin((2L + 1)\omega/2)}{\sin(\omega/2)} \leq 1,$$

and this frequency response approaches 1 as  $\omega \rightarrow 0$ . The frequency response thus peaks at 0, and this peak gets narrower and narrower as  $L$  increases, i.e. as

we use more and more samples in the averaging process. This filter thus “keeps” only the lowest frequencies. When it comes to the highest frequencies it is seen that the frequency response is small for  $\omega \approx \pi$ . In fact it is straightforward to see that  $|\lambda_S(\pi)| = 1/(2L + 1)$ . In Figure 3.8 we have plotted the frequency response for moving average filters with  $L = 1$ ,  $L = 5$ , and  $L = 20$ .

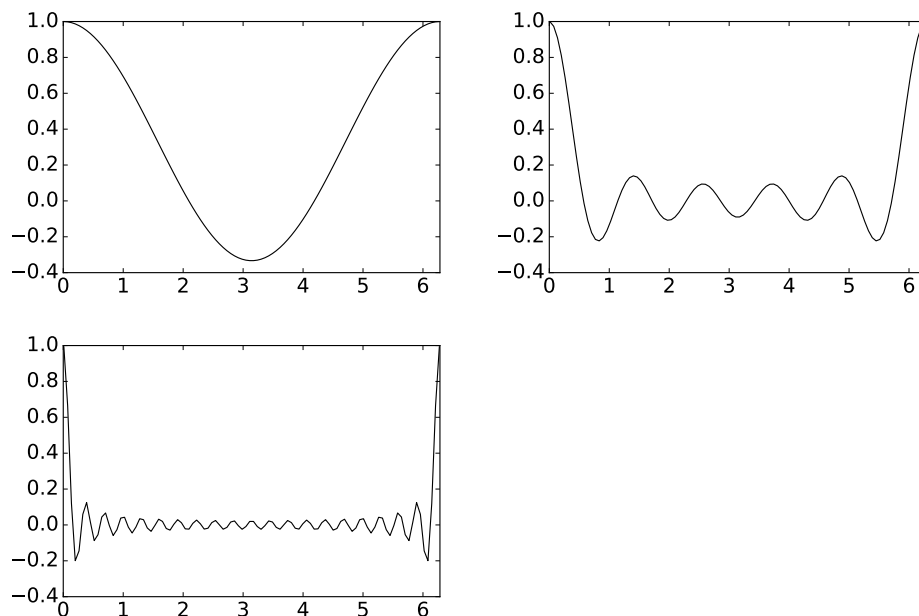


Figure 3.8: The frequency response of moving average filters with  $L = 1$ ,  $L = 5$ , and  $L = 20$ .

Unfortunately, the frequency response is far from a filter which keeps some frequencies unaltered, while annihilating others: Although the filter distinguishes between high and low frequencies, it slightly changes the small frequencies. Moreover, the higher frequencies are not annihilated, even when we increase  $L$  to high values.

### Example 3.32: Ideal low-pass filters

By definition, the ideal low-pass filter keeps frequencies near 0 unchanged, and completely removes frequencies near  $\pi$ . We now have the theory in place in order to find the filter coefficients for such a filter: In Example 2.16 we implemented the ideal low-pass filter with the help of the DFT. Mathematically you can see that this code is equivalent to computing  $(F_N)^H D F_N$  where  $D$  is the diagonal matrix with the entries  $0, \dots, L$  and  $N - L, \dots, N - 1$  being 1, the rest being 0. Clearly this is a digital filter, with frequency response as stated. If the filter should keep the angular frequencies  $|\omega| \leq \omega_c$  only, where  $\omega_c$  describes the highest frequency we should keep, we should choose  $L$  so that  $\omega_c = 2\pi L/N$ . Again, in Example 2.2

we computed the DFT of this vector, and it followed from Theorem 2.7 that the IDFT of this vector equals its DFT. This means that we can find the filter coefficients by using Equation (3.10), i.e. we take an IDFT. We then get the filter coefficients

$$\frac{1}{N} \frac{\sin(\pi k(2L + 1)/N)}{\sin(\pi k/N)}.$$

This means that the filter coefficients lie as  $N$  points uniformly spaced on the curve  $\frac{1}{N} \frac{\sin(\pi t(2L+1)/2)}{\sin(\pi t/2)}$  between 0 and 1. This curve has been encountered many other places in these notes. The filter which keeps only the frequency  $\omega_c = 0$  has all filter coefficients being  $\frac{1}{N}$  (set  $L = 1$ ), and when we include all frequencies (set  $L = N$ ) we get the filter where  $x_0 = 1$  and all other filter coefficients are 0. When we are between these two cases, we get a filter where  $s_0$  is the biggest coefficient, while the others decrease towards 0 along the curve we have computed. The bigger  $L$  and  $N$  are, the quicker they decrease to zero. All filter coefficients are usually nonzero for this filter, since this curve is zero only at certain points. This is unfortunate, since it means that the filter is time-consuming to compute.

The two previous examples show an important duality between vectors which are 1 on some elements and 0 on others (also called window vectors), and the vector  $\frac{1}{N} \frac{\sin(\pi k(2L+1)/N)}{\sin(\pi k/N)}$  (also called a sinc): filters of the one type correspond to frequency responses of the other type, and vice versa. The examples also show that, in some cases only the filter coefficients are known, while in other cases only the frequency response is known. In any case we can deduce the one from the other, and both cases are important.

Filters are much more efficient when there are few nonzero filter coefficients. In this respect the second example displays a problem: in order to create filters with particularly nice properties (such as being an ideal low-pass filter), one may need to sacrifice computational complexity by increasing the number of nonzero filter coefficients. The trade-off between computational complexity and desirable filter properties is a very important issue in *filter design theory*.

### Example 3.33: Dropping filter coefficients

In order to decrease the computational complexity for the ideal low-pass filter in Example 3.32, one can for instance include only the first filter coefficients, i.e.

$$\left\{ \frac{1}{N} \frac{\sin(\pi k(2L + 1)/N)}{\sin(\pi k/N)} \right\}_{k=-N_0}^{N_0}.$$

Hopefully this gives us a filter where the frequency response is not that different from the ideal low-pass filter. Let us set  $N = 128$ ,  $L = 32$ , so that the filter removes all frequencies  $\omega > \pi/2$ . In Figure 3.9 we show the corresponding frequency responses.  $N_0$  has been chosen so that the given percentage of all coefficients are included.

This shows that we should be careful when we omit filter coefficients: if we drop too many, the frequency response is far away from that of an ideal bandpass

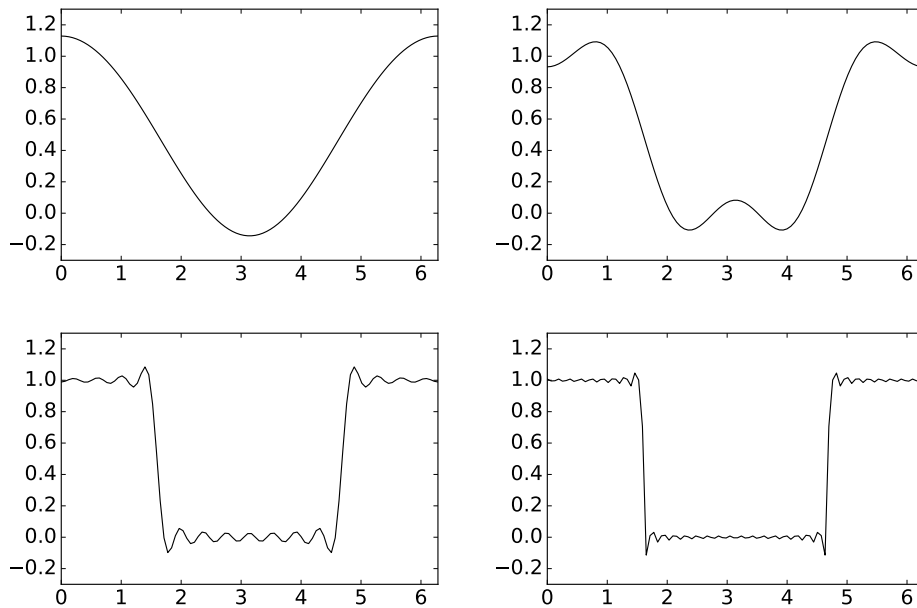


Figure 3.9: The frequency response which results by including the first  $1/32$ , the first  $1/16$ , the first  $1/4$ , and all of the filter coefficients for the ideal low-pass filter.

filter. In particular, we see that the new frequency response oscillates wildly near the discontinuity of the ideal low-pass filter. Such oscillations are called *Gibbs oscillations*.

### Example 3.34: Filters and the MP3 standard

We mentioned previously that the MP3 standard splits the sound into frequency bands. This splitting is actually performed by particular filters, which we will consider now. In the example above, we saw that when we dropped the last filter coefficients in the ideal low-pass filter, there were some undesired effects in the frequency response of the resulting filter. Are there other and better approximations to the ideal low-pass filter which uses the same number of filter coefficients? This question is important, since the ear is sensitive to certain frequencies, and we would like to extract these frequencies for special processing, using as low computational complexity as possible. In the MP3-standard, such filters have been constructed. These filters are more advanced than the ones we have seen up to now. They have as many as 512 filter coefficients! We will not go into the details on how these filters are constructed, but only show how their frequency responses look.

In the left plot in Figure 3.10, the “prototype filter” used in the MP3 standard is shown. We see that this is very close to an ideal low-pass filter. Moreover, many

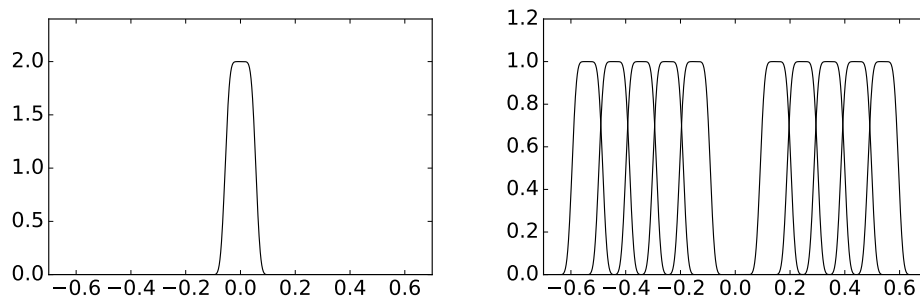


Figure 3.10: Frequency responses of some filters used in the MP3 standard. The prototype filter is shown left. The other frequency responses at right are simply shifted copies of this.

of the undesirable effect from the previous example have been eliminated: The oscillations near the discontinuities are much smaller, and the values are lower away from 0. Using Property 4 in Theorem 2.7, it is straightforward to construct filters with similar frequency responses, but centered around different frequencies: We simply need to multiply the filter coefficients with a complex exponential, in order to obtain a filter where the frequency response has been shifted to the left or right. In the MP3 standard, this observation is used to construct 32 filters, each having a frequency response which is a shifted copy of that of the prototype filter, so that all filters together cover the entire frequency range. 5 of these frequency responses are shown in the right plot in Figure 3.10. To understand the effects of the different filters, let us apply them to our sample sound. If you apply all filters in the MP3 standard in successive order with the most low-pass filters first, the result can be found in the file [mp3bands.wav](#). You should interpret the result as low frequencies first, followed by the high frequencies.  $\pi$  corresponds to the frequency 22.05KHz (i.e. the highest representable frequency equals half the sampling rate on 44.1KHz). The different filters are concentrated on  $1/32$  of these frequencies each, so that the angular frequencies you here are  $[\pi/64, 3\pi/64]$ ,  $[3\pi/64, 5\pi/64]$ ,  $[5\pi/64, 7\pi/64]$ , and so on, in that order.

In Section 3.3.1 we mentioned that the psychoacoustic model of the MP3 standard applied a window to the sound data, followed by an FFT to that data. This is actually performed in parallel on the same sound data. Applying two different operations in parallel to the sound data may seem strange. In the MP3 standard [20] (p. 109) this is explained by “the lack of spectral selectivity obtained at low frequencies“ by the filters above. In other words, the FFT can give more precise frequency information than the filters can. This more precise information is then used to compute psychoacoustic information such as masking thresholds, and this information is applied to the output of the filters.



**Example 3.35: Reducing the treble using Pascals triangle**

When reducing the treble it is reasonable to let the middle sample  $x_i$  count more than the neighbors in the average, so an alternative is to compute the average by instead writing

$$z_n = (x_{n-1} + 2x_n + x_{n+1})/4$$

The coefficients 1, 2, 1 here have been taken from row 2 in Pascal's triangle. It turns out that this is a good choice of coefficients. Also if we take averages of more numbers it will turn out that higher rows of Pascals triangle are good choices. Let us take a look at why this is the case. Let  $S$  be the moving average filter of two elements, i.e.

$$(S\mathbf{x})_n = \frac{1}{2}(x_{n-1} + x_n).$$

In Example 3.31 we had an odd number of filter coefficients. Here we have only two. We see that the frequency response in this case is

$$\lambda_S(\omega) = \frac{1}{2}(1 + e^{-i\omega}) = e^{-i\omega/2} \cos(\omega/2).$$

The frequency response is complex now, since the filter is not symmetric in this case. Let us now apply this filter  $k$  times, and denote by  $S_k$  the resulting filter. Theorem 3.18 gives us that the frequency response of  $S_k$  is

$$\lambda_{S^k}(\omega) = \frac{1}{2^k}(1 + e^{-i\omega})^k = e^{-ik\omega/2} \cos^k(\omega/2),$$

which is a polynomial in  $e^{-i\omega}$  with the coefficients taken from Pascal's triangle (remember that the values in Pascals triangle are the coefficients of  $x$  in the expression  $(1 + x)^k$ , i.e. the binomial coefficients  $\binom{k}{r}$  for  $0 \leq r \leq k$ ). At least, this partially explains how filters with coefficients taken from Pascal's triangle appear. The reason why these are more desirable than moving average filters, and are used much for smoothing abrupt changes in images and in sound, is the following: Since  $(1 + e^{-i\omega})^k$  is a factor in  $\lambda_{S^k}(\omega)$ , it has a zero of multiplicity of  $k$  at  $\omega = \pi$ . In other words, when  $k$  is large,  $\lambda_{S^k}$  has a zero of high multiplicity at  $\omega = \pi$ , and this implies that the frequency response is very flat for  $\omega \approx \pi$  when  $k$  increases, i.e. the filter is good at removing the highest frequencies. This can be seen in Figure 3.11, where we have plotted the magnitude of the frequency response when  $k = 5$ , and when  $k = 30$ . Clearly the latter frequency response is much flatter for  $\omega \approx \pi$ . On the other side, it is easy to show that the moving average filters of Example 3.31 had a zero of multiplicity one at  $\omega = \pi$ , regardless of  $L$ . Clearly, the corresponding frequency responses, shown in Figure 3.8, were not as flat for  $\omega \approx \pi$ , when compared to the ones in Figure 3.11.

While using  $S^k$  gives a desirable behaviour for  $\omega \approx \pi$ , we see that the behaviour is not so desirable for small frequencies  $\omega \approx 0$ : Only frequencies very close to 0 are kept unaltered. It should be possible to produce better low-pass

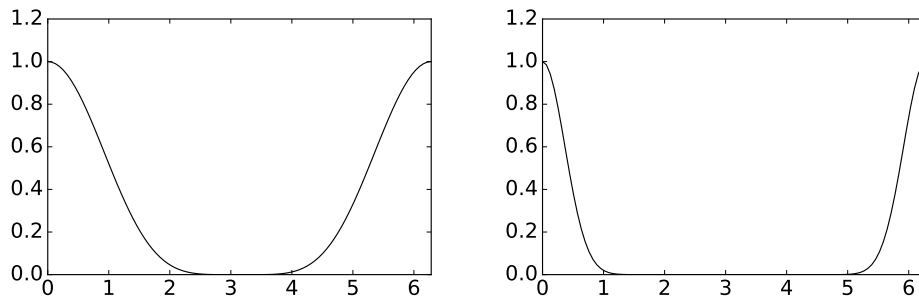


Figure 3.11: The frequency response of filters corresponding to iterating the moving average filter  $\{1/2, 1/2\}$   $k = 5$  and  $k = 30$  times (i.e. using row  $k$  in Pascal's triangle).

filters than this also, and the frequency responses we plotted for the filters used in the MP3 standard gives an indication to this.

Let us now see how to implement the filters  $S^k$ . Since convolution corresponds to multiplication of polynomials, we can obtain their filter coefficients with the following code

```
t = [1.]
for kval in range(k):
    t = convolve(t, [1/2., 1/2.])
```

Note that  $S^k$  has  $k + 1$  filter coefficients, and that  $S^k$  corresponds to the filter coefficients of a symmetric filter when  $k$  is even. Having computed  $\mathbf{t}$ , we can simply compute the convolution of the input  $\mathbf{x}$  and  $\mathbf{t}$ . In using `conv` we disregard the circularity of  $S$ , and we introduce a time delay. These issues will, however, not be audible when we listen to the output. An example of the result of smoothing is shown in Figure 3.12.

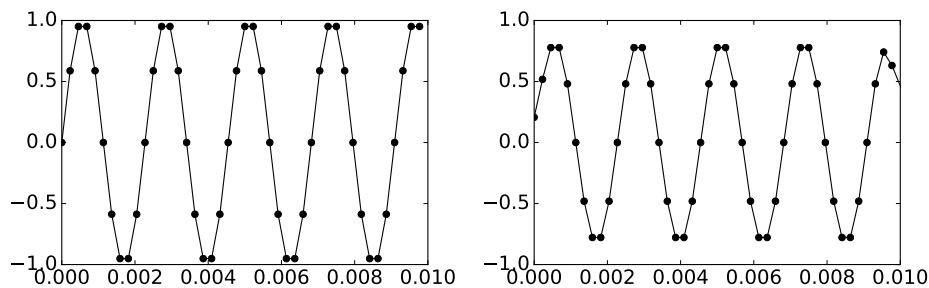


Figure 3.12: Reducing the treble. The original sound signal is shown left, the result after filtering using row 4 in Pascal's triangle is shown right.

The left plot shows the samples of the pure sound with frequency 440Hz (with sampling frequency  $f_s = 4400\text{Hz}$ ). The right plot shows the result of

applying the averaging process by using row 4 of Pascals triangle. We see that the oscillations have been reduced. In Exercise 3.39 you will be asked to implement reducing the treble in our sample audio file. If you do this you should hear that the sound gets softer when you increase  $k$ : For  $k = 32$  the sound can be found in the file [castanetstreble32.wav](#), for  $k = 256$  it can be found in the file [castanetstreble256.wav](#).

### Example 3.36: Reducing the bass using Pascals triangle

Due to Observation 3.22 and Example 3.35, we can create bass-reducing filters by adding an alternating sign to rows in Pascals triangle. Consider the bass-reducing filter deduced from the fourth row in Pascals triangle:

$$z_n = \frac{1}{16}(x_{n-2} - 4x_{n-1} + 6x_n - 4x_{n+1} + x_{n+2})$$

Let us apply this filter to the sound in Figure 3.12. The result is shown in Figure 3.13.

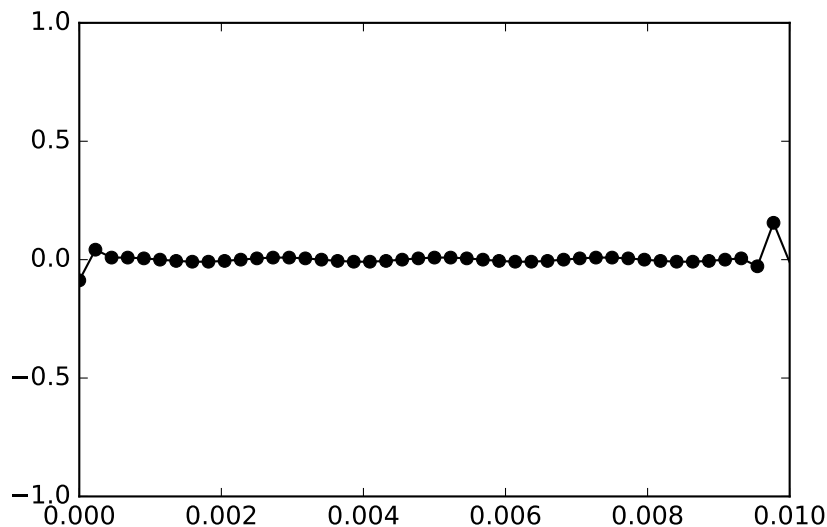


Figure 3.13: The result of applying the bass-reducing filter deduced from row 4 in Pascals triangle to the pure sound in the left plot of Figure 3.12.

We observe that the samples oscillate much more than the samples of the original sound. In Exercise 3.39 you will be asked to implement reducing the bass in our sample audio file. The new sound will be difficult to hear for large  $k$ , and we will explain why later. For  $k = 1$  the sound can be found in the file [castanetsbass1.wav](#), for  $k = 2$  it can be found in the file [castanetsbass2.wav](#).

Even if the sound is quite low, you can hear that more of the bass has disappeared for  $k = 2$ .

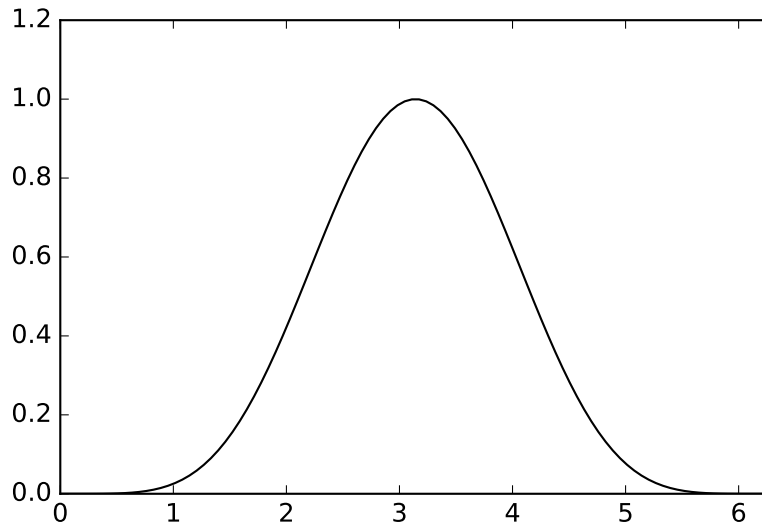


Figure 3.14: The frequency response of the bass reducing filter, which corresponds to row 5 of Pascal's triangle.

The frequency response we obtain from using row 5 of Pascal's triangle is shown in Figure 3.14. It is just the frequency response of the corresponding treble-reducing filter shifted with  $\pi$ . The alternating sign can also be achieved if we write the frequency response  $\frac{1}{2^k}(1+e^{-i\omega})^k$  from Example 3.35 as  $\frac{1}{2^k}(1-e^{-i\omega})^k$ , which corresponds to applying the filter  $S(\mathbf{x}) = \frac{1}{2}(-x_{n-1} + x_n)$   $k$  times.

### Exercise 3.37: Composing time delay filters

Let  $E_{d_1}$  and  $E_{d_2}$  be two time delay filters. Show that  $E_{d_1}E_{d_2} = E_{d_1+d_2}$  (i.e. that the composition of two time delays again is a time delay) in two different ways:

- Give a direct argument which uses no computations.
- By using Property 3 in Theorem 2.7, i.e. by using a property for the Discrete Fourier Transform.

### Exercise 3.38: Adding echo filters

Consider the two filters  $S_1 = \{1, 0, \dots, 0, c\}$  and  $S_2 = \{1, 0, \dots, 0, -c\}$ . Both of these can be interpreted as filters which add an echo. Show that  $\frac{1}{2}(S_1 + S_2) = I$ . What is the interpretation of this relation in terms of echos?

**Exercise 3.39: Reducing bass and treble**

Write code where you reduce the treble and the bass as described in examples 3.35 and 3.36, generate the sounds you heard in these examples, and verify that they are the same. In your code, it will be necessary to scale the values after reducing the bass, but not the values after reducing the treble. Explain why this is the case. How high must  $k$  be in order for you to hear difference from the actual sound? How high can you choose  $k$  and still recognize the sound at all? If you solved Exercise 3.9, you can also use the function `filterS` to perform the filtering, rather than using the `convolve` function (the latter disregards circularity).

**Exercise 3.40: Constructing a high-pass filter**

Consider again Example 3.32. Find an expression for a filter so that only frequencies so that  $|\omega - \pi| < \omega_c$  are kept, i.e. the filter should only keep angular frequencies close to  $\pi$  (i.e. here we construct a high-pass filter).

**Exercise 3.41: Combining low-pass and high-pass filters**

In this exercise we will investigate how we can combine low-pass and high-pass filters to produce other filters

- a) Assume that  $S_1$  and  $S_2$  are low-pass filters. What kind of filter is  $S_1S_2$ ? What if both  $S_1$  and  $S_2$  are high-pass filters?
- b) Assume that one of  $S_1, S_2$  is a high-pass filter, and that the other is a low-pass filter. What kind of filter  $S_1S_2$  in this case?

**Exercise 3.42: Composing filters**

A filter  $S_1$  has the frequency response  $\frac{1}{2}(1 + \cos \omega)$ , and another filter has the frequency response  $\frac{1}{2}(1 + \cos(2\omega))$ .

- a) Is  $S_1S_2$  a low-pass filter, or a high-pass filter?
- b) What does the filter  $S_1S_2$  do with angular frequencies close to  $\omega = \pi/2$ .
- c) Find the filter coefficients of  $S_1S_2$ .

**Hint.** Use Theorem 3.18 to compute the frequency response of  $S_1S_2$  first.

- d) Write down the matrix of the filter  $S_1S_2$  for  $N = 8$ .

**Exercise 3.43: Composing filters**

An operation describing some transfer of data in a system is defined as the composition of the following three filters:

- First a time delay filter with delay  $d_1 = 2$ , due to internal transfer of data in the system,

- then the treble-reducing filter  $T = \{1/4, \underline{1/2}, 1/4\}$ ,
- finally a time delay filter with delay  $d_2 = 4$  due to internal transfer of the filtered data.

We denote by  $T_2 = E_{d_2}TE_{d_1} = E_4TE_2$  the operation which applies these filters in succession.

a) Explain why  $T_2$  also is a digital filter. What is (the magnitude of) the frequency response of  $E_{d_1}$ ? What is the connection between (the magnitude of) the frequency response of  $T$  and  $T_2$ ?

b) Show that  $T_2 = \{0, 0, 0, 0, 1/4, 1/2, 1/4\}$ .

**Hint.** Use the expressions  $(E_{d_1}\mathbf{x})_n = x_{n-d_1}$ ,  $(T\mathbf{x})_n = \frac{1}{4}x_{n+1} + \frac{1}{2}x_n + \frac{1}{4}x_{n-1}$ ,  $(E_{d_2}\mathbf{x})_n = x_{n-d_2}$ , and compute first  $(E_{d_1}\mathbf{x})_n$ , then  $(TE_{d_1}\mathbf{x})_n$ , and finally  $(T_2\mathbf{x})_n = (E_{d_2}TE_{d_1}\mathbf{x})_n$ . From the last expression you should be able to read out the filter coefficients.

c) Assume that  $N = 8$ . Write down the  $8 \times 8$ -circulant Toeplitz matrix for the filter  $T_2$ .

### Exercise 3.44: Filters in the MP3 standard

In Example 3.34, we mentioned that the filters used in the MP3-standard were constructed from a low-pass prototype filter by multiplying the filter coefficients with a complex exponential. Clearly this means that the new frequency response is a shift of the old one. The disadvantage is, however, that the new filter coefficients are complex. It is possible to address this problem as follows. Assume that  $t_k$  are the filter coefficients of a filter  $S_1$ , and that  $S_2$  is the filter with filter coefficients  $\cos(2\pi kn/N)t_k$ , where  $n \in \mathbb{N}$ . Show that

$$\lambda_{S_2}(\omega) = \frac{1}{2}(\lambda_{S_1}(\omega - 2\pi n/N) + \lambda_{S_1}(\omega + 2\pi n/N)).$$

In other words, when we multiply (modulate) the filter coefficients with a cosine, the new frequency response can be obtained by shifting the old frequency response with  $2\pi n/N$  in both directions, and taking the average of the two.

### Exercise 3.45: Explain code

a) Explain what the code below does, line by line.

```
x, fs = audioread('sounds/castanets.wav')
N, nchannels = shape(x)
z = zeros((N, nchannels))
for n in range(1, N-1):
    z[n] = 2*x[n+1] + 4*x[n] + 2*x[n-1]
z[0] = 2*x[1] + 4*x[0] + 2*x[N-1]
z[N-1] = 2*x[0] + 4*x[N-1] + 2*x[N-2]
z = z/abs(z).max()
play(z, fs)
```

Comment in particular on what happens in the three lines directly after the `for`-loop, and why we do this. What kind of changes in the sound do you expect to hear?

b) Write down the compact filter notation for the filter which is used in the code, and write down a  $5 \times 5$  circulant Toeplitz matrix which corresponds to this filter. Plot the (continuous) frequency response. Is the filter a low-pass- or a high-pass filter?

c) Another filter is given by the circulant Toeplitz matrix

$$\begin{pmatrix} 4 & -2 & 0 & 0 & -2 \\ -2 & 4 & -2 & 0 & 0 \\ 0 & -2 & 4 & -2 & 0 \\ 0 & 0 & -2 & 4 & -2 \\ -2 & 0 & 0 & -2 & 4 \end{pmatrix}.$$

Express a connection between the frequency responses of this filter and the filter from b). Is the new filter a low-pass- or a high-pass filter?

### 3.5 More general filters

The starting point for defining filters at the beginning of this chapter was equations on the form

$$z_n = \sum_k t_k x_{n-k}.$$

For most filters we have looked at, we had a limited number of nonzero  $t_k$ , and this enabled us to compute them on a computer using a finite number of additions and multiplications. Filters which have a finite number of nonzero filter coefficients are also called *FIR-filters* (FIR is short for Finite Impulse Response. Recall that the impulse response of a filter can be found from the filter coefficients). However, there exist many useful filters which are not FIR filters, i.e. where the sum above is infinite. The ideal lowpass filter from Example 3.32 was one example. It turns out that many such cases can be made computable if we change our procedure slightly. The old procedure for computing a filter is to compute  $\mathbf{z} = S\mathbf{x}$ . Consider the following alternative:

**Idea 3.23.** *More general filters (1).*

Let  $\mathbf{x} \in \mathbb{R}^N$ , and  $T$  an  $N \times N$  filter. By solving the system  $T\mathbf{z} = \mathbf{x}$  for  $\mathbf{z}$  we get another filter, which we denote by  $S$ .

Of course  $T$  must then be the inverse of  $S$  (which also is a filter), but the point is that the inverse of a filter may have a finite number of filter coefficients, even if the filter itself does not. In such cases this new procedure is more attractive than the old one, since the equation system can be solved with few arithmetic operations when  $T$  has few filter coefficients.

It turns out that there also are highly computable filters where neither the filter nor its inverse have a finite number of filter coefficients. Consider the following idea:

**Idea 3.24.** *More general filters (2).*

Let  $\mathbf{x}$  be the input to a filter, and let  $U$  and  $V$  be filters. By solving the system  $U\mathbf{z} = V\mathbf{x}$  for  $\mathbf{z}$  we get another filter, which we denote by  $S$ . The filter  $S$  can be implemented in two steps: first we compute the right hand side  $\mathbf{y} = V\mathbf{x}$ , and then we solve the equation  $U\mathbf{z} = \mathbf{y}$ .

If both  $U$  and  $V$  are invertible we have that the filter is  $S = U^{-1}V$ , and this is invertible with inverse  $S^{-1} = V^{-1}U$ . The point is that, when  $U$  and  $V$  have a finite number of filter coefficients, both  $S$  and its inverse will typically have an infinite number of filter coefficients. The filters from this idea are thus more general than the ones from the previous idea, and the new idea makes a wider class of filters implementable using row reduction of sparse matrices. Computing a filter by solving  $U\mathbf{z} = V\mathbf{x}$  may also give meaning when the matrices  $U$  and  $V$  are singular: The matrix system can have a solution even if  $U$  is singular. Therefore we should be careful in using the form  $T = U^{-1}V$ .

We have the following result concerning the frequency responses:

**Theorem 3.25.** *Frequency response of IIR filters.*

Assume that  $S$  is the filter defined from the equation  $U\mathbf{z} = V\mathbf{x}$ . Then we have that  $\lambda_S(\omega) = \frac{\lambda_V(\omega)}{\lambda_U(\omega)}$  whenever  $\lambda_U(\omega) \neq 0$ .

*Proof.* Set  $\mathbf{x} = \phi_n$ . We have that  $U\mathbf{z} = \lambda_{U,n}\lambda_{S,n}\phi_n$ , and  $V\mathbf{x} = \lambda_{V,n}\phi_n$ . If the expressions are equal we must have that  $\lambda_{U,n}\lambda_{S,n} = \lambda_{V,n}$ , so that  $\lambda_{S,n} = \frac{\lambda_{V,n}}{\lambda_{U,n}}$  for all  $n$ . By the definition of the continuous frequency response this means that  $\lambda_S(\omega) = \frac{\lambda_V(\omega)}{\lambda_U(\omega)}$  whenever  $\lambda_U(\omega) \neq 0$ .  $\square$

The following example clarifies the points made above, and how one may construct  $U$  and  $V$  from  $S$ . The example also shows that, in addition to making some filters with infinitely many filter coefficients computable, the procedure  $U\mathbf{z} = V\mathbf{x}$  for computing a filter can also reduce the complexity in some filters where we already have a finite number of filter coefficients.

### Example 3.46: Moving average filter

Consider again the moving average filter  $S$  from Example 3.31:

$$z_n = \frac{1}{2L+1}(x_{n+L} + \cdots + x_n + \cdots + x_{n-L}).$$

If we implemented this directly,  $2L$  additions would be needed for each  $n$ , so that we would need a total of  $2NL$  additions. However, we can also write



$$\begin{aligned}
z_{n+1} &= \frac{1}{2L+1}(x_{n+1+L} + \cdots + x_{n+1} + \cdots + x_{n+1-L}) \\
&= \frac{1}{2L+1}(x_{n+L} + \cdots + x_n + \cdots + x_{n-L}) + \frac{1}{2L+1}(x_{n+1+L} - x_{n-L}) \\
&= z_n + \frac{1}{2L+1}(x_{n+1+L} - x_{n-L}).
\end{aligned}$$

This means that we can also compute the output from the formula

$$z_{n+1} - z_n = \frac{1}{2L+1}(x_{n+1+L} - x_{n-L}),$$

which can be written on the form  $Uz = Vx$  with  $U = \{1, -1\}$  and  $V = \frac{1}{2L+1}\{1, 0, \dots, 0, -1\}$  where the 1 is placed at index  $-L-1$  and the  $-1$  is placed at index  $L$ . We now perform only  $2N$  additions in computing the right hand side, and solving the equation system requires only  $2(N-1)$  additions. The total number of additions is thus  $2N + 2(N-1) = 4N - 2$ , which is much less than the previous  $2LN$  when  $L$  is large.

A perhaps easier way to find  $U$  and  $V$  is to consider the frequency response of the moving average filter, which is

$$\begin{aligned}
\frac{1}{2L+1}(e^{-Li\omega} + \dots + e^{Li\omega}) &= \frac{1}{2L+1}e^{-Li\omega} \frac{1 - e^{(2L+1)i\omega}}{1 - e^{i\omega}} \\
&= \frac{\frac{1}{2L+1}(-e^{(L+1)i\omega} + e^{-Li\omega})}{1 - e^{i\omega}},
\end{aligned}$$

where we have used the formula for the sum of a geometric series. From here we easily see the frequency responses of  $U$  and  $V$  from the numerator and the denominator.

Filters with an infinite number of filter coefficients are also called *IIR filters* (IIR stands for *Infinite Impulse Response*). Thus, we have seen that some IIR filters may still have efficient implementations.

### Exercise 3.47: A concrete IIR filter

A filter is defined by demanding that  $z_{n+2} - z_{n+1} + z_n = x_{n+1} - x_n$ .

- a) Compute and plot the frequency response of the filter.
- b) Use a computer to compute the output when the input vector is  $x = (1, 2, \dots, 10)$ . In order to do this you should write down two  $10 \times 10$ -circulant Toeplitz matrices.

## 3.6 Implementation of filters

As we saw in Example 3.46, a filter with many filter coefficients could be factored into the application of two simpler filters, and this could be used as a basis for an efficient implementation. There are also several other possible efficient implementations of filters. In this section we will consider two such techniques. The first technique considers how we can use the DFT to speed up the computation of filters. The second technique considers how we can factorize a filter into a product of simpler filters.

### 3.6.1 Implementation of filters using the DFT

If there are  $k$  filter coefficients, a direct implementation of a filter would require  $kN$  multiplications. Since filters are diagonalized by the DFT, one can also compute the filter as the product  $S = F_N^H D F_N$ . This would instead require  $O(N \log_2 N)$  complex multiplications when we use the FFT algorithm, which may be a higher number of multiplications. We will however see that, by slightly changing our algorithm, we may end up with a DFT-based implementation of the filter which requires fewer multiplications.

The idea is to split the computation of the filter into smaller parts. Assume that we compute  $M$  elements of the filter at a time. If the nonzero filter coefficients of  $S$  are  $t_{-k_0}, \dots, t_{k-k_0-1}$ , we have that

$$(S\mathbf{x})_t = \sum_r t_r x_{s-r} = t_{-k_0} x_{t+k_0} + \dots + t_{k-k_0-1} x_{t-(k-k_0-1)}.$$

From this it is clear that  $(S\mathbf{x})_t$  only depends on  $x_{t-(k-k_0-1)}, \dots, x_{t+k_0}$ . This means that, if we restrict the computation of  $S$  to  $x_{t-(k-k_0-1)}, \dots, x_{t+M-1+k_0}$ , the outputs  $x_t, \dots, x_{t+M-1}$  will be the same as without this restriction. This means that we can compute the output  $M$  elements at a time, at each step multiplying with a circulant Toeplitz matrix of size  $(M+k-1) \times (M+k-1)$ . If we choose  $M$  so that  $M+k-1 = 2^r$ , we can use the FFT and IFFT algorithms to compute  $S = F_N^H D F_N$ , and we require  $O(r2^r)$  multiplications for every block of length  $M$ . The total number of multiplications is  $\frac{Nr2^r}{M} = \frac{Nr2^r}{2^r - k + 1}$ . If  $k = 128$ , you can check on your calculator that the smallest value is for  $r = 10$  with value  $11.4158 \times N$ . Since the direct implementation gives  $kN$  multiplications, this clearly gives a benefit for the new approach, it gives a 90% decrease in the number of multiplications.

### 3.6.2 Factoring a filter

In practice, filters are often applied in hardware, and applied in real-time scenarios where performance is a major issue. The most CPU-intensive tasks in such applications often have few memory locations available. These tasks are thus not compatible with filters with many filter coefficients, since for each output sample we then need access to many input samples and filter coefficients. A strategy which addresses this is to factorize the filter into the product of several smaller

filters, and then applying each filter in turn. Since the frequency response of the product of filters equals the product of the frequency responses, we get the following idea:

**Idea 3.26.** *Factorizing a filter.*

Let  $S$  be a filter with real coefficients. Assume that

$$\lambda_S(\omega) = Ke^{ik\omega}(e^{i\omega} - a_1) \dots (e^{i\omega} - a_m)(e^{2i\omega} + b_1e^{i\omega} + c_1) \dots (e^{2i\omega} + b_ne^{i\omega} + c_n). \quad (3.14)$$

Then we can write  $S = KE_k A_1 \dots A_m B_1 \dots B_n$ , where  $A_i = \{1, -a_i\}$  and  $B_i = \{1, b_i, c_i\}$ .

Note that in Equation (3.14)  $a_i$  correspond to the real roots of the frequency response, while  $b_i, c_i$  are obtained by pairing the complex conjugate roots. Clearly the frequency responses of  $A_i, B_i$  equal the factors in the frequency response of  $S$ , which in any case can be factored into the product of filters with 2 and 3 filter coefficients, followed by a time-delay.

Note that, even though this procedure factorizes a filter into smaller parts (which is attractive for hardware implementations since smaller filters require fewer locations in memory), the number of arithmetic operations is usually not reduced. However, consider Example 3.35, where we factorized the treble-reducing filters into a product of moving average filters of length 2 (all roots in the previous idea are real, and equal). Each application of a moving average filter of length 2 does not really require any multiplications, since multiplication with  $\frac{1}{2}$  corresponds to a bitshift. Therefore, the factorization of Example 3.35 removes the need for doing any multiplications at all, while keeping the number of additions the same. There are computational savings in this case, due to the special filter structure here.

### Exercise 3.48: Implementing the factorization

Write a function `filterdftimpl`, which takes the filter coefficients  $\mathbf{t}$  and the value  $k_0$  from this section, computes the optimal  $M$ , and implements the filter as here.

### Exercise 3.49: Factoring concrete filter

Factor the filter  $S = \{1, 5, 10, 6\}$  into a product of two filters, one with two filter coefficients, and one with three filter coefficients.

## 3.7 Summary

We defined digital filters, which do the same job for digital sound as analog filters do for (continuous) sound. Digital filters turned out to be linear transformations diagonalized by the DFT. We proved several other equivalent characterizations

of digital filters as well, such as being time-invariant, and having a matrix which is circulant and Toeplitz. Just as for continuous sound, digital filters are characterized by their frequency response, which explains how the filter treats the different frequencies. We also went through several important examples of filters, some of which corresponded to meaningful operations on sound, such as adjustment of bass and treble, and adding echo. We also explained that there exist filters with useful implementations which have an infinite number of filter coefficients, and we considered techniques for implementing filters efficiently. Most of the topics covered on that can also be found in [37]. We also took a look at the role of filters in the MP3 standard for compression of sound.

In signal processing literature, the assumption that vectors are periodic is often not present, and filters are thus not defined as finite-dimensional operations. With matrix notation they would then be viewed as infinite matrices which have the Toeplitz structure (i.e. constant values on the diagonals), but with no circulation. The circulation in the matrices, as well as the restriction to finite vectors, come from the assumption of a periodic vector. There are, however, also some books which view filters as circulant Toeplitz matrices as we have done, such as [16].

#### What you should have learned in this chapter.

- How to write down the circulant Toeplitz matrix from a digital filter expression, and vice versa.
- How to find the first column of this matrix ( $\mathbf{s}$ ) from the filter coefficients ( $\mathbf{t}$ ), and vice versa.
- The compact filter notation for filters with a finite number of filter coefficients.
- The definition of convolution, its connection with filters, and the `conv`-function for computing convolution.
- Connection between applying a filter and multiplying polynomials.
- The formal definition of a digital filter in terms of having the Fourier vectors as eigenvectors.
- The definition of the vector frequency response in terms of the corresponding eigenvalues.
- The definition of time-invariance and the three equivalent characterizations of a filter.
- For filters, eigenvalues can be computed by taking the DFT of the first column  $\mathbf{s}$ , and there is no need to compute eigenvectors explicitly.
- How to apply a digital filter to a sum of sines or cosines, by splitting these into a sum of eigenvectors.

- The definition of the continuous frequency response in terms of the filter coefficients  $t$ .
- Connection with the vector frequency response.
- Properties of the continuous frequency response, in particular that the product of two frequency responses equals the frequency response of the product.
- How to compute the frequency response of the product of two filters,.
- How to find the filter coefficients when the continuous frequency response is known.
- Simple examples of filters, such as time delay filters and filters which add echo.
- Low-pass and high-pass filters and their frequency responses, and their interpretation as treble- and bass-reducing filters. Moving average filters, and filters arising from rows in Pascal's triangle, as examples of such filters.
- How to pass between low-pass and high-pass filters by adding an alternating sign to the filter coefficients.

## Chapter 4

# Symmetric filters and the DCT

In Chapter 1 we approximated a signal of finite duration with trigonometric functions. Since these are all periodic, there are some undesirable effects near the boundaries of the signal (at least when the values at the boundaries are different), and this resulted in a slowly converging Fourier series. This was addressed by instead considering the symmetric extension of the function, for which we obtained a more precise Fourier representation, as fewer Fourier basis vectors were needed in order to get a precise approximation.

This chapter is dedicated to addressing these thoughts for vectors. We will start by defining symmetric extensions of vectors, similarly to how we defined these for functions. Just as the Fourier series of a symmetric function was a cosine series, we will see that the symmetric extension can be viewed as a cosine vector. This gives rise to a different change of coordinates than the DFT, which we will call the DCT, which enables us to express a symmetric vector as a sum of cosine-vectors (instead of the non-symmetric complex exponentials). Since a cosine also can be associated with a given frequency, the DCT is otherwise similar to the DFT, in that it extracts the frequency information in the vector. The advantage is that the DCT can give more precise frequency information than the DFT, since it avoids the discontinuity problem of the Fourier series. This makes the DCT very practical for applications, and we will explain some of these applications. We will also show that the DCT has a very efficient implementation, comparable with the FFT.

In this chapter we will also see that the DCT has a very similar role as the DFT when it comes to filters: just as the DFT diagonalized filters, we will see that symmetric filters can be diagonalized by the DCT, when we apply the filter to the symmetric extension of the input. We will actually show that the filters which preserve our symmetric extensions are exactly the symmetric filters.

## 4.1 Symmetric vectors and the DCT

As in Chapter 1, vectors can also be extended in a symmetric manner, besides the simple periodic extension procedure from Figure 2.1. In Figure 4.1 we have shown such an extension of a vector  $\mathbf{x}$ . It has  $\mathbf{x}$  as its first half, and a copy of  $\mathbf{x}$  in reverse order as its second half.

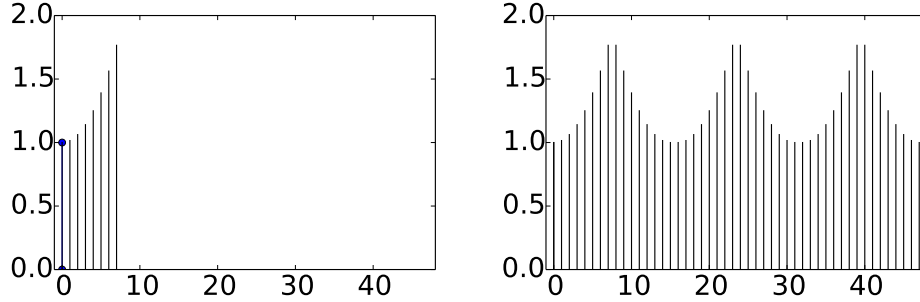


Figure 4.1: A vector and its symmetric extension.

We will call this the symmetric extension of  $\mathbf{x}$ :

**Definition 4.1.** *Symmetric extension of a vector.*

By the *symmetric extension* of  $\mathbf{x} \in \mathbb{R}^N$ , we mean the symmetric vector  $\check{\mathbf{x}} \in \mathbb{R}^{2N}$  defined by

$$\check{x}_k = \begin{cases} x_k & 0 \leq k < N \\ x_{2N-1-k} & N \leq k < 2N-1 \end{cases} \quad (4.1)$$

Clearly, the symmetric extension is symmetric around  $N - 1/2$ . This is not the only way to construct a symmetric extension, as we will return to later. As shown in Figure 4.1, but not included in Definition 4.1, we also repeat  $\check{\mathbf{x}} \in \mathbb{R}^{2N}$  in order to obtain a periodic vector. Creating a symmetric extension is thus a two-step process:

- First, “mirror” the vector to obtain a vector in  $\mathbb{R}^{2N}$ ,
- repeat this periodically to obtain a periodic vector.

The result from the first step lies in an  $N$ -dimensional subspace of all vectors in  $\mathbb{R}^{2N}$ , which we will call *the space of symmetric vectors*. To account for the fact that a periodic vector can have a different symmetry point than  $N - 1/2$ , let us make the following general definition:

**Definition 4.2.** *Symmetric vector.*

We say that a periodic vector  $\mathbf{x}$  is *symmetric* if there exists a number  $d$  so that  $x_{d+k} = x_{d-k}$  for all  $k$  so that  $d+k$  and  $d-k$  are integers.  $d$  is called the *symmetry point* of  $\mathbf{x}$ .

Due to the inherent periodicity of  $\mathbf{x}$ , it is clear that  $N$  must be an even number for symmetric vectors to exist at all.  $d$  can take any value, and it may not be an integer: It can also be an odd multiple of  $1/2$ , because then both  $d+k$  and  $d-k$  are integers when  $k$  also is an odd multiple of  $1/2$ . The symmetry point in symmetric extensions as defined in Definition 4.1 was  $d = N - 1/2$ . This is very common in the literature, and this is why we concentrate on this in this chapter. Later we will also consider symmetry around  $N - 1$ , as this also is much used.

We would like to find a basis for the  $N$ -dimensional space of symmetric vectors, and we would like this basis to be similar to the Fourier basis. Since the Fourier basis corresponds to the standard basis in the frequency domain, we are lead to studying the DFT of a symmetric vector. If the symmetry point is an integer, it is straightforward to prove the following:

**Theorem 4.3.** *Symmetric vectors with integer symmetry points.*

Let  $d$  be an integer. The following are equivalent

- $\mathbf{x}$  is real and symmetric with  $d$  as symmetry point.
- $(\hat{\mathbf{x}})_n = z_n e^{-2\pi idn/N}$  where  $z_n$  are real numbers so that  $z_n = z_{N-n}$ .

*Proof.* Assume first that  $d = 0$ . It follows in this case from property 2a) of Theorem 2.7 that  $(\hat{\mathbf{x}})_n$  is a real vector. Combining this with property 1 of Theorem 2.7 we see that  $\hat{\mathbf{x}}$ , just as  $\mathbf{x}$ , also must be a real vector symmetric about 0. Since the DFT is one-to-one, it follows that  $\mathbf{x}$  is real and symmetric about 0 if and only if  $\hat{\mathbf{x}}$  is. From property 3 of Theorem 2.7 it follows that, when  $d$  is an integer,  $\mathbf{x}$  is real and symmetric about  $d$  if and only if  $(\hat{\mathbf{x}})_n = z_n e^{-2\pi idn/N}$ , where  $z_n$  is real and symmetric about 0. This completes the proof.  $\square$

Symmetric extensions were here defined by having the non-integer symmetry point  $N - 1/2$ , however. For these we prove the following, which is slightly more difficult.

**Theorem 4.4.** *Symmetric vectors with non-integer symmetry points.*

Let  $d$  be an odd multiple of  $1/2$ . The following are equivalent

- $\mathbf{x}$  is real and symmetric with  $d$  as symmetry point.
- $(\hat{\mathbf{x}})_n = z_n e^{-2\pi idn/N}$  where  $z_n$  are real numbers so that  $z_{N-n} = -z_n$ .

*Proof.* When  $\mathbf{x}$  is as stated we can write



$$\begin{aligned}
(\hat{\mathbf{x}})_n &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-2\pi i k n / N} \\
&= \frac{1}{\sqrt{N}} \left( \sum_{s \geq 0} x_{d+s} e^{-2\pi i (d+s)n / N} + \sum_{s \geq 0} x_{d-s} e^{-2\pi i (d-s)n / N} \right) \\
&= \frac{1}{\sqrt{N}} \sum_{s \geq 0} x_{d+s} \left( e^{-2\pi i (d+s)n / N} + e^{-2\pi i (d-s)n / N} \right) \\
&= \frac{1}{\sqrt{N}} e^{-2\pi i d n / N} \sum_{s \geq 0} x_{d+s} \left( e^{-2\pi i s n / N} + e^{2\pi i s n / N} \right) \\
&= \frac{1}{\sqrt{N}} e^{-2\pi i d n / N} \sum_{s \geq 0} 2x_{d+s} \cos(2\pi s n / N).
\end{aligned}$$

Here  $s$  runs through odd multiples of  $1/2$ . Since  $z_n = \frac{1}{\sqrt{N}} \sum_{s \geq 0} 2x_{d+s} \cos(2\pi s n / N)$  is a real number, we can write the result as  $z_n e^{-2\pi i d n / N}$ . Substituting  $N - n$  for  $n$ , we get

$$\begin{aligned}
(\hat{\mathbf{x}})_{N-n} &= \frac{1}{\sqrt{N}} e^{-2\pi i d (N-n) / N} \sum_{s \geq 0} 2x_{d+s} \cos(2\pi s (N-n) / N) \\
&= \frac{1}{\sqrt{N}} e^{-2\pi i d (N-n) / N} \sum_{s \geq 0} 2x_{d+s} \cos(-2\pi s n / N + 2\pi s) \\
&= -\frac{1}{\sqrt{N}} e^{-2\pi i d (N-n) / N} \sum_{s \geq 0} 2x_{d+s} \cos(2\pi s n / N) = -z_n e^{-2\pi i d (N-n) / N}.
\end{aligned}$$

This shows that  $z_{N-n} = -z_n$ , and this completes one way of the proof. The other way, we can write

$$x_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} (\hat{\mathbf{x}})_n e^{2\pi i k n / N}$$

if  $(\hat{\mathbf{x}})_n = z_n e^{-2\pi i d n / N}$  and  $(\hat{\mathbf{x}})_{N-n} = -z_n e^{-2\pi i d (N-n) / N}$ , the sum of the  $n$ 'th term and the  $N - n$ 'th term in the sum is

$$\begin{aligned}
&z_n e^{-2\pi i d n / N} e^{2\pi i k n / N} - z_n e^{-2\pi i d (N-n) / N} e^{2\pi i k (N-n) / N} \\
&= z_n (e^{2\pi i (k-d)n / N} - e^{-2\pi i d + 2\pi i d n / N - 2\pi i k n / N}) \\
&= z_n (e^{2\pi i (k-d)n / N} + e^{2\pi i (d-k)n / N}) = 2z_n \cos(2\pi (k-d)n / N).
\end{aligned}$$

This is real, so that all  $x_k$  are real. If we set  $k = d + s$ ,  $k = d - s$  here we get

$$\begin{aligned} 2z_n \cos(2\pi((d+s)-d)n/N) &= 2z_n \cos(2\pi sn/N) \\ 2z_n \cos(2\pi((d-s)-d)n/N) &= 2z_n \cos(-2\pi sn/N) = 2z_n \cos(2\pi sn/N). \end{aligned}$$

By adding terms together and comparing we must have that  $x_{d+s} = x_{d-s}$ , and the proof is done.  $\square$

Now, let us specialize to symmetric extensions as defined in Definition 4.1, i.e. where  $d = N - 1/2$ . The following result gives us an orthonormal basis for the symmetric extensions, which are very simple in the frequency domain:

**Theorem 4.5.** *Orthonormal basis for symmetric vectors.*

The set of all  $\mathbf{x}$  symmetric around  $N - 1/2$  is a vector space of dimension  $N$ , and we have that

$$\left\{ \mathbf{e}_0, \left\{ \frac{1}{\sqrt{2}} \left( e^{\pi in/(2N)} \mathbf{e}_n + e^{-\pi in/(2N)} \mathbf{e}_{2N-n} \right) \right\}_{n=1}^{N-1} \right\}$$

is an orthonormal basis for  $\hat{\mathbf{x}}$  where  $\mathbf{x}$  is symmetric around  $N - 1/2$ .

*Proof.* For a vector  $\mathbf{x}$  symmetric about  $d = N - 1/2$  we know that

$$(\hat{\mathbf{x}})_n = z_n e^{-2\pi i(N-1/2)n/(2N)},$$

and the only requirement on the vector  $\mathbf{z}$  is the antisymmetry condition  $z_{2N-n} = -z_n$ . The vectors  $\mathbf{z}_i = \frac{1}{\sqrt{2}}(\mathbf{e}_i - \mathbf{e}_{2N-i})$ ,  $1 \leq i \leq N - 1$ , together with the vector  $\mathbf{z}_0 = \mathbf{e}_0$ , are clearly orthonormal and satisfies the antisymmetry condition. From these we obtain that

$$\left\{ \mathbf{e}_0, \left\{ \frac{1}{\sqrt{2}} \left( e^{-2\pi i(N-1/2)n/(2N)} \mathbf{e}_n - e^{-2\pi i(N-1/2)(2N-n)/(2N)} \mathbf{e}_{2N-n} \right) \right\}_{n=1}^{N-1} \right\}$$

is an orthonormal basis for the  $\hat{\mathbf{x}}$  with  $\mathbf{x}$  symmetric. We can write

$$\begin{aligned} & \frac{1}{\sqrt{2}} \left( e^{-2\pi i(N-1/2)n/(2N)} \mathbf{e}_n - e^{-2\pi i(N-1/2)(2N-n)/(2N)} \mathbf{e}_{2N-n} \right) \\ &= \frac{1}{\sqrt{2}} \left( e^{-\pi in} e^{\pi in/(2N)} \mathbf{e}_n + e^{\pi in} e^{-\pi in/(2N)} \mathbf{e}_{2N-n} \right) \\ &= \frac{1}{\sqrt{2}} e^{\pi in} \left( e^{\pi in/(2N)} \mathbf{e}_n + e^{-\pi in/(2N)} \mathbf{e}_{2N-n} \right). \end{aligned}$$

This also means that

$$\left\{ \mathbf{e}_0, \left\{ \frac{1}{\sqrt{2}} \left( e^{\pi in/(2N)} \mathbf{e}_n + e^{-\pi in/(2N)} \mathbf{e}_{2N-n} \right) \right\}_{n=1}^{N-1} \right\}$$

is an orthonormal basis.  $\square$

We immediately get the following result:

**Theorem 4.6.** *Orthonormal basis for symmetric vectors.*

We have that

$$\left\{ \frac{1}{\sqrt{2N}} \cos \left( 2\pi \frac{0}{2N} \left( k + \frac{1}{2} \right) \right), \left\{ \frac{1}{\sqrt{N}} \cos \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right) \right\}_{n=1}^{N-1} \right\} \quad (4.2)$$

is an orthonormal basis for the set of vectors symmetric around  $N - 1/2$  in  $\mathbb{R}^{2N}$ . Moreover, the  $n$ 'th vector in this basis has frequency contribution only from the indices  $n$  and  $2N - n$ .

*Proof.* Since the IDFT is unitary, the IDFT applied to the vectors above gives an orthonormal basis for the set of symmetric extensions. We get that

$$(F_{2N})^H(\mathbf{e}_0) = \left( \frac{1}{\sqrt{2N}}, \frac{1}{\sqrt{2N}}, \dots, \frac{1}{\sqrt{2N}} \right) = \frac{1}{\sqrt{2N}} \cos \left( 2\pi \frac{0}{2N} \left( k + \frac{1}{2} \right) \right).$$

We also get that

$$\begin{aligned} & (F_{2N})^H \left( \frac{1}{\sqrt{2}} \left( e^{\pi i n / (2N)} \mathbf{e}_n + e^{-\pi i n / (2N)} \mathbf{e}_{2N-n} \right) \right) \\ &= \frac{1}{\sqrt{2}} \left( e^{\pi i n / (2N)} \frac{1}{\sqrt{2N}} e^{2\pi i n k / (2N)} + e^{-\pi i n / (2N)} \frac{1}{\sqrt{2N}} e^{2\pi i (2N-n)k / (2N)} \right) \\ &= \frac{1}{\sqrt{2}} \left( e^{\pi i n / (2N)} \frac{1}{\sqrt{2N}} e^{2\pi i n k / (2N)} + e^{-\pi i n / (2N)} \frac{1}{\sqrt{2N}} e^{-2\pi i n k / (2N)} \right) \\ &= \frac{1}{2\sqrt{N}} \left( e^{2\pi i (n/(2N))(k+1/2)} + e^{-2\pi i (n/(2N))(k+1/2)} \right) = \frac{1}{\sqrt{N}} \cos \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right). \end{aligned}$$

Since  $F_{2N}$  is unitary, and thus preserves the scalar product, the given vectors are orthonormal.  $\square$

We need to address one final thing before we can define the DCT: The vector  $\mathbf{x}$  we start with is in  $\mathbb{R}^N$ , but the vectors above are in  $\mathbb{R}^{2N}$ . We would like to have orthonormal vectors in  $\mathbb{R}^N$ , so that we can use them to decompose  $\mathbf{x}$ . It is possible to show with a direct argument that, when we restrict the vectors above to the first  $N$  elements, they are still orthogonal. We will, however, apply a more instructive argument to show this, which gives us some intuition into the connection with symmetric filters. We start with the following result, which shows that a filter preserves symmetric vectors if and only if the filter is symmetric.

**Theorem 4.7.** *Criteria for preserving symmetric vectors.*

Let  $S$  be a filter. The following are equivalent

- $S$  preserves symmetric vectors (i.e.  $S\mathbf{x}$  is a symmetric vector whenever  $\mathbf{x}$  is).
- The set of filter coefficients of  $S$  is a symmetric vector.

Also, when  $S$  preserves symmetric vectors, the following hold:

- The vector of filter coefficients has an integer symmetry point if and only if the input and output have the same type (integer or non-integer) of symmetry point.
- The input and output have the same symmetry point if and only if the filter is symmetric.

*Proof.* Assume that the filter  $S$  maps a symmetric vector with symmetry at  $d_1$  to another symmetric vector. Let  $\mathbf{x}$  be the symmetric vector so that  $(\hat{\mathbf{x}})_n = e^{-2\pi i d_1 n/N}$  for  $n < N/2$ . Since the output is a symmetric vector, we must have that

$$\lambda_{S,n} e^{-2\pi i d_1 n/N} = z_n e^{-2\pi i d_2 n/N}$$

for some  $d_2, z_n$  and for  $n < N/2$ . But this means that  $\lambda_{S,n} = y_n e^{-2\pi i (d_2 - d_1) n/N}$ . Similar reasoning applies for  $n > N/2$ , so that  $\lambda_{S,n}$  clearly equals  $\hat{s}$  for some symmetric vector  $\mathbf{s}$  from Theorems 4.3 and 4.4. This vector equals (up to multiplication with  $\sqrt{N}$ ) the filter coefficients of  $S$ , which therefore is a symmetric. Moreover, it is clear that the filter coefficients have an integer symmetry point if and only if the input and output vector either both have an integer symmetry point, or both a non-integer symmetry point.  $\square$

Since the filter coefficients of a filter which preserves symmetric vectors also is a symmetric vector, this means that its frequency response takes the form  $\lambda_{S,n} = z_n e^{-2\pi i d n/N}$ , where  $\mathbf{z}$  is a real vector. This means that the phase (argument) of the frequency response is  $-2\pi d n/N$  or  $\pi - 2\pi d n/N$ , depending on the sign of  $z_n$ . In other words, the phase is linear in  $n$ . Filters which preserve symmetric vectors are therefore also called *linear phase filters*.

Note also that the case  $d = 0$  or  $d = N - 1/2$  corresponds to symmetric filters. An example of linear phase filters which are not symmetric are smoothing filters where the coefficients are taken from odd rows in Pascal's triangle.

When  $S$  is symmetric, it preserves symmetric extensions, so that it makes sense to restrict  $S$  to symmetric vectors. We therefore make the following definition.

**Definition 4.8.** *Symmetric restriction.*

Assume that  $S : \mathbb{R}^{2N} \rightarrow \mathbb{R}^{2N}$  is a symmetric filter. We define  $S_r : \mathbb{R}^N \rightarrow \mathbb{R}^N$  as the mapping which sends  $\mathbf{x} \in \mathbb{R}^N$  to the first  $N$  components of the vector  $S\check{\mathbf{x}}$ .  $S_r$  is also called the symmetric restriction of  $S$ .

$S_r$  is clearly linear, and the restriction of  $S$  to vectors symmetric about  $N - 1/2$  is characterized by  $S_r$ . We continue with the following result:

**Theorem 4.9.** *Expression for  $S_r$ .*

Assume that  $S : \mathbb{R}^{2N} \rightarrow \mathbb{R}^{2N}$  is a symmetric filter, and that

$$S = \begin{pmatrix} S_1 & S_2 \\ S_3 & S_4 \end{pmatrix}.$$

Then  $S_r$  is symmetric, and  $S_r = S_1 + (S_2)^f$ , where  $(S_2)^f$  is the matrix  $S_2$  with the columns reversed.

*Proof.* With  $S$  as in the text of the theorem, we compute

$$\begin{aligned} S_r \mathbf{x} &= (S_1 \quad S_2) \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \\ x_{N-1} \\ \vdots \\ x_0 \end{pmatrix} = S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + S_2 \begin{pmatrix} x_{N-1} \\ \vdots \\ x_0 \end{pmatrix} \\ &= S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + (S_2)^f \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} = (S_1 + (S_2)^f) \mathbf{x}, \end{aligned}$$

so that  $S_r = S_1 + (S_2)^f$ . Since  $S$  is symmetric,  $S_1$  is also symmetric.  $(S_2)^f$  is also symmetric, since it is constant on anti-diagonals. It follows then that  $S$  is also symmetric. This completes the proof.  $\square$

Note that  $S_r$  is not a digital filter, since its matrix is not circulant. In particular, its eigenvectors are not pure tones. In the block matrix factorization of  $S$ ,  $S_2$  contains the circulant part of the matrix, and forming  $(S_2)^f$  means that the circulant parts switch corners. With the help of Theorem 4.9 we can finally establish the orthogonality of the cosine-vectors in  $\mathbb{R}^N$ .

**Corollary 4.10.** *Basis of eigenvectors for  $S_r$ .*

Let  $S$  be a symmetric filter, and let  $S_r$  be the mapping defined in Theorem 4.9. Define

$$d_{n,N} = \begin{cases} \sqrt{\frac{1}{N}} & , n = 0 \\ \sqrt{\frac{2}{N}} & , 1 \leq n < N \end{cases}$$

and  $\mathbf{d}_n = d_{n,N} \cos\left(2\pi \frac{n}{2N} \left(k + \frac{1}{2}\right)\right)$  for  $0 \leq n \leq N - 1$ , then  $\{\mathbf{d}_0, \mathbf{d}_1, \dots, \mathbf{d}_{N-1}\}$  is an orthonormal basis of eigenvectors for  $S_r$ .

*Proof.* Let  $S$  be a symmetric filter of length  $2N$ . We know then that  $\lambda_{S,n} = \lambda_{S,2N-n}$ , so that

$$\begin{aligned}
& S\left(\cos\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right)\right) \\
&= S\left(\frac{1}{2}\left(e^{2\pi i(n/(2N))(k+1/2)} + e^{-2\pi i(n/(2N))(k+1/2)}\right)\right) \\
&= \frac{1}{2}\left(e^{\pi in/(2N)}S\left(e^{2\pi ink/(2N)}\right) + e^{-\pi in/(2N)}S\left(e^{-2\pi ink/(2N)}\right)\right) \\
&= \frac{1}{2}\left(e^{\pi in/(2N)}\lambda_{S,n}e^{2\pi ink/(2N)} + e^{-\pi in/(2N)}\lambda_{S,2N-n}e^{-2\pi ink/(2N)}\right) \\
&= \frac{1}{2}\left(\lambda_{S,n}e^{2\pi i(n/(2N))(k+1/2)} + \lambda_{S,2N-n}e^{-2\pi i(n/(2N))(k+1/2)}\right) \\
&= \lambda_{S,n}\frac{1}{2}\left(e^{2\pi i(n/(2N))(k+1/2)} + e^{-2\pi i(n/(2N))(k+1/2)}\right) \\
&= \lambda_{S,n}\cos\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right),
\end{aligned}$$

where we have used that  $e^{2\pi ink/(2N)}$  is an eigenvector of  $S$  with eigenvalue  $\lambda_{S,n}$ , and  $e^{-2\pi ink/(2N)} = e^{2\pi i(2N-n)k/(2N)}$  is an eigenvector of  $S$  with eigenvalue  $\lambda_{S,2N-n}$ . This shows that the vectors are eigenvectors for symmetric filters of length  $2N$ . It is also clear that the first half of the vectors must be eigenvectors for  $S_r$  with the same eigenvalue, since when  $\mathbf{y} = S\mathbf{x} = \lambda_{S,n}\mathbf{x}$ , we also have that

$$(y_0, y_1, \dots, y_{N-1}) = S_r(x_0, x_1, \dots, x_{N-1}) = \lambda_{S,n}(x_0, x_1, \dots, x_{N-1}).$$

To see why these vectors are orthogonal, choose at the outset a symmetric filter where  $\{\lambda_{S,n}\}_{n=0}^{N-1}$  are distinct. Then the cosine-vectors of length  $N$  are also eigenvectors with distinct eigenvalues, and they must be orthogonal since  $S_r$  is symmetric. Moreover, since

$$\begin{aligned}
& \sum_{k=0}^{2N-1} \cos^2 \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right) \\
&= \sum_{k=0}^{N-1} \cos^2 \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right) + \sum_{k=N}^{2N-1} \cos^2 \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right) \\
&= \sum_{k=0}^{N-1} \cos^2 \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right) + \sum_{k=0}^{N-1} \cos^2 \left( 2\pi \frac{n}{2N} \left( k + N + \frac{1}{2} \right) \right) \\
&= \sum_{k=0}^{N-1} \cos^2 \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right) + (-1)^{2n} \sum_{k=0}^{N-1} \cos^2 \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right) \\
&= 2 \sum_{k=0}^{N-1} \cos^2 \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right),
\end{aligned}$$

where we used that  $\cos(x + n\pi) = (-1)^n \cos x$ . This means that

$$\left\| \left\{ \cos \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right) \right\}_{k=0}^{2N-1} \right\| = \sqrt{2} \left\| \left\{ \cos \left( 2\pi \frac{n}{2N} \left( k + \frac{1}{2} \right) \right) \right\}_{k=0}^{N-1} \right\|.$$

Thus, in order to make the vectors orthonormal when we consider the first  $N$  elements instead of all  $2N$  elements, we need to multiply with  $\sqrt{2}$ . This gives us the vectors  $\mathbf{d}_n$  as defined in the text of the theorem. This completes the proof.  $\square$

We now clearly see the analogy between symmetric functions and vectors: while the first can be written as a sum of cosine-functions, the second can be written as a sum of cosine-vectors. The orthogonal basis we have found is given its own name:

**Definition 4.11.** *DCT basis.*

We denote by  $\mathcal{D}_N$  the orthogonal basis  $\{\mathbf{d}_0, \mathbf{d}_1, \dots, \mathbf{d}_{N-1}\}$ . We also call  $\mathcal{D}_N$  the  $N$ -point *DCT* basis.

Using the DCT basis instead of the Fourier basis we can make the following definitions, which parallel those for the DFT:

**Definition 4.12.** *Discrete Cosine Transform.*

The change of coordinates from the standard basis of  $\mathbb{R}^N$  to the DCT basis  $\mathcal{D}_N$  is called the *discrete cosine transform* (or DCT). The  $N \times N$  matrix  $\text{DCT}_N$  that represents this change of basis is called the ( $N$ -point) DCT matrix. If  $\mathbf{x}$  is a vector in  $\mathbb{R}^N$ , its coordinates  $\mathbf{y} = (y_0, y_1, \dots, y_{N-1})$  relative to the DCT basis are called the DCT coefficients of  $\mathbf{x}$  (in other words,  $\mathbf{y} = \text{DCT}_N \mathbf{x}$ ).

Note that we can also write

$$\text{DCT}_N = \sqrt{\frac{2}{N}} \begin{pmatrix} 1/\sqrt{2} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} (\cos(2\pi \frac{n}{2N}(k+1/2))). \quad (4.3)$$

Since this matrix is orthogonal, it is immediate that

$$\begin{aligned} (\cos(2\pi \frac{n}{2N}(k+1/2)))^{-1} &= \frac{2}{N} (\cos(2\pi \frac{n+1/2}{2N}k)) \begin{pmatrix} 1/2 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \quad (4.4) \\ (\cos(2\pi \frac{n+1/2}{2N}k))^{-1} &= \frac{2}{N} \begin{pmatrix} 1/2 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} (\cos(2\pi \frac{n}{2N}(k+1/2))). \end{aligned} \quad (4.5)$$

In other words, not only can  $\text{DCT}_N$  be directly expressed in terms of a cosine-matrix, but our developments helped us to express the inverse of a cosine matrix in terms of other cosine-matrices. In the literature different types of cosine-matrices have been useful:

- I** Cosine-matrices with entries  $\cos(2\pi nk/(2(N-1)))$ .
- II** Cosine-matrices with entries  $\cos(2\pi n(k+1/2)/(2N))$ .
- III** Cosine-matrices with entries  $\cos(2\pi(n+1/2)k/(2N))$ .
- IV** Cosine-matrices with entries  $\cos(2\pi(n+1/2)(k+1/2)/(2N))$ .

We will call these type-I, type-II, type-III, and type-IV cosine-matrices, respectively. What we did above handles the case of type-II cosine-matrices. It will turn out that not all of these cosine-matrices are orthogonal, but that we in all cases, as we did above for type-II cosine matrices, can express the inverse of a cosine-matrix of one type in terms of a cosine-matrix of another type, and that any cosine-matrix is easily expressed in terms of an orthogonal matrix. These orthogonal matrices will be called  $\text{DCT}_N^{(I)}$ ,  $\text{DCT}_N^{(II)}$ ,  $\text{DCT}_N^{(III)}$ , and  $\text{DCT}_N^{(IV)}$ , respectively, and they are all called DCT-matrices. The  $\text{DCT}_N$  we constructed above is thus  $\text{DCT}_N^{(II)}$ . The type-II DCT matrix is the most commonly used, and the type is therefore often dropped when referring to these. We will consider the other cases of cosine-matrices at different places in this book: In the next chapter we will run into type-I cosine matrices, in connection with a different extension strategy used for wavelets. Type-IV cosine-matrices will be encountered in exercises 4.5 and 4.6 at the end of this section.



As with the Fourier basis vectors, the DCT basis vectors are called synthesis vectors, since we can write

$$\mathbf{x} = y_0 \mathbf{d}_0 + y_1 \mathbf{d}_1 + \cdots + y_{N-1} \mathbf{d}_{N-1} \quad (4.6)$$

in the same way as for the DFT. Following the same reasoning as for the DFT,  $\text{DCT}_N^{-1}$  is the matrix where the  $\mathbf{d}_n$  are columns. But since these vectors are real and orthonormal,  $\text{DCT}_N$  must be the matrix where the  $\mathbf{d}_n$  are rows. Moreover, since Theorem 4.9 also states that the same vectors are eigenvectors for filters which preserve symmetric extensions, we can state the following:

**Theorem 4.13.** *The DCT is orthogonal.*

$\text{DCT}_N$  is the orthogonal matrix where the rows are  $\mathbf{d}_n$ . Moreover, for any digital filter  $S$  which preserves symmetric extensions,  $(\text{DCT}_N)^T$  diagonalizes  $S_r$ , i.e.  $S_r = \text{DCT}_N^T D \text{DCT}_N$  where  $D$  is a diagonal matrix.

Let us also make the following definition:

**Definition 4.14.** *IDCT.*

We will call  $\mathbf{x} = (\text{DCT}_N)^T \mathbf{y}$  the inverse DCT or (IDCT) of  $\mathbf{x}$ .

Matlab's functions for computing the DCT and IDCT are called `dct`, and `idct`, respectively. These are defined exactly as they are here, contrary to the case for the FFT (where a different normalizing factor was used).

With these functions we can repeat examples 2.16- 2.18, by simply replacing the calls to `DFTImpl` with calls to the DCT counterparts. You may not here much improvements in these simple experiments, but in theory the DCT should be able to approximate sound better.

Similarly to the DFT, one can think of the DCT as a least squares approximation and the unique representation of a function having the same sample values, but this time in terms of sinusoids instead of complex exponentials:

**Theorem 4.15.** *Interpolation with the DCT basis.*

Let  $f$  be a function defined on the interval  $[0, T]$ , and let  $\mathbf{x}$  be the sampled vector given by

$$x_k = f((2k+1)T/(2N)) \quad \text{for } k = 0, 1, \dots, N-1.$$

There is exactly one linear combination  $g(t)$  on the form

$$\sum_{n=0}^{N-1} y_n d_{n,N} \cos(2\pi(n/2)t/T)$$

which satisfies the conditions

$$g((2k+1)T/(2N)) = f((2k+1)T/(2N)), \quad k = 0, 1, \dots, N-1,$$

and its coefficients are determined by  $\mathbf{y} = \text{DCT}_N \mathbf{x}$ .

*Proof.* This follows by inserting  $t = (2k + 1)T/(2N)$  in the equation

$$g(t) = \sum_{n=0}^{N-1} y_n d_{n,N} \cos(2\pi(n/2)t/T)$$

to arrive at the equations

$$f(kT/N) = \sum_{n=0}^{N-1} y_n d_{n,N} \cos\left(2\pi \frac{n}{2N} \left(k + \frac{1}{2}\right)\right) \quad 0 \leq k \leq N - 1.$$

This gives us an equation system for finding the  $y_n$  with the invertible DCT matrix as coefficient matrix, and the result follows.  $\square$

Due to this there is a slight difference to how we applied the DFT, due to the subtle change in the sample points, from  $kT/N$  for the DFT, to  $(2k + 1)T/(2N)$  for the DCT. The sample points for the DCT are thus the midpoints on the intervals in a uniform partition of  $[0, T]$  into  $N$  intervals, while they for the DFT are the start points on the intervals. Also, the frequencies are divided by 2. In Figure 4.2 we have plotted the sinusoids of Theorem 4.15 for  $T = 1$ , as well as the sample points used in that theorem.

The sample points in the upper left plot correspond to the first column in the DCT matrix, the sample points in the upper right plot to the second column of the DCT matrix, and so on (up to normalization with  $d_{n,N}$ ). As  $n$  increases, the functions oscillate more and more. As an example,  $y_5$  says how much content of maximum oscillation there is. In other words, the DCT of an audio signal shows the proportion of the different frequencies in the signal, and the two formulas  $\mathbf{y} = \text{DCT}_N \mathbf{x}$  and  $\mathbf{x} = (\text{DCT}_N)^T \mathbf{y}$  allow us to switch back and forth between the time domain representation and the frequency domain representation of the sound. In other words, once we have computed  $\mathbf{y} = \text{DCT}_N \mathbf{x}$ , we can analyse the frequency content of  $\mathbf{x}$ . If we want to reduce the bass we can decrease the  $\mathbf{y}$ -values with small indices and if we want to increase the treble we can increase the  $\mathbf{y}$ -values with large indices.

### Example 4.1: Computing lower order DCTs

As with Example 2.3, exact expressions for the DCT can be written down just for a few specific cases. It turns out that the case  $N = 4$  as considered in Example 2.3 does not give the same type of nice, exact values, so let us instead consider the case  $N = 2$ . We have that

$$\text{DCT}_4 = \begin{pmatrix} \frac{1}{\sqrt{2}} \cos(0) & \frac{1}{\sqrt{2}} \cos(0) \\ \cos\left(\frac{\pi}{2} \left(0 + \frac{1}{2}\right)\right) & \cos\left(\frac{\pi}{2} \left(1 + \frac{1}{2}\right)\right) \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

The DCT of the same vector as in Example 2.3 can now be computed as:

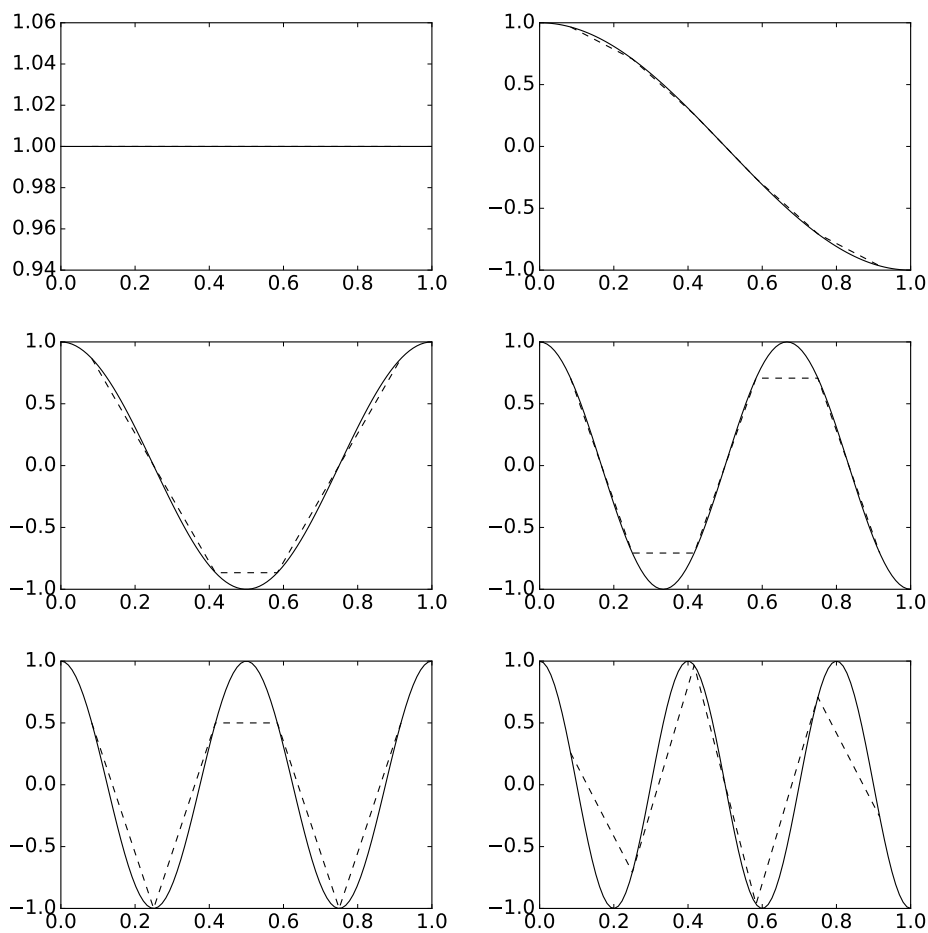


Figure 4.2: The 6 different sinusoids used in DCT for  $N = 6$ , i.e.  $\cos(2\pi(n/2)t)$ ,  $0 \leq n < 6$ . The plots also show piecewise linear functions (in red) between the sample points  $\frac{2k+1}{2N}$   $0 \leq k < 6$ , since only the values at these points are used in Theorem 4.15.

$$\text{DCT}_2 \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} \frac{3}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}.$$

### Exercise 4.2: Computing eigenvalues

Consider the matrix

$$S = \frac{1}{3} \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

- a) Compute the eigenvalues and eigenvectors of  $S$  using the results of this section. You should only need to perform one DFT or one DCT in order to achieve this.
- b) Use a computer to compute the eigenvectors and eigenvalues of  $S$  also. What are the differences from what you found in a)?
- c) Find a filter  $T$  so that  $S = T_r$ . What kind of filter is  $T$ ?

### Exercise 4.3: Writing down lower order $S_r$

Consider the averaging filter  $S = \{\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\}$ . Write down the matrix  $S_r$  for the case when  $N = 4$ .

### Exercise 4.4: Writing down lower order DCTs

As in Example 4.1, state the exact cartesian form of the DCT matrix for the case  $N = 3$ .

### Exercise 4.5: DCT-IV

Show that the vectors  $\left\{ \cos \left( 2\pi \frac{n+\frac{1}{2}}{2N} \left( k + \frac{1}{2} \right) \right) \right\}_{n=0}^{N-1}$  in  $\mathbb{R}^N$  are orthogonal, with lengths  $\sqrt{N/2}$ . This means that the matrix with entries  $\sqrt{\frac{2}{N}} \cos \left( 2\pi \frac{n+\frac{1}{2}}{2N} \left( k + \frac{1}{2} \right) \right)$  is orthogonal. Since this matrix also is symmetric, it is its own inverse. This is the DCT-IV, which we denote by  $\text{DCT}_N^{(\text{IV})}$ . Although we will not consider this, the DCT-IV also has an efficient implementation.

**Hint.** Compare with the orthogonal vectors  $\mathbf{d}_n$ , used in the DCT.

### Exercise 4.6: MDCT

The MDCT is defined as the  $N \times (2N)$ -matrix  $M$  with elements  $M_{n,k} = \cos(2\pi(n+1/2)(k+1/2+N/2)/(2N))$ . This exercise will take you through the details of the transformation which corresponds to multiplication with this matrix. The MDCT is very useful, and is also used in the MP3 standard and in more recent standards.

a) Show that

$$M = \sqrt{\frac{N}{2}} \text{DCT}_N^{(\text{IV})} \begin{pmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{pmatrix}$$

where  $A$  and  $B$  are the  $(N/2) \times N$ -matrices

$$A = \begin{pmatrix} \cdots & \cdots & 0 & -1 & -1 & 0 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & -1 & \cdots & \cdots & \cdots & \cdots & -1 & 0 \\ -1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & -1 \end{pmatrix} = \begin{pmatrix} -I_{N/2}^f & -I_{N/2} \end{pmatrix}$$

$$B = \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & -1 \\ 0 & 1 & \cdots & \cdots & \cdots & \cdots & -1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & 0 & 1 & -1 & 0 & \cdots & \cdots \end{pmatrix} = \begin{pmatrix} I_{N/2} & -I_{N/2}^f \end{pmatrix}.$$

Due to this expression, any algorithm for the DCT-IV can be used to compute the MDCT.

b) The MDCT is not invertible, since it is not a square matrix. We will show here that it still can be used in connection with invertible transformations. We first define the IMDCT as the matrix  $M^T/N$ . Transposing the matrix expression we obtained in a) gives

$$\frac{1}{\sqrt{2N}} \begin{pmatrix} \mathbf{0} & B^T \\ A^T & \mathbf{0} \end{pmatrix} \text{DCT}_N^{(\text{IV})}$$

for the IMDCT, which thus also has an efficient implementation. Show that if

$$\mathbf{x}_0 = (x_0, \dots, x_{N-1}) \quad \mathbf{x}_1 = (x_N, \dots, x_{2N-1}) \quad \mathbf{x}_2 = (x_{2N}, \dots, x_{3N-1})$$

and

$$\mathbf{y}_{0,1} = M \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{pmatrix} \quad \mathbf{y}_{1,2} = M \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix}$$

(i.e. we compute two MDCT's where half of the data overlap), then

$$\mathbf{x}_1 = \{\text{IMDCT}(\mathbf{y}_{0,1})\}_{k=N}^{2N-1} + \{\text{IMDCT}(\mathbf{y}_{1,2})\}_{k=0}^{N-1}.$$

Even though the MDCT itself is not invertible, the input can still be recovered from overlapping MDCT's.

## 4.2 Improvements using the DCT for interpolation

Recall that, in Section 3.2.2, we explained how to approximate an analog filter from the samples. It turns out that, when an analog filter is symmetric, we can use symmetric extensions to create a better approximation from the samples.

Assume that  $s$  is an analog filter, and that we apply it to a general function  $f$ . Denote as before the symmetric extension of  $f$  by  $\check{f}$ . We start with the following observation, which follows from the continuity of  $s$ .

**Observation 4.16.** *Using symmetric extensions for approximations.*

Since  $(\check{f})_N$  is a better approximation to  $\check{f}$ , compared to what  $f_N$  is to  $f$ ,  $s((\check{f})_N)$  is a better approximation to  $s(\check{f})$ , compared to what  $s(f_N)$  is to  $s(f)$ .

Since  $s(\check{f})$  agrees with  $s(f)$  except near the boundaries, we can thus conclude that  $s((\check{f})_N)$  is a better approximation to  $s(f)$  than what  $s(f_N)$  is.

We have seen that the restriction of  $s$  to  $V_{M,T}$  is equivalent to an  $N \times N$  digital filter  $S$ , where  $N = 2M + 1$ . Let  $\mathbf{x}$  be the samples of  $f$ ,  $\check{\mathbf{x}}$  the samples of  $\check{f}$ . Turning around the fact that  $(\check{f})_N$  is a better approximation to  $\check{f}$ , compared to what  $f_N$  is to  $f$ , the following is clear.

**Observation 4.17.** *Using symmetric extensions for approximations.*

The samples  $\check{\mathbf{x}}$  are a better approximation to the samples of  $(\check{f})_N$ , than the samples  $\mathbf{x}$  are to the samples of  $f_N$ .

Now, let  $\mathbf{z} = S\mathbf{x}$ , and  $\check{\mathbf{z}} = S\check{\mathbf{x}}$ . The following is also clear from the preceding observation, due to continuity of the digital filter  $S$ .

**Observation 4.18.** *Using symmetric extensions for approximations.*

$\check{\mathbf{z}}$  is a better approximation to  $S(\text{samples of } (\check{f})_N) = \text{samples of } s((\check{f})_N)$ , than  $\mathbf{z}$  is to  $S(\text{samples of } f_N) = \text{samples of } s(f_N)$ .

Since by Observation 4.16  $s((\check{f})_N)$  is a better approximation to the output  $s(f)$ , we conclude that  $\check{\mathbf{z}}$  is a better approximation than  $\mathbf{z}$  to the samples of the output of the filter.

**Observation 4.19.** *Using symmetric extensions for approximations.*

$S\check{\mathbf{x}}$  is a better approximation to the samples of  $s(f)$  than  $S\mathbf{x}$  is ( $\mathbf{x}$  are the samples of  $f$ ).

Now, let us also bring in the assumption that  $s$  is symmetric. Then the corresponding digital filter  $S$  is also symmetric, and we know then that we can view its restriction to symmetric extensions in  $\mathbb{R}^{2N}$  in terms of the mapping  $S_r : \mathbb{R}^N \rightarrow \mathbb{R}^N$ . We can thus specialize Figure 3.3 to symmetric filters by adding the step of creating the symmetric extension, and replacing  $S$  with  $S_r$ . We have summarized these remarks in Figure 4.3. The DCT appears here, since we have used Theorem 4.15 to interpolate with the DCT basis, instead of the Fourier basis. Note that this also requires that the sampling is performed as required

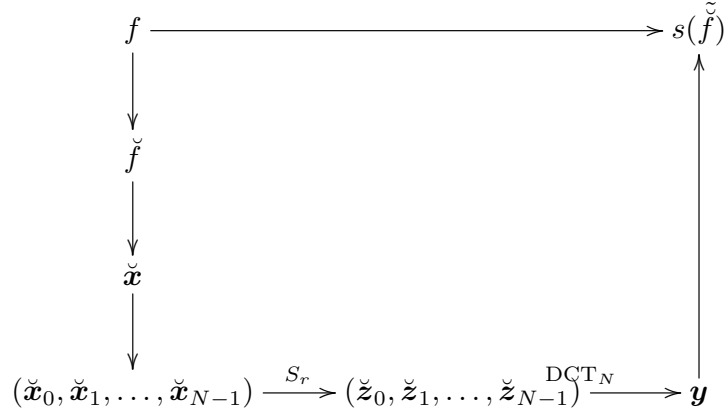


Figure 4.3: The connections between the new mapping  $S_r$ , sampling, and interpolation. The right vertical arrow represents interpolation with the DCT, i.e. that we compute  $\sum_{n=0}^{N-1} y_n d_{n,N} \cos(2\pi(n/2)t/T)$  for values of  $t$ .

in that theorem, i.e. the samples are the midpoints on all intervals. This new sampling procedure is not indicated in Figure 4.3.

Figure 4.3 can be further simplified to that shown in Figure 4.4.

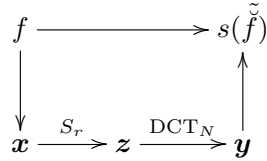


Figure 4.4: Simplification of Figure 4.3. The left vertical arrow represents sampling as dictated by the DCT.

Note that the assumption that  $s$  is symmetric only helped us to implement the approximation  $s(\tilde{f})$  more efficiently, since  $S_r$  has  $N$  points and  $S$  has  $2N$  points.  $s(\tilde{f})$  can in any way be used as an approximation, even if  $s$  is not symmetric, but the mapping does not preserve symmetry.

As mentioned in Section 3.2, interpolation of a function from its samples can be seen as a special case. This can thus be illustrated as in Figure 4.5.

Note that the approximation lies in  $V_{2M,2T}$  (i.e. it is in a higher order Fourier space), but the point is that the same number of samples is used.

### 4.2.1 Implementations of symmetric filters

Symmetric filters are also important for applications since they can be implemented efficiently. To see this, we can write

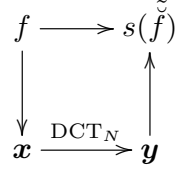


Figure 4.5: How we can approximate a function from its samples with the DCT.

$$\begin{aligned}
 (S\mathbf{x})_n &= \sum_{k=0}^{N-1} s_k x_{(n-k) \bmod N} \\
 &= s_0 x_n + \sum_{k=1}^{(N-1)/2} s_k x_{(n-k) \bmod N} + \sum_{k=(N+1)/2}^{N-1} s_k x_{(n-k) \bmod N} \\
 &= s_0 x_n + \sum_{k=1}^{(N-1)/2} s_k x_{(n-k) \bmod N} + \sum_{k=1}^{(N-1)/2} s_k x_{(n-(N-k)) \bmod N} \\
 &= s_0 x_n + \sum_{k=1}^{(N-1)/2} s_k (x_{(n-k) \bmod N} + x_{(n+k) \bmod N}). \tag{4.7}
 \end{aligned}$$

If we compare the first and last expressions here, we need the same number of summations, but the number of multiplications needed in the latter expression has been halved.

**Observation 4.20.** *Reducing arithmetic operations for symmetric filters.*

Assume that a symmetric filter has  $2s + 1$  filter coefficients. The filter applied to a vector of length  $N$  can then be implemented using  $(s + 1)N$  multiplications and  $2sN$  additions. This gives a reduced number of arithmetic operations when compared to a filter with the same number of coefficients which is not symmetric, where a direct implementations requires  $(2s + 1)N$  multiplications and  $2sN$  additions.

Similarly to as in Section 3.6.2, a symmetric filter can be factored into a product of symmetric filters. To see how, note first that a real polynomial is symmetric if and only if  $1/a$  is a root whenever  $a$  is. If we pair together the factors for the roots  $a, 1/a$  when  $a$  is real we get a component in the frequency response of degree 2. If we pair the factors for the roots  $a, 1/a, \bar{a}, 1/\bar{a}$  when  $a$  is complex, we get a component in the frequency response of degree 4. We thus get the following idea:

**Idea 4.21.** *Factorizing symmetric filters.*

Let  $S$  be a symmetric filter with real coefficients. There exist constants  $K, a_1, \dots, a_m, b_1, c_1, \dots, b_n, c_n$  so that



$$\begin{aligned}\lambda_S(\omega) = & K(a_1 e^{i\omega} + 1 + a_1 e^{-i\omega}) \dots (a_m e^{i\omega} + 1 + a_m e^{-i\omega}) \\ & \times (b_1 e^{2i\omega} + c_1 e^{i\omega} + 1 + c_1 e^{-i\omega} + b_1 e^{-2i\omega}) \dots \\ & \times (b_n e^{2i\omega} + c_n e^{i\omega} + 1 + c_n e^{-i\omega} + b_n e^{-2i\omega}).\end{aligned}$$

We can write  $S = KA_1 \dots A_m B_1 \dots B_n$ , where  $A_i = \{a_i, \underline{1}, a_i\}$  and  $B_i = \{b_i, c_i, \underline{1}, c_i, b_i\}$ .

In any case we see that the component filters have 3 and 5 filter coefficients.

### Exercise 4.7: Component expressions for a symmetric filter

Assume that  $S = t_{-L}, \dots, t_0, \dots, t_L$  is a symmetric filter. Use Equation (4.7) to show that  $z_n = (S\mathbf{x})_n$  in this case can be split into the following different formulas, depending on  $n$ :

a)  $0 \leq n < L$ :

$$z_n = t_0 x_n + \sum_{k=1}^n t_k (x_{n+k} + x_{n-k}) + \sum_{k=n+1}^L t_k (x_{n+k} + x_{n-k+N}). \quad (4.8)$$

b)  $L \leq n < N - L$ :

$$z_n = t_0 x_n + \sum_{k=1}^L t_k (x_{n+k} + x_{n-k}). \quad (4.9)$$

c)  $N - L \leq n < N$ :

$$z_n = t_0 x_n + \sum_{k=1}^{N-1-n} t_k (x_{n+k} + x_{n-k}) + \sum_{k=N-1-n+1}^L t_k (x_{n+k-N} + x_{n-k}). \quad (4.10)$$

The `convolve` function may not pick up this reduction in the number of multiplications, since it does not assume that the filter is symmetric. We will still use the `convolve` function in implementations, however, due to its heavy optimization.

## 4.3 Efficient implementations of the DCT

When we defined the DCT in the preceding section, we considered symmetric vectors of twice the length, and viewed these in the frequency domain. In order to have a fast algorithm for the DCT, which are comparable to the FFT algorithms we developed in Section 2.3, we need to address the fact that vectors of twice the length seem to be involved. The following theorem addresses this. This

result is much used in practical implementations of DCT, and can also be used for practical implementation of the DFT as we will see in Exercise 4.9. Note that the result, and the following results in this section, are stated in terms of the cosine matrix  $C_N$  (where the entries are  $(C_N)_{n,k} = \cos\left(2\pi\frac{n}{2N}\left(k + \frac{1}{2}\right)\right)$ , rather than the  $\text{DCT}_N$  matrix (which uses the additional scaling factor  $d_{n,N}$  for the rows). The reason is that  $C_N$  appears to me most practical for stating algorithms. When computing the DCT, we simply need to scale with the  $d_{n,N}$  at the end, after using the statements below.

**Theorem 4.22.** *DCT algorithm.*

Let  $\mathbf{y} = C_N \mathbf{x}$ . Then we have that

$$y_n = \left( \cos\left(\pi\frac{n}{2N}\right) \Re((\text{DFT}_N \mathbf{x}^{(1)})_n) + \sin\left(\pi\frac{n}{2N}\right) \Im((\text{DFT}_N \mathbf{x}^{(1)})_n) \right), \quad (4.11)$$

where  $\mathbf{x}^{(1)} \in \mathbb{R}^N$  is defined by

$$\begin{aligned} (\mathbf{x}^{(1)})_k &= x_{2k} \text{ for } 0 \leq k \leq N/2 - 1 \\ (\mathbf{x}^{(1)})_{N-k-1} &= x_{2k+1} \text{ for } 0 \leq k \leq N/2 - 1, \end{aligned}$$

*Proof.* Using the definition of  $C_N$ , and splitting the computation of  $\mathbf{y} = C_N \mathbf{x}$  into two sums, corresponding to the even and odd indices as follows:

$$\begin{aligned} y_n &= \sum_{k=0}^{N-1} x_k \cos\left(2\pi\frac{n}{2N}\left(k + \frac{1}{2}\right)\right) \\ &= \sum_{k=0}^{N/2-1} x_{2k} \cos\left(2\pi\frac{n}{2N}\left(2k + \frac{1}{2}\right)\right) + \sum_{k=0}^{N/2-1} x_{2k+1} \cos\left(2\pi\frac{n}{2N}\left(2k + 1 + \frac{1}{2}\right)\right). \end{aligned}$$

If we reverse the indices in the second sum, this sum becomes

$$\sum_{k=0}^{N/2-1} x_{N-2k-1} \cos\left(2\pi\frac{n}{2N}\left(N - 2k - 1 + \frac{1}{2}\right)\right).$$

If we then also shift the indices with  $N/2$  in this sum, we get

$$\begin{aligned} &\sum_{k=N/2}^{N-1} x_{2N-2k-1} \cos\left(2\pi\frac{n}{2N}\left(2N - 2k - 1 + \frac{1}{2}\right)\right) \\ &= \sum_{k=N/2}^{N-1} x_{2N-2k-1} \cos\left(2\pi\frac{n}{2N}\left(2k + \frac{1}{2}\right)\right), \end{aligned}$$

where we used that  $\cos$  is symmetric and periodic with period  $2\pi$ . We see that we now have the same  $\cos$ -terms in the two sums. If we thus define the vector  $\mathbf{x}^{(1)}$  as in the text of the theorem, we see that we can write

$$\begin{aligned} y_n &= \sum_{k=0}^{N-1} (\mathbf{x}^{(1)})_k \cos\left(2\pi \frac{n}{2N} \left(2k + \frac{1}{2}\right)\right) \\ &= \Re\left(\sum_{k=0}^{N-1} (\mathbf{x}^{(1)})_k e^{-2\pi i n(2k+\frac{1}{2})/(2N)}\right) \\ &= \Re\left(e^{-\pi i n/(2N)} \sum_{k=0}^{N-1} (\mathbf{x}^{(1)})_k e^{-2\pi i n k/N}\right) \\ &= \Re\left(e^{-\pi i n/(2N)} (\text{DFT}_N \mathbf{x}^{(1)})_n\right) \\ &= \left(\cos\left(\pi \frac{n}{2N}\right) \Re((\text{DFT}_N \mathbf{x}^{(1)})_n) + \sin\left(\pi \frac{n}{2N}\right) \Im((\text{DFT}_N \mathbf{x}^{(1)})_n)\right), \end{aligned}$$

where we have recognized the  $N$ -point DFT. This completes the proof.  $\square$

With the result above we have avoided computing a DFT of double size. If we in the proof above define the  $N \times N$ -diagonal matrix  $Q_N$  by  $Q_{n,n} = e^{-\pi i n/(2N)}$ , the result can also be written on the more compact form

$$\mathbf{y} = C_N \mathbf{x} = \Re\left(Q_N \text{DFT}_N \mathbf{x}^{(1)}\right).$$

We will, however, not use this form, since there is complex arithmetic involved, contrary to Equation(4.11). Code which uses Equation (4.11) to compute the DCT, using the function `FFTImpl` from Section 2.3, can look as follows:

```
def DCTImpl(x):
    """
    Compute the DCT of the vector x

    x: a vector
    """
    N = len(x)
    if N > 1:
        x1 = concatenate([x[0::2], x[-1:0:-2]]).astype(complex)
        FFTImpl(x1, FFTKernelStandard)
        cosvec = cos(pi*arange(float(N))/(2*N))
        sinvec = sin(pi*arange(float(N))/(2*N))
        if ndim(x) == 1:
            x[:] = cosvec*real(x1) + sinvec*imag(x1)
        else:
            for s2 in xrange(shape(x)[1]):
                x[:, s2] = cosvec*real(x1[:, s2]) \
                    + sinvec*imag(x1[:, s2])
        x[0] *= sqrt(1/float(N))
        x[1:] *= sqrt(2/float(N))
```

In the code, the vector  $\mathbf{x}^{(1)}$  is created first by rearranging the components, and it is sent as input to `FFTImpl`. After this we take real parts and imaginary parts, and multiply with the  $\cos$ - and  $\sin$ -terms in Equation (4.11).

### 4.3.1 Efficient implementations of the IDCT

As with the FFT, it is straightforward to modify the DCT implementation so that it returns the IDCT. To see how we can do this, write from Theorem 4.22, for  $n \geq 1$

$$\begin{aligned} y_n &= \left( \cos\left(\pi \frac{n}{2N}\right) \Re((\text{DFT}_N \mathbf{x}^{(1)})_n) + \sin\left(\pi \frac{n}{2N}\right) \Im((\text{DFT}_N \mathbf{x}^{(1)})_n) \right) \\ y_{N-n} &= \left( \cos\left(\pi \frac{N-n}{2N}\right) \Re((\text{DFT}_N \mathbf{x}^{(1)})_{N-n}) + \sin\left(\pi \frac{N-n}{2N}\right) \Im((\text{DFT}_N \mathbf{x}^{(1)})_{N-n}) \right) \\ &= \left( \sin\left(\pi \frac{n}{2N}\right) \Re((\text{DFT}_N \mathbf{x}^{(1)})_n) - \cos\left(\pi \frac{n}{2N}\right) \Im((\text{DFT}_N \mathbf{x}^{(1)})_n) \right), \end{aligned} \quad (4.12)$$

where we have used the symmetry of  $\text{DFT}_N$  for real signals. These two equations enable us to determine  $\Re((\text{DFT}_N \mathbf{x}^{(1)})_n)$  and  $\Im((\text{DFT}_N \mathbf{x}^{(1)})_n)$  from  $y_n$  and  $y_{N-n}$ . We get

$$\begin{aligned} \cos\left(\pi \frac{n}{2N}\right) y_n + \sin\left(\pi \frac{n}{2N}\right) y_{N-n} &= \Re((\text{DFT}_N \mathbf{x}^{(1)})_n) \\ \sin\left(\pi \frac{n}{2N}\right) y_n - \cos\left(\pi \frac{n}{2N}\right) y_{N-n} &= \Im((\text{DFT}_N \mathbf{x}^{(1)})_n). \end{aligned}$$

Adding we get

$$\begin{aligned} (\text{DFT}_N \mathbf{x}^{(1)})_n &= \cos\left(\pi \frac{n}{2N}\right) y_n + \sin\left(\pi \frac{n}{2N}\right) y_{N-n} + i(\sin\left(\pi \frac{n}{2N}\right) y_n - \cos\left(\pi \frac{n}{2N}\right) y_{N-n}) \\ &= (\cos\left(\pi \frac{n}{2N}\right) + i \sin\left(\pi \frac{n}{2N}\right))(y_n - i y_{N-n}) = e^{\pi i n / (2N)}(y_n - i y_{N-n}). \end{aligned}$$

This means that  $(\text{DFT}_N \mathbf{x}^{(1)})_n = e^{\pi i n / (2N)}(y_n + i y_{N-n}) = (y_n + i y_{N-n})/Q_{n,n}$  for  $n \geq 1$ . Since  $\Im((\text{DFT}_N \mathbf{x}^{(1)})_0) = 0$  we have that  $(\text{DFT}_N \mathbf{x}^{(1)})_0 = \frac{1}{d_{0,N}} y_0 = y_0/Q_{0,0}$ . This means that  $\mathbf{x}^{(1)}$  can be recovered by taking the IDFT of the vector with component 0 being  $y_0/Q_{0,0}$ , and the remaining components being  $(y_n - i y_{N-n})/Q_{n,n}$ :

**Theorem 4.23.** *IDCT algorithm.*

Let  $\mathbf{x} = (C_N)^{-1} \mathbf{y}$ , and let  $\mathbf{z}$  be the vector with component 0 being  $y_0/Q_{0,0}$ , and the remaining components being  $(y_n - i y_{N-n})/Q_{n,n}$ . Then we have that

$$\mathbf{x}^{(1)} = \text{IDFT}_N \mathbf{z},$$

where  $\mathbf{x}^{(1)}$  is defined as in Theorem 4.22.

The implementation of IDCT can thus go as follows:

```

def IDCTImpl(y):
    """
    Compute the IDCT of the vector y

    y: a vector
    """
    N = len(y)
    if N > 1:
        y[0] /= sqrt(1/float(N))
        y[1:] /= sqrt(2/float(N))
        Q = exp(-pi*1j*arange(float(N))/(2*N))
        y1 = zeros_like(y).astype(complex)
        y1[0] = y[0]/Q[0]
        if ndim(y) == 1:
            y1[1:] = (y[1:] - 1j*y[-1:0:-1])/Q[1:]
        else:
            for s2 in xrange(shape(y)[1]):
                y1[1:, s2] = (y[1:, s2] - 1j*y[-1:0:-1, s2])/Q[1:]
    FFTImpl(y1, FFTKernelStandard, 0)
    y[0::2] = real(y1[0:(N/2)])
    y[1::2] = real(y1[-1:(N/2-1):-1])

```

### 4.3.2 Reduction in the number of arithmetic operations

Let us also state a result which confirms that the DCT and IDCT implementations we have described give the same type of reductions in the number multiplications as the FFT and IFFT:

**Theorem 4.24.** *Number of multiplications required by the DCT and IDCT algorithms.*

The DCT and the IDCT can be implemented so that they use any FFT and IFFT algorithms. Their operation counts then have the same order as these. In particular, when the standard FFT algorithms of Section 2.3 are used, i.e. their operation counts are  $O(5N \log_2 N/2)$ . In comparison, the operation count for a direct implementation of the  $N$ -point DCT/IDCT is  $2N^2$ .

Note that we divide the previous operation counts by 2 since the DCT applies an FFT to real input only, and the operation count for the FFT can be halved when we adapt to real data, see Exercise 2.27.

*Proof.* By Theorem 2.20, the number of multiplications required by the standard FFT algorithm from Section 2.3 adapted to real data is  $O(N \log_2 N)$ , while the number of additions is  $O(3N \log_2 N/2)$ . By Theorem 4.22, two additional multiplications and one addition are required for each index (so that we have  $2N$  extra real multiplications and  $N$  extra real additions in total), but this does not affect the operation count, since  $O(N \log_2 N + 2N) = O(N \log_2 N)$ . Since the operation counts for the IFFT is the same as for the FFT, we only need to count the additional multiplications needed in forming the vector  $z = (y_n - iy_{N-n})/Q_{n,n}$ . Clearly, this also does not affect the order of the algorithm.  $\square$

Since the DCT and IDCT can be implemented using the FFT and IFFT, it has the same advantages as the FFT when it comes to parallel computing.

Much literature is devoted to reducing the number of multiplications in the DFT and the DCT even further than what we have done (see [22] for one of the most recent developments). Another note on computational complexity is in order: we have not counted the operations  $\sin$  and  $\cos$  in the DCT. The reason is that these values can be precomputed, since we take the sine and cosine of a specific set of values for each DCT or DFT of a given size. This is contrary to multiplication and addition, since these include the input values, which are only known at runtime. We have, however, not written down that we use precomputed arrays for sine and cosine in our algorithms: This is an issue to include in more optimized algorithms.

### Exercise 4.8: Trick for reducing the number of multiplications with the DCT

In this exercise we will take a look at a small trick which reduces the number of additional multiplications we need for DCT algorithm from Theorem 4.22. This exercise does not reduce the order of the DCT algorithms, but we will see in Exercise 4.9 how the result can be used to achieve this.

a) Assume that  $\mathbf{x}$  is a real signal. Equation (4.12), which said that

$$\begin{aligned} y_n &= \cos\left(\pi\frac{n}{2N}\right) \Re((\text{DFT}_N \mathbf{x}^{(1)})_n) + \sin\left(\pi\frac{n}{2N}\right) \Im((\text{DFT}_N \mathbf{x}^{(1)})_n) \\ y_{N-n} &= \sin\left(\pi\frac{n}{2N}\right) \Re((\text{DFT}_N \mathbf{x}^{(1)})_n) - \cos\left(\pi\frac{n}{2N}\right) \Im((\text{DFT}_N \mathbf{x}^{(1)})_n) \end{aligned}$$

for the  $n$ 'th and  $N - n$ 'th coefficient of the DCT. This can also be rewritten as

$$\begin{aligned} y_n &= \left(\Re((\text{DFT}_N \mathbf{x}^{(1)})_n) + \Im((\text{DFT}_N \mathbf{x}^{(1)})_n)\right) \cos\left(\pi\frac{n}{2N}\right) \\ &\quad - \Im((\text{DFT}_N \mathbf{x}^{(1)})_n) \left(\cos\left(\pi\frac{n}{2N}\right) - \sin\left(\pi\frac{n}{2N}\right)\right) \\ y_{N-n} &= -\left(\Re((\text{DFT}_N \mathbf{x}^{(1)})_n) + \Im((\text{DFT}_N \mathbf{x}^{(1)})_n)\right) \cos\left(\pi\frac{n}{2N}\right) \\ &\quad + \Re((\text{DFT}_N \mathbf{x}^{(1)})_n) \left(\sin\left(\pi\frac{n}{2N}\right) + \cos\left(\pi\frac{n}{2N}\right)\right). \end{aligned}$$

Explain that the first two equations require 4 multiplications to compute  $y_n$  and  $y_{N-n}$ , and that the last two equations require 3 multiplications to compute  $y_n$  and  $y_{N-n}$ .

b) Explain why the trick in a) reduces the number of additional multiplications in a DCT, from  $2N$  to  $3N/2$ .

c) Explain why the trick in a) can be used to reduce the number of additional multiplications in an IDCT with the same number.

**Hint.** match the expression  $e^{\pi i n/(2N)}(y_n - i y_{N-n})$  you encountered in the IDCT with the rewriting you did in b).

d) Show that the penalty of the trick we here have used to reduce the number of multiplications, is an increase in the number of additional additions from  $N$  to  $3N/2$ . Why can this trick still be useful?

### Exercise 4.9: An efficient joint implementation of the DCT and the FFT

In this exercise we will explain another joint implementation of the DFT and the DCT, which has the benefit of a low multiplication count, at the expense of a higher addition count. It also has the benefit that it is specialized to real vectors, with a very structured implementation (this is not always the case for the quickest FFT implementations. Not surprisingly, one often sacrifices clarity of code when one pursues higher computational speed). a) of this exercise can be skipped, as it is difficult and quite technical. For further details of the algorithm the reader is referred to [48].

a) Let  $\mathbf{y} = \text{DFT}_N \mathbf{x}$  be the  $N$ -point DFT of the real vector  $\mathbf{x}$ . Show that

$$\Re(y_n) = \begin{cases} \Re((\text{DFT}_{N/2} \mathbf{x}^{(e)})_n) + (C_{N/4} \mathbf{z})_n & 0 \leq n \leq N/4 - 1 \\ \Re((\text{DFT}_{N/2} \mathbf{x}^{(e)})_n) & n = N/4 \\ \Re((\text{DFT}_{N/2} \mathbf{x}^{(e)})_n) - (C_{N/4} \mathbf{z})_{N/2-n} & N/4 + 1 \leq n \leq N/2 - 1 \end{cases} \quad (4.13)$$

$$\Im(y_n) = \begin{cases} \Im((\text{DFT}_{N/2} \mathbf{x}^{(e)})_n) & n = 0 \\ \Im((\text{DFT}_{N/2} \mathbf{x}^{(e)})_n) + (C_{N/4} \mathbf{w})_{N/4-n} & 1 \leq n \leq N/4 - 1 \\ \Im((\text{DFT}_{N/2} \mathbf{x}^{(e)})_n) + (C_{N/4} \mathbf{w})_{n-N/4} & N/4 \leq n \leq N/2 - 1 \end{cases} \quad (4.14)$$

where  $\mathbf{x}^{(e)}$  is as defined in Theorem 2.15, where  $\mathbf{z}, \mathbf{w} \in \mathbb{R}^{N/4}$  defined by

$$\begin{aligned} z_k &= x_{2k+1} + x_{N-2k-1} & 0 \leq k \leq N/4 - 1, \\ w_k &= (-1)^k (x_{N-2k-1} - x_{2k+1}) & 0 \leq k \leq N/4 - 1, \end{aligned}$$

Explain from this how you can make an algorithm which reduces an FFT of length  $N$  to an FFT of length  $N/2$  (on  $\mathbf{x}^{(e)}$ ), and two DCT's of length  $N/4$  (on  $\mathbf{z}$  and  $\mathbf{w}$ ). We will call this algorithm the revised FFT algorithm.

a) says nothing about the coefficients  $y_n$  for  $n > \frac{N}{2}$ . These are obtained in the same way as before through symmetry. a) also says nothing about  $y_{N/2}$ . This can be obtained with the same formula as in Theorem 2.15.

Let us now compute the number of arithmetic operations our revised algorithm needs. Denote by the number of real multiplications needed by the revised  $N$ -point FFT algorithm

b) Explain from the algorithm in a) that

$$M_N = 2(M_{N/4} + 3N/8) + M_{N/2} \quad A_N = 2(A_{N/4} + 3N/8) + A_{N/2} + 3N/2 \quad (4.15)$$

**Hint.**  $3N/8$  should come from the extra additions/multiplications (see Exercise 4.8) you need to compute when you run the algorithm from Theorem 4.22 for  $C_{N/4}$ . Note also that the equations in a) require no extra multiplications, but that there are six equations involved, each needing  $N/4$  additions, so that we need  $6N/4 = 3N/2$  extra additions.

c) Explain why  $x_r = M_{2^r}$  is the solution to the difference equation

$$x_{r+2} - x_{r+1} - 2x_r = 3 \times 2^r,$$

and that  $x_r = A_{2^r}$  is the solution to

$$x_{r+2} - x_{r+1} - 2x_r = 9 \times 2^r.$$

and show that the general solution to these are  $x_r = \frac{1}{2}r2^r + C2^r + D(-1)^r$  for multiplications, and  $x_r = \frac{3}{2}r2^r + C2^r + D(-1)^r$  for additions.

d) Explain why, regardless of initial conditions to the difference equations,  $M_N = O(\frac{1}{2}N \log_2 N)$  and  $A_N = O(\frac{3}{2}N \log_2 N)$  both for the revised FFT and the revised DCT. The total number of operations is thus  $O(2N \log_2 N)$ , i.e. half the operation count of the split-radix algorithm. The orders of these algorithms are thus the same, since we here have adapted to read data.

e) Explain that, if you had not employed the trick from Exercise 4.8, we would instead have obtained  $M_N = O(\frac{2}{3} \log_2 N)$ , and  $A_N = O(\frac{4}{3} \log_2 N)$ , which equal the orders for the number of multiplications/additions for the split-radix algorithm. In particular, the order of the operation count remains the same, but the trick from Exercise 4.8 turned a bigger percentage of the arithmetic operations into additions.

The algorithm we here have developed thus is constructed from the beginning to apply for real data only. Another advantage of the new algorithm is that it can be used to compute both the DCT and the DFT.

### Exercise 4.10: Implementation of the IFFT/IDCT

We did not write down corresponding algorithms for the revised IFFT and IDCT algorithms. We will consider this in this exercise.

a) Using equations (4.13)-(4.14), show that

$$\begin{aligned} \Re(y_n) - \Re(y_{N/2-n}) &= 2(C_{N/4}\mathbf{z})_n \\ \Im(y_n) + \Im(y_{N/2-n}) &= 2(C_{N/4}\mathbf{w})_{N/4-n} \end{aligned}$$

for  $1 \leq n \leq N/4 - 1$ . Explain how one can compute  $\mathbf{z}$  and  $\mathbf{w}$  from this using two IDCT's of length  $N/4$ .



b) Using equations (4.13)-(4.14), show that

$$\begin{aligned}\Re(y_n) + \Re(y_{N/2-n}) &= \Re((\text{DFT}_{N/2}\mathbf{x}^{(e)})_n) \\ \Im(y_n) - \Im(y_{N/2-n}) &= \Im((\text{DFT}_{N/2}\mathbf{x}^{(e)})_n),\end{aligned}$$

and explain how one can compute  $\mathbf{x}^{(e)}$  from this using an IFFT of length  $N/2$ .

## 4.4 Summary

We started this chapter by extending a previous result which had to do with that the Fourier series of a symmetric function converged quicker. To build on this we first needed to define symmetric extensions of vectors and symmetric vectors, before we classified symmetric extensions in the frequency domain. From this we could find a nice, orthonormal basis for the symmetric extensions, which lead us to the definition of the DCT. We also saw a connection with symmetric filters: These are exactly the filters which preserve symmetric extensions, and we could characterize symmetric filters restricted to symmetric extension as an  $N$ -dimensional mapping. We also showed that it is smart to replace the DFT with the DCT when we work with filters which are known to be symmetric. Among other things, this lead to a better way of approximating analog filters, and better interpolation of functions.

We also showed how to obtain an efficient implementation of the DCT, which could reuse the FFT implementation. The DCT has an important role in the MP3 standard. As we have explained, the MP3 standard applies several filters to the sound, in order to split it into bands concentrating on different frequency ranges. Later we will look closer at how these filters can be implemented and constructed. The implementation can use transforms similar to the MDCT, as explained in Exercise 4.6. The MDCT is also used in the more advanced version of the MP3 standard (layer III). Here it is applied to the filtered data to obtain a higher spectral resolution of the sound. The MDCT is applied to groups of 576 (in special circumstances 192) samples. The MP3 standard document [20] does not dig into the theory for this, only representing what is needed in order to make an implementation. It is somewhat difficult to read this document, since it is written in quite a different language, familiar mainly to those working with international standards.

The different type of cosine-matrices can all be associated with some extension strategy for the signal. [34] contains a review of these.

The DCT is particularly popular for processing sound data before they are compressed with lossless techniques such as Huffman coding or arithmetic coding. The reason is, as mentioned, that the DCT provides a better approximation from a low-dimensional space than the DFT does, and that it has a very efficient implementation. Libraries exist which goes into lengths to provide efficient implementation of the FFT and the DCT. FFTW, short for *Fastest Fourier Transform in the West* [17], is perhaps the best known of these.

Signal processing literature often does not motivate digital filters in explaining where they come from, and where the input to the filters come from. Using analog filters to motivate this, and to argue for improvements in using the DCT and symmetric extensions, is not that common. Much literature simply says that the property of linear phase is good, without elaborating on this further.

## Chapter 5

# Motivation for wavelets and some simple examples

In the first part of the book our focus was to approximate functions or vectors with trigonometric functions. We saw that the Discrete Fourier transform could be used to obtain a representation of a vector in terms of such functions, and that computations could be done efficiently with the FFT algorithm. This was useful for analyzing, filtering, and compressing sound and other discrete data. The approach with trigonometric functions has some limitations, however. One of these is that, in a representation with trigonometric functions, the frequency content is fixed over time. This is in contrast with most sound data, where the characteristics are completely different in different parts. We have also seen that, even if a sound has a simple representation in terms of trigonometric functions on two different parts, the representation of the entire sound may not be simple. In particular, if the function is nonzero only on a very small interval, a representation of it in terms of trigonometric functions is not so simple.

In this chapter we are going to introduce the basic properties of an alternative to Fourier analysis for representing functions. This alternative is called wavelets. Similar to Fourier analysis, wavelets are also based on the idea of expressing a function in some basis. But in contrast to Fourier analysis, where the basis is fixed, wavelets provide a general framework with many different types of bases. In this chapter we first give a motivation for wavelets, before we continue by introducing some very simple wavelets. The first wavelet we look at can be interpreted as an approximation scheme based on piecewise constant functions. The next wavelet we look at is similar, but with piecewise linear functions used instead. Following these examples we will establish a more general framework, based on experiences from the simple wavelets. In the following chapters we will interpret this framework in terms of filters, and use this connection to construct even more interesting wavelets.

Core functions in this chapter are collected in a module called `dwt`.

## 5.1 Why wavelets?

The left image in Figure 5.1 shows a view of the entire Earth.

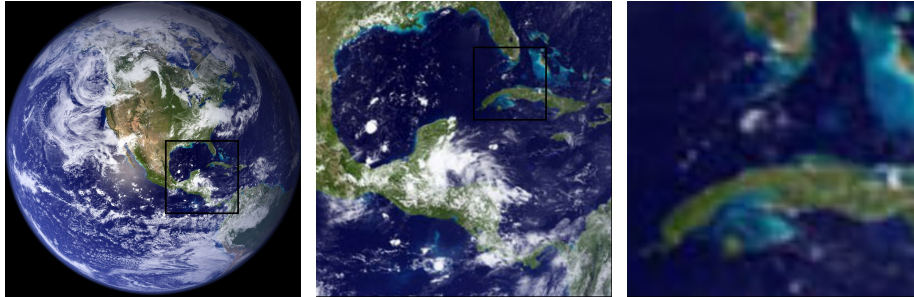


Figure 5.1: A view of Earth from space, together with versions of the image where we have zoomed in.

The startup image in Google Earth<sup>TM</sup>, a program for viewing satellite images, maps and other geographic information, is very similar to this. In the middle image we have zoomed in on the Mexican Gulf, as marked with a rectangle in the left image. Similarly, in the right image we have further zoomed in on Cuba and a small portion of Florida, as marked with a rectangle in the middle image. There is clearly an amazing amount of information available behind a program like Google Earth<sup>TM</sup>, since we there can zoom further in, and obtain enough detail to differentiate between buildings and even trees or cars all over the Earth. So, when the Earth is spinning in the opening screen of Google Earth<sup>TM</sup>, all the Earth's buildings appear to be spinning with it! If this was the case the Earth would not be spinning on the screen, since there would just be so much information to process that a laptop would not be able to display a rotating Earth.

There is a simple reason that the globe can be shown spinning in spite of the huge amounts of information that need to be handled. We are going to see later that a digital image is just a rectangular array of numbers that represent the color at a dense set of points. As an example, the images in Figure 5.1 are made up of a grid of  $1064 \times 1064$  points, which gives a total of 1 132 096 points. The color at a point is represented by three eight-bit integers, which means that the image files contain a total of 3 396 288 bytes each. So regardless of how close to the surface of the Earth our viewpoint is, the resulting image always contains the same number of points. This means that when we are far away from the Earth we can use a very coarse model of the geographic information that is being displayed, but as we zoom in, we need to display more details and therefore need a more accurate model.

### Observation 5.1. *Images model.*

When discrete information is displayed in an image, there is no need to use a mathematical model that contains more detail than what is visible in the image.

A consequence of Observation 5.1 is that for applications like Google Earth™ we should use a mathematical model that makes it easy to switch between different levels of detail, or different resolutions. Such models are called *multiresolution models*, and wavelets are prominent examples of this kind of models. We will see that multiresolution models also provide us with means of approximating functions, just as Taylor series and Fourier series. Our new approximation scheme differs from these in one important respect, however: When we approximate with Taylor series and Fourier series, the error must be computed at the same data points as well, so that the error contains just as much information as the approximating function, and the function to be approximated. Multiresolution models on the other hand will be defined in such a way that the error and the “approximating function” each contain half of the information from the function we approximate, i.e. their amount of data is reduced. This property makes multiresolution models attractive for the problems at hand, when compared to approaches such as Taylor series and Fourier series.

When we zoom in with Google Earth™, it seems that this is done continuously. The truth is probably that the program only has representations at some given resolutions (since each representation requires memory), and that one interpolates between these to give the impression of a continuous zoom. In the coming chapters we will first look at how we can represent the information at different resolutions, so that only new information at each level is included.

We will now turn to how wavelets are defined more formally, and construct the simplest wavelet we have. Its construction goes in the following steps: First we introduce what we call resolution spaces, and the corresponding scaling function. Then we introduce the detail spaces, and the corresponding mother wavelet. These two functions will give rise to certain bases for these spaces, and we will define the Discrete Wavelet Transform as a change of coordinates between these bases.

## 5.2 A wavelet based on piecewise constant functions

Our starting point will be the space of piecewise constant functions on an interval  $[0, N)$ . This will be called a resolution space.

**Definition 5.2.** *The resolution space  $V_0$ .*

Let  $N$  be a natural number. The resolution space  $V_0$  is defined as the space of functions defined on the interval  $[0, N)$  that are constant on each subinterval  $[n, n + 1)$  for  $n = 0, \dots, N - 1$ .

Note that this also corresponds to piecewise constant functions which are periodic with period  $N$ . We will, just as we did in Fourier analysis, identify a function defined on  $[0, N)$  with its (period  $N$ ) periodic extension. An example of a function in  $V_0$  for  $N = 10$  is shown in Figure 5.2. It is easy to check that  $V_0$  is a linear space, and for computations it is useful to know the dimension of the space and have a basis.

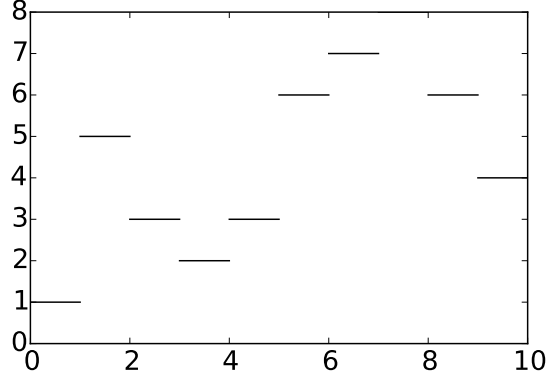


Figure 5.2: A piecewise constant function.

**Lemma 5.3.** *The function  $\phi$ .*

Define the function  $\phi(t)$  by

$$\phi(t) = \begin{cases} 1, & \text{if } 0 \leq t < 1; \\ 0, & \text{otherwise;} \end{cases} \quad (5.1)$$

and set  $\phi_n(t) = \phi(t - n)$  for any integer  $n$ . The space  $V_0$  has dimension  $N$ , and the  $N$  functions  $\{\phi_n\}_{n=0}^{N-1}$  form an orthonormal basis for  $V_0$  with respect to the standard inner product

$$\langle f, g \rangle = \int_0^N f(t)g(t) dt. \quad (5.2)$$

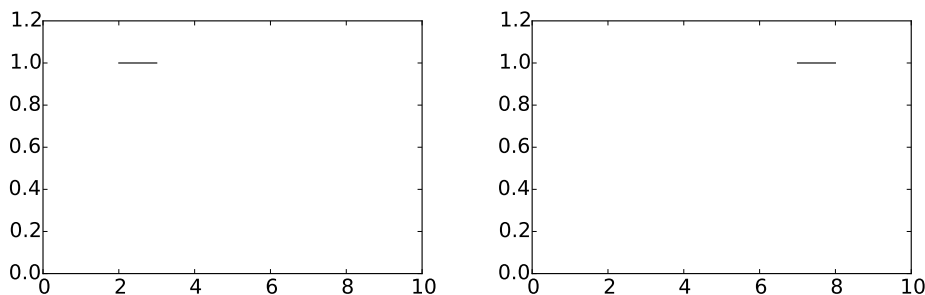
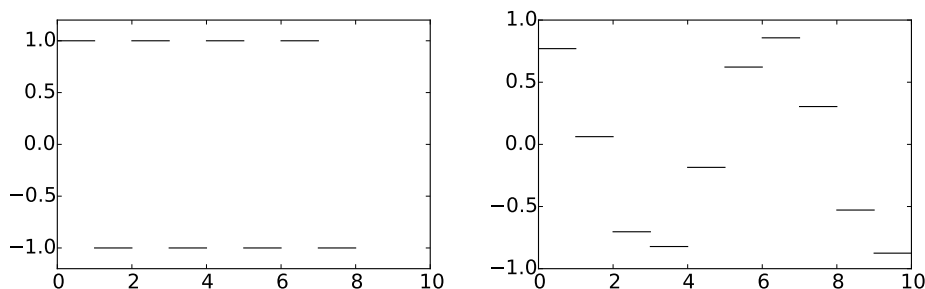
In particular, any  $f \in V_0$  can be represented as

$$f(t) = \sum_{n=0}^{N-1} c_n \phi_n(t) \quad (5.3)$$

for suitable coefficients  $(c_n)_{n=0}^{N-1}$ . The function  $\phi_n$  is referred to as the *characteristic* function of the interval  $[n, n + 1)$ .

Note the small difference between the inner product we define here from the inner product we used for functions previously: Here there is no scaling  $1/T$  involved. Also, for wavelets we will only consider real functions, and the inner product will therefore not be defined for complex functions. Two examples of the basis functions defined in Lemma 5.3 are shown in Figure 5.3.

*Proof.* Two functions  $\phi_{n_1}$  and  $\phi_{n_2}$  with  $n_1 \neq n_2$  clearly satisfy  $\int \phi_{n_1}(t)\phi_{n_2}(t)dt = 0$  since  $\phi_{n_1}(t)\phi_{n_2}(t) = 0$  for all values of  $x$ . It is also easy to check that  $\|\phi_n\| = 1$  for all  $n$ . Finally, any function in  $V_0$  can be written as a linear combination of the functions  $\phi_0, \phi_1, \dots, \phi_{N-1}$ , so the conclusion of the lemma follows.  $\square$

Figure 5.3: The basis functions  $\phi_2$  and  $\phi_7$  from  $\phi_0$ .Figure 5.4: Examples of functions from  $V_0$ . The square wave in  $V_0$  (left), and an approximation to  $\cos t$  from  $V_0$  (right).

In our discussion of Fourier analysis, the starting point was the function  $\sin(2\pi t)$  that has frequency 1. We can think of the space  $V_0$  as being analogous to this function: The function  $\sum_{n=0}^{N-1} (-1)^n \phi_n(t)$  is (part of the) square wave that we discussed in Chapter 1, and which also oscillates regularly like the sine function, see the left plot in Figure 5.4. The difference is that we have more flexibility since we have a whole space at our disposal instead of just one function — the right plot in Figure 5.4 shows another function in  $V_0$ .

In Fourier analysis we obtained a linear space of possible approximations by including sines of frequency 1, 2, 3,  $\dots$ , up to some maximum. We use a similar approach for constructing wavelets, but we double the frequency each time and label the spaces as  $V_0, V_1, V_2, \dots$

**Definition 5.4.** *Refined resolution spaces.*

The space  $V_m$  for the interval  $[0, N)$  is the space of piecewise constant functions defined on  $[0, N)$  that are constant on each subinterval  $[n/2^m, (n+1)/2^m)$  for  $n = 0, 1, \dots, 2^m N - 1$ .

Some examples of functions in the spaces  $V_1, V_2$  and  $V_3$  for the interval  $[0, 10]$  are shown in Figure 5.5. As  $m$  increases, we can represent smaller details. In particular, the function in the rightmost is a piecewise constant function that oscillates like  $\sin(2\pi 2^2 t)$  on the interval  $[0, 10]$ .

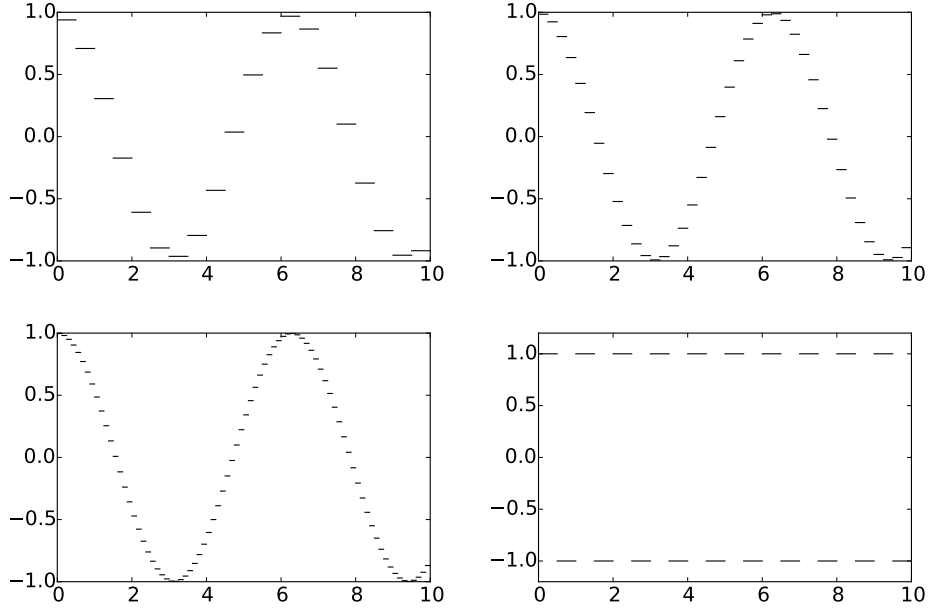


Figure 5.5: Piecewise constant approximations to  $\cos t$  on the interval  $[0, 10]$  in the spaces  $V_1$ ,  $V_2$ , and  $V_3$ . The lower right plot shows the square wave in  $V_2$ .

It is easy to find a basis for  $V_m$ , we just use the characteristic functions of each subinterval.

**Lemma 5.5.** *Basis for  $V_m$ .*

Let  $[0, N)$  be a given interval with  $N$  some positive integer. Then the dimension of  $V_m$  is  $2^m N$ . The functions

$$\phi_{m,n}(t) = 2^{m/2} \phi(2^m t - n), \quad \text{for } n = 0, 1, \dots, 2^m N - 1 \quad (5.4)$$

form an orthonormal basis for  $V_m$ , which we will denote by  $\phi_m$ . Any function  $f \in V_m$  can thus be represented uniquely as

$$f(t) = \sum_{n=0}^{2^m N - 1} c_{m,n} \phi_{m,n}(t).$$

*Proof.* The functions given by Equation (5.4) are nonzero on the subintervals  $[n/2^m, (n+1)/2^m)$  which we referred to in Definition 5.4, so that  $\phi_{m,n_1} \phi_{m,n_2} = 0$  when  $n_1 \neq n_2$ , since these intervals are disjoint. The only mysterious thing may be the normalisation factor  $2^{m/2}$ . This comes from the fact that

$$\int_0^N \phi(2^m t - n)^2 dt = \int_{n/2^m}^{(n+1)/2^m} \phi(2^m t - n)^2 dt = 2^{-m} \int_0^1 \phi(u)^2 du = 2^{-m}.$$

The normalisation therefore thus ensures that  $\|\phi_{m,n}\| = 1$  for all  $m$ .  $\square$



In the following we will always denote the coordinates in the basis  $\phi_m$  by  $c_{m,n}$ . Note that our definition restricts the dimensions of the spaces we study to be on the form  $N2^m$ . In Chapter 6 we will explain how this restriction can be dropped, but until then the dimensions will be assumed to be on this form. In the theory of wavelets, the function  $\phi$  is also called a *scaling function*. The origin behind this name is that the scaled (and translated) functions  $\phi_{m,n}$  of  $\phi$  are used as basis functions for the refined resolution spaces. Later on we will see that other scaling functions  $\phi$  can be chosen, where the scaled versions  $\phi_{m,n}$  will be used to define similar resolution spaces, with slightly different properties.

### 5.2.1 Function approximation property

Each time  $m$  is increased by 1, the dimension of  $V_m$  doubles, and the subinterval on which the functions in  $V_m$  are constant are halved in size. It therefore seems reasonable that, for most functions, we can find good approximations in  $V_m$  provided  $m$  is big enough.

**Theorem 5.6.** *Resolution spaces and approximation.*

Let  $f$  be a given function that is continuous on the interval  $[0, N]$ . Given  $\epsilon > 0$ , there exists an integer  $m \geq 0$  and a function  $g \in V_m$  such that

$$|f(t) - g(t)| \leq \epsilon$$

for all  $t$  in  $[0, N]$ .

*Proof.* Since  $f$  is (uniformly) continuous on  $[0, N]$ , we can find an integer  $m$  so that  $|f(t_1) - f(t_2)| \leq \epsilon$  for any two numbers  $t_1$  and  $t_2$  in  $[0, N]$  with  $|t_1 - t_2| \leq 2^{-m}$ . Define the approximation  $g$  by

$$g(t) = \sum_{n=0}^{2^m N - 1} f(t_{m,n+1/2}) \phi_{m,n}(t),$$

where  $t_{m,n+1/2}$  is the midpoint of the subinterval  $[n2^{-m}, (n+1)2^{-m})$ ,

$$t_{m,n+1/2} = (n + 1/2)2^{-m}.$$

For  $t$  in this subinterval we then obviously have  $|f(t) - g(t)| \leq \epsilon$ , and since these intervals cover  $[0, N]$ , the conclusion holds for all  $t \in [0, N]$ .  $\square$

Theorem 5.6 does not tell us how to find the approximation  $g$  although the proof makes use of an approximation that interpolates  $f$  at the midpoint of each subinterval. Note that if we measure the error in the  $L^2$ -norm, we have

$$\|f - g\|^2 = \int_0^N |f(t) - g(t)|^2 dt \leq N\epsilon^2,$$

so  $\|f - g\| \leq \epsilon\sqrt{N}$ . We therefore have the following corollary.

**Corollary 5.7.** *Resolution spaces and approximation.*

Let  $f$  be a given continuous function on the interval  $[0, N]$ . Then

$$\lim_{m \rightarrow \infty} \|f - \text{proj}_{V_m}(f)\| = 0.$$

Figure 5.6 illustrates how some of the approximations of the function  $f(x) = x^2$  from the resolution spaces for the interval  $[0, 1]$  improve with increasing  $m$ .

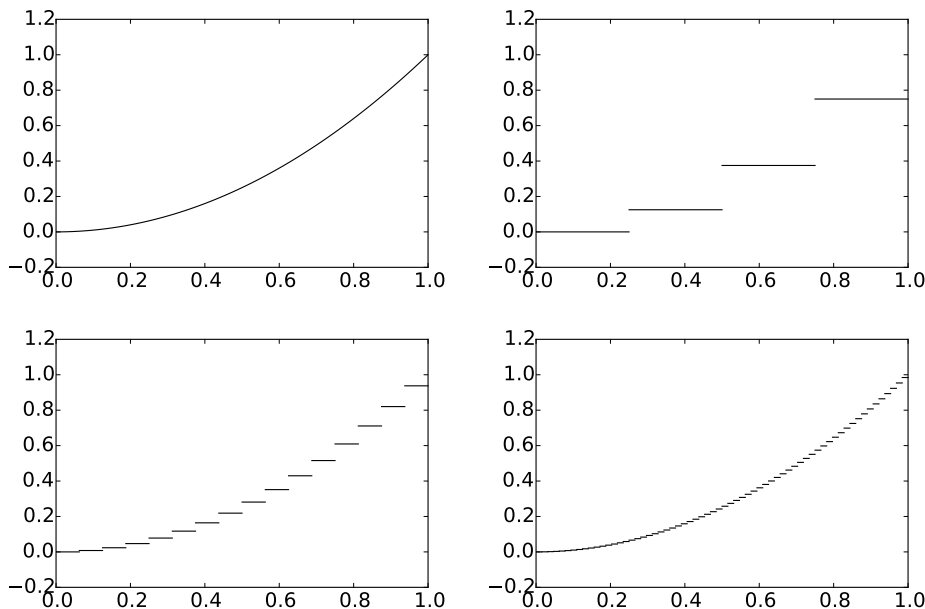


Figure 5.6: Comparison of the function defined by  $f(t) = t^2$  on  $[0, 1]$  with the projection onto  $V_2$ ,  $V_4$ , and  $V_6$ , respectively.

**5.2.2 Detail spaces and wavelets**

So far we have described a family of function spaces that allow us to determine arbitrarily good approximations to a continuous function. The next step is to introduce the so-called detail spaces and the wavelet functions. We start by observing that since

$$[n, n + 1) = [2n/2, (2n + 1)/2) \cup [(2n + 1)/2, (2n + 2)/2),$$

we have

$$\phi_{0,n} = \frac{1}{\sqrt{2}}\phi_{1,2n} + \frac{1}{\sqrt{2}}\phi_{1,2n+1}.$$

This provides a formal proof of the intuitive observation that  $V_0 \subset V_1$ , for if  $g \in V_0$ , we can write

$$g(t) = \sum_{n=0}^{N-1} c_{0,n} \phi_{0,n}(t) = \sum_{n=0}^{N-1} c_{0,n} (\phi_{1,2n} + \phi_{1,2n+1}) / \sqrt{2},$$

and the right-hand side clearly lies in  $V_1$ . Since also

$$\begin{aligned} \phi_{m-1,n}(t) &= 2^{(m-1)/2} \phi(2^{m-1}t - n) = 2^{(m-1)/2} \phi_{0,n}(2^{m-1}t) \\ &= 2^{(m-1)/2} \frac{1}{\sqrt{2}} (\phi_{1,2n}(2^{m-1}t) + \phi_{1,2n+1}(2^{m-1}t)) \\ &= 2^{(m-1)/2} (\phi(2^m t - 2n) + \phi(2^m t - (2n + 1))) = \frac{1}{\sqrt{2}} (\phi_{m,2n}(t) + \phi_{m,2n+1}(t)), \end{aligned}$$

we also have that

$$\phi_{m-1,n} = \frac{1}{\sqrt{2}} \phi_{m,2n} + \frac{1}{\sqrt{2}} \phi_{m,2n+1}, \quad (5.5)$$

so that also  $V_k \subset V_{k+1}$  for any integer  $k \geq 0$ .

**Lemma 5.8.** *Resolution spaces are nested.*

The spaces  $V_0, V_1, \dots, V_m, \dots$  are nested,

$$V_0 \subset V_1 \subset V_2 \subset \dots \subset V_m \dots$$

This means that it is meaningful to project  $V_{k+1}$  onto  $V_k$ . The next step is to characterize the projection from  $V_1$  onto  $V_0$ , and onto the orthogonal complement of  $V_0$  in  $V_1$ . Before we do this, let us make the following definitions.

**Definition 5.9.** *Detail spaces.*

The orthogonal complement of  $V_{m-1}$  in  $V_m$  is denoted  $W_{m-1}$ . All the spaces  $W_k$  are also called detail spaces, or error spaces.

The name detail space is used since the projection from  $V_m$  onto  $V_{m-1}$  is considered as a (low-resolution) approximation, and the error, which lies in  $W_{m-1}$ , is the detail which is left out when we replace with this approximation. We will also write  $g_m = g_{m-1} + e_{m-1}$  when we split  $g_m \in V_m$  into a sum of a low-resolution approximation and a detail component. In the context of our Google Earth™ example, in Figure 5.1 you should interpret  $g_0$  as the left image, the middle image as an excerpt of  $g_1$ , and  $e_0$  as the additional details which are needed to reproduce the middle image from the left image.

Since  $V_0$  and  $W_0$  are mutually orthogonal spaces they are also linearly independent spaces. When  $U$  and  $V$  are two such linearly independent spaces, we will write  $U \oplus V$  for the vector space consisting of all vectors of the form  $\mathbf{u} + \mathbf{v}$ , with  $\mathbf{u} \in U$ ,  $\mathbf{v} \in V$ .  $U \oplus V$  is also called the *direct sum* of  $U$  and  $V$ . This also makes sense if we have more than two vector spaces (such as  $U \oplus V \oplus W$ ),

and the direct sum clearly obeys the associate law  $U \oplus (V \oplus W) = (U \oplus V) \oplus W$ . Using the direct sum notation, we can first write

$$V_m = V_{m-1} \oplus W_{m-1}. \quad (5.6)$$

Since  $V_m$  has dimension  $2^m N$ , it follows that also  $W_{m-1}$  has dimension  $2^{m-1} N$ . We can continue the direct sum decomposition by also writing  $V_{m-1}$  as a direct sum, then  $V_{m-2}$  as a direct sum, and so on, and end up with

$$V_m = V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}, \quad (5.7)$$

where the spaces on the right hand side have dimension  $N, N, 2N, \dots, 2^{m-1} N$ . This decomposition will be important for our purposes. It says that the resolution space  $V_m$  can be written as the sum of a lower order resolution space  $V_0$ , and  $m$  detail spaces  $W_0, \dots, W_{m-1}$ . We will later interpret this splitting into a low-resolution component and  $m$  detail components.

It turns out that the following function will play the same role for the detail space  $W_k$  as the function  $\phi$  plays for the resolution space  $V_k$ .

**Definition 5.10.** *The function  $\psi$ .*

We define

$$\psi(t) = (\phi_{1,0}(t) - \phi_{1,1}(t))/\sqrt{2} = \phi(2t) - \phi(2t - 1), \quad (5.8)$$

and

$$\psi_{m,n}(t) = 2^{m/2} \psi(2^m t - n), \quad \text{for } n = 0, 1, \dots, 2^m N - 1. \quad (5.9)$$

The functions  $\phi$  and  $\psi$  are shown in Figure 5.7.

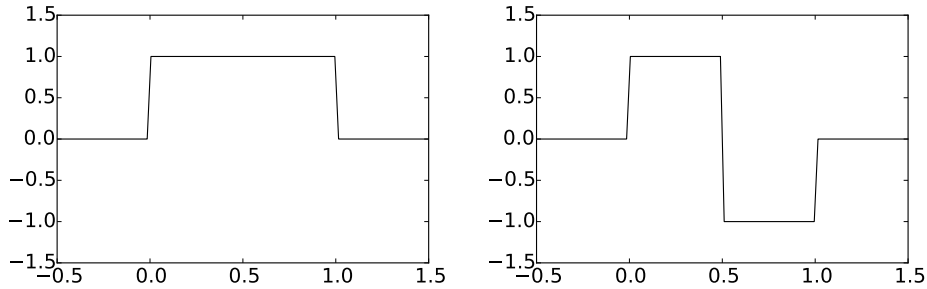


Figure 5.7: The functions  $\phi$  and  $\psi$  we used to analyse the space of piecewise constant functions.

As in the proof for Equation (5.5), it follows that

$$\psi_{m-1,n} = \frac{1}{\sqrt{2}} \phi_{m,2n} - \frac{1}{\sqrt{2}} \phi_{m,2n+1}, \quad (5.10)$$

Clearly  $\psi$  is supported on  $[0, 1)$ , and  $\|\psi\| = 1$ . From this it follows as for  $\phi_0$  that the  $\{\psi_{0,n}\}_{n=0}^{N-1}$  are orthonormal. In the same way as for  $\phi_m$ , it follows

also that the  $\{\psi_{m,n}\}_{n=0}^{2^m N-1}$  is orthonormal for any  $m$ . We will write  $\psi_m$  for the orthonormal basis  $\{\psi_{m,n}\}_{n=0}^{2^m N-1}$ , and we will always denote the coordinates in the basis  $\psi_m$  by  $w_{m,n}$ . The next result motivates the definition of  $\psi$ , and states how we can project from  $V_1$  onto  $V_0$  and  $W_0$ , i.e. find the low-resolution approximation and the detail component of  $g_1 \in V_1$ .

**Lemma 5.11.** *Orthonormal bases.*

For  $0 \leq n < N$  we have that

$$\text{proj}_{V_0}(\phi_{1,n}) = \begin{cases} \phi_{0,n/2}/\sqrt{2}, & \text{if } n \text{ is even;} \\ \phi_{0,(n-1)/2}/\sqrt{2}, & \text{if } n \text{ is odd.} \end{cases} \quad (5.11)$$

$$\text{proj}_{W_0}(\phi_{1,n}) = \begin{cases} \psi_{0,n/2}/\sqrt{2}, & \text{if } n \text{ is even;} \\ -\psi_{0,(n-1)/2}/\sqrt{2}, & \text{if } n \text{ is odd.} \end{cases} \quad (5.12)$$

In particular,  $\psi_0$  is an orthonormal basis for  $W_0$ . More generally, if  $g_1 = \sum_{n=0}^{2N-1} c_{1,n} \phi_{1,n} \in V_1$ , then

$$\text{proj}_{V_0}(g_1) = \sum_{n=0}^{N-1} c_{0,n} \phi_{0,n}, \text{ where } c_{0,n} = \frac{c_{1,2n} + c_{1,2n+1}}{\sqrt{2}} \quad (5.13)$$

$$\text{proj}_{W_0}(g_1) = \sum_{n=0}^{N-1} w_{0,n} \psi_{0,n}, \text{ where } w_{0,n} = \frac{c_{1,2n} - c_{1,2n+1}}{\sqrt{2}}. \quad (5.14)$$

*Proof.* We first observe that  $\phi_{1,n}(t) \neq 0$  if and only if  $n/2 \leq t < (n+1)/2$ . Suppose that  $n$  is even. Then the intersection

$$\left[ \frac{n}{2}, \frac{n+1}{2} \right) \cap [n_1, n_1+1) \quad (5.15)$$

is nonempty only if  $n_1 = \frac{n}{2}$ . Using the orthogonal decomposition formula we get

$$\begin{aligned} \text{proj}_{V_0}(\phi_{1,n}) &= \sum_{k=0}^{N-1} \langle \phi_{1,n}, \phi_{0,k} \rangle \phi_{0,k} = \langle \phi_{1,n}, \phi_{0,n_1} \rangle \phi_{0,n_1} \\ &= \int_{n/2}^{(n+1)/2} \sqrt{2} dt \phi_{0,n/2} = \frac{1}{\sqrt{2}} \phi_{0,n/2}. \end{aligned}$$

Using this we also get

$$\begin{aligned} \text{proj}_{W_0}(\phi_{1,n}) &= \phi_{1,n} - \frac{1}{\sqrt{2}} \phi_{0,n/2} = \phi_{1,n} - \frac{1}{\sqrt{2}} \left( \frac{1}{\sqrt{2}} \phi_{1,n} + \frac{1}{\sqrt{2}} \phi_{1,n+1} \right) \\ &= \frac{1}{2} \phi_{1,n} - \frac{1}{2} \phi_{1,n+1} = \psi_{0,n/2}/\sqrt{2}. \end{aligned}$$

This proves the expressions for both projections when  $n$  is even. When  $n$  is odd, the intersection (5.15) is nonempty only if  $n_1 = (n - 1)/2$ , which gives the expressions for both projections when  $n$  is odd in the same way. In particular we get

$$\begin{aligned} \text{proj}_{W_0}(\phi_{1,n}) &= \phi_{1,n} - \frac{\phi_{0,(n-1)/2}}{\sqrt{2}} = \phi_{1,n} - \frac{1}{\sqrt{2}} \left( \frac{1}{\sqrt{2}}\phi_{1,n-1} + \frac{1}{\sqrt{2}}\phi_{1,n} \right) \\ &= \frac{1}{2}\phi_{1,n} - \frac{1}{2}\phi_{1,n-1} = -\psi_{0,(n-1)/2}/\sqrt{2}. \end{aligned}$$

$\psi_0$  must be an orthonormal basis for  $W_0$  since  $\psi_0$  is contained in  $W_0$ , and both have dimension  $N$ .

We project the function  $g_1$  in  $V_1$  using the formulas in (5.11). We first split the sum into even and odd values of  $n$ ,

$$g_1 = \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n} = \sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{1,2n+1}. \quad (5.16)$$

We can now apply the two formulas in (5.11),

$$\begin{aligned} \text{proj}_{V_0}(g_1) &= \text{proj}_{V_0} \left( \sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{1,2n+1} \right) \\ &= \sum_{n=0}^{N-1} c_{1,2n} \text{proj}_{V_0}(\phi_{1,2n}) + \sum_{n=0}^{N-1} c_{1,2n+1} \text{proj}_{V_0}(\phi_{1,2n+1}) \\ &= \sum_{n=0}^{N-1} c_{1,2n}\phi_{0,n}/\sqrt{2} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{0,n}/\sqrt{2} \\ &= \sum_{n=0}^{N-1} \frac{c_{1,2n} + c_{1,2n+1}}{\sqrt{2}} \phi_{0,n} \end{aligned}$$

which proves Equation (5.13). Equation (5.14) is proved similarly.  $\square$

In Figure 5.8 we have used Lemma 5.11 to plot the projections of  $\phi_{1,0} \in V_1$  onto  $V_0$  and  $W_0$ . It is an interesting exercise to see from the plots why exactly these functions should be least-squares approximations of  $\phi_{1,n}$ . It is also an interesting exercise to prove the following from Lemma 5.11:

**Proposition 5.12.** *Projections.*

Let  $f(t) \in V_1$ , and let  $f_{n,1}$  be the value  $f$  attains on  $[n, n + 1/2)$ , and  $f_{n,2}$  the value  $f$  attains on  $[n + 1/2, n + 1)$ . Then  $\text{proj}_{V_0}(f)$  is the function in  $V_0$  which equals  $(f_{n,1} + f_{n,2})/2$  on the interval  $[n, n + 1)$ . Moreover,  $\text{proj}_{W_0}(f)$  is the function in  $W_0$  which is  $(f_{n,1} - f_{n,2})/2$  on  $[n, n + 1/2)$ , and  $-(f_{n,1} - f_{n,2})/2$  on  $[n + 1/2, n + 1)$ .

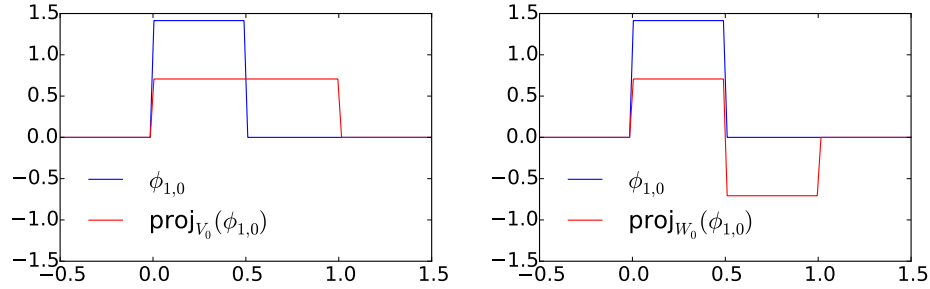


Figure 5.8: The projection of  $\phi_{1,0} \in V_1$  onto  $V_0$  and  $W_0$ .

In other words, the projection on  $V_0$  is constructed by averaging on two subintervals, while the projection on  $W_0$  is constructed by taking the difference from the mean. This sounds like a reasonable candidate for the least-squares approximations. In the exercise we generalize these observations.

In the same way as in Lemma 5.11, it is possible to show that

$$\text{proj}_{W_{m-1}}(\phi_{m,n}) = \begin{cases} \psi_{m-1,n/2}/\sqrt{2}, & \text{if } n \text{ is even;} \\ -\psi_{m-1,(n-1)/2}/\sqrt{2}, & \text{if } n \text{ is odd.} \end{cases} \quad (5.17)$$

From this it follows as before that  $\psi_m$  is an orthonormal basis for  $W_m$ . If  $\{\mathcal{B}_i\}_{i=1}^n$  are mutually independent bases, we will in the following write  $(\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_n)$  for the basis where the basis vectors from  $\mathcal{B}_i$  are included before  $\mathcal{B}_j$  when  $i < j$ . With this notation, the decomposition in Equation (5.7) can be restated as follows

**Theorem 5.13.** *Bases for  $V_m$ .*

$\phi_m$  and  $(\phi_0, \psi_0, \psi_1, \dots, \psi_{m-1})$  are both bases for  $V_m$ .

The function  $\psi$  thus has the property that its dilations and translations together span the detail components. Later we will encounter other functions, which also will be denoted by  $\psi$ , and have similar properties. In the theory of wavelets, such  $\psi$  are called *mother wavelets*. There is one important property of  $\psi$ , which we will return to:

**Observation 5.14.** *Vanishing moment.*

We have that  $\int_0^N \psi(t)dt = 0$ .

This can be seen directly from the plot in Figure 5.7, since the parts of the graph above and below the  $x$ -axis cancel. In general we say that  $\psi$  has  $k$  vanishing moments if the integrals  $\int t^l \psi(t)dt = 0$  for all  $0 \leq l \leq k - 1$ . Due to Observation 5.14,  $\psi$  has one vanishing moment. In Chapter 7 we will show that mother wavelets with many vanishing moments are very desirable when it comes to approximation of functions.

We now have all the tools needed to define the Discrete Wavelet Transform.

**Definition 5.15.** *Discrete Wavelet Transform.*

The DWT (Discrete Wavelet Transform) is defined as the change of coordinates from  $\phi_1$  to  $(\phi_0, \psi_0)$ . More generally, the  $m$ -level DWT is defined as the change of coordinates from  $\phi_m$  to  $(\phi_0, \psi_0, \psi_1, \dots, \psi_{m-1})$ . In an  $m$ -level DWT, the change of coordinates from

$$(\phi_{m-k+1}, \psi_{m-k+1}, \psi_{m-k+2}, \dots, \psi_{m-1}) \text{ to } (\phi_{m-k}, \psi_{m-k}, \psi_{m-k+1}, \dots, \psi_{m-1}) \quad (5.18)$$

is also called the  $k$ 'th stage. The ( $m$ -level) IDWT (Inverse Discrete Wavelet Transform) is defined as the change of coordinates the opposite way.

The DWT corresponds to replacing as many  $\phi$ -functions as we can with  $\psi$ -functions, i.e. replacing the original function with a sum of as much detail at different resolutions as possible. We now can state the following result.

**Theorem 5.16.** *Expression for the DWT.*

If  $g_m = g_{m-1} + e_{m-1}$  with

$$g_m = \sum_{n=0}^{2^m N-1} c_{m,n} \phi_{m,n} \in V_m,$$

$$g_{m-1} = \sum_{n=0}^{2^{m-1} N-1} c_{m-1,n} \phi_{m-1,n} \in V_{m-1} \quad e_{m-1} = \sum_{n=0}^{2^{m-1} N-1} w_{m-1,n} \psi_{m-1,n} \in W_{m-1},$$

then the change of coordinates from  $\phi_m$  to  $(\phi_{m-1}, \psi_{m-1})$  (i.e. first stage in a DWT) is given by

$$\begin{pmatrix} c_{m-1,n} \\ w_{m-1,n} \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} c_{m,2n} \\ c_{m,2n+1} \end{pmatrix} \quad (5.19)$$

Conversely, the change of coordinates from  $(\phi_{m-1}, \psi_{m-1})$  to  $\phi_m$  (i.e. the last stage in an IDWT) is given by

$$\begin{pmatrix} c_{m,2n} \\ c_{m,2n+1} \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} c_{m-1,n} \\ w_{m-1,n} \end{pmatrix} \quad (5.20)$$

*Proof.* Equations (5.5) and (5.10) say that

$$\phi_{m-1,n} = \phi_{m,2n}/\sqrt{2} + \phi_{m,2n+1}/\sqrt{2} \quad \psi_{m-1,n} = \phi_{m,2n}/\sqrt{2} - \phi_{m,2n+1}/\sqrt{2}.$$

The change of coordinate matrix from the basis  $\{\phi_{m-1,n}, \psi_{m-1,n}\}$  to  $\{\phi_{m,2n}, \phi_{m,2n+1}\}$  is thus  $\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}$ . This proves Equation (5.20). Equation (5.19) follows immediately since this matrix equals its inverse.  $\square$



Above we assumed that  $N$  is even. In Exercise 5.8 we will see how we can handle the case when  $N$  is odd.

From Theorem 5.16, we see that, if we had defined

$$\mathcal{C}_m = \{\phi_{m-1,0}, \psi_{m-1,0}, \phi_{m-1,1}, \psi_{m-1,1}, \dots, \phi_{m-1,2^{m-1}N-1}, \psi_{m-1,2^{m-1}N-1}\}. \quad (5.21)$$

i.e. we have reordered the basis vectors in  $(\phi_{m-1}, \psi_{m-1})$  (the subscript  $m$  is used since  $\mathcal{C}_m$  is a basis for  $V_m$ ), it is apparent from Equation (5.20) that  $G = P_{\phi_m \leftarrow \mathcal{C}_m}$  is the matrix where

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

is repeated along the main diagonal  $2^{m-1}N$  times. Also, from Equation (5.19) it is apparent that  $H = P_{\mathcal{C}_m \leftarrow \phi_m}$  is the same matrix. Such matrices are called *block diagonal matrices*. This particular block diagonal matrix is clearly orthogonal. Let us make the following definition.

**Definition 5.17.** *DWT and IDWT kernel transformations.*

The matrices  $H = P_{\mathcal{C}_m \leftarrow \phi_m}$  and  $G = P_{\phi_m \leftarrow \mathcal{C}_m}$  are called the *DWT and IDWT kernel transformations*. The DWT and the IDWT can be expressed in terms of these kernel transformations by

$$\text{DWT} = P_{(\phi_{m-1}, \psi_{m-1}) \leftarrow \mathcal{C}_m} H \text{ and } \text{IDWT} = G P_{\mathcal{C}_m \leftarrow (\phi_{m-1}, \psi_{m-1})},$$

respectively, where

- $P_{(\phi_{m-1}, \psi_{m-1}) \leftarrow \mathcal{C}_m}$  is a permutation matrix which groups the even elements first, then the odd elements,
- $P_{\mathcal{C}_m \leftarrow (\phi_{m-1}, \psi_{m-1})}$  is a permutation matrix which places the first half at the even indices, the last half at the odd indices.

Clearly, the kernel transformations  $H$  and  $G$  also invert each other. The point of using the kernel transformation is that they compute the output sequentially, similarly to how a filter does. Clearly also the kernel transformations are very similar to a filter, and we will return to this in the next chapter.

At each level in a DWT,  $V_k$  is split into one low-resolution component from  $V_{k-1}$ , and one detail component from  $W_{k-1}$ . We have illustrated this in figure 5.9, where the arrows represent changes of coordinates.

The detail component from  $W_{k-1}$  is not subject to further transformation. This is seen in the figure since  $\psi_{k-1}$  is a leaf node, i.e. there are no arrows going out from  $\psi_{m-1}$ . In a similar illustration for the IDWT, the arrows would go the opposite way.

The Discrete Wavelet Transform is the analogue in a wavelet setting to the Discrete Fourier transform. When applying the DFT to a vector of length  $N$ ,

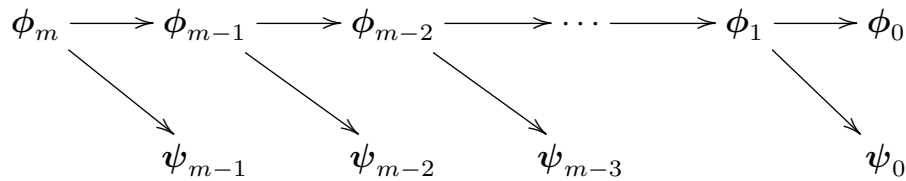


Figure 5.9: Illustration of a wavelet transform.

one starts by viewing this vector as coordinates relative to the standard basis. When applying the DWT to a vector of length  $N$ , one instead views the vector as coordinates relative to the basis  $\phi_m$ . This makes sense in light of Exercise 5.1.

### Exercise 5.1: The vector of samples is the coordinate vector

Show that the coordinate vector for  $f \in V_0$  in the basis  $\{\phi_{0,0}, \phi_{0,1}, \dots, \phi_{0,N-1}\}$  is  $(f(0), f(1), \dots, f(N-1))$ . This shows that, for  $f \in V_m$ , there is no loss of information in working with the samples of  $f$  rather than  $f$  itself.

### Exercise 5.2: Proposition 5.12

Prove Proposition 5.12.

### Exercise 5.3: Computing projections 1

In this exercise we will consider the two projections from  $V_1$  onto  $V_0$  and  $W_0$ .

- Consider the projection  $\text{proj}_{V_0}$  of  $V_1$  onto  $V_0$ . Use Lemma 5.11 to write down the matrix for  $\text{proj}_{V_0}$  relative to the bases  $\phi_1$  and  $\phi_0$ .
- Similarly, use Lemma 5.11 to write down the matrix for  $\text{proj}_{W_0} : V_1 \rightarrow W_0$  relative to the bases  $\phi_1$  and  $\psi_0$ .

### Exercise 5.4: Computing projections 2

Consider again the projection  $\text{proj}_{V_0}$  of  $V_1$  onto  $V_0$ .

- Explain why  $\text{proj}_{V_0}(\phi) = \phi$  and  $\text{proj}_{V_0}(\psi) = 0$ .
- Show that the matrix of  $\text{proj}_{V_0}$  relative to  $(\phi_0, \psi_0)$  is given by the diagonal matrix where the first half of the entries on the diagonal are 1, the second half 0.
- Show in a similar way that the projection of  $V_1$  onto  $W_0$  has a matrix relative to  $(\phi_0, \psi_0)$  given by the diagonal matrix where the first half of the entries on the diagonal are 0, the second half 1.

**Exercise 5.5: Computing projections 3**

Show that

$$\text{proj}_{V_0}(f) = \sum_{n=0}^{N-1} \left( \int_n^{n+1} f(t) dt \right) \phi_{0,n}(t) \quad (5.22)$$

for any  $f$ . Show also that the first part of Proposition 5.12 follows from this.

**Exercise 5.6: Finding the least squares error**

Show that

$$\left\| \sum_n \left( \int_n^{n+1} f(t) dt \right) \phi_{0,n}(t) - f \right\|^2 = \langle f, f \rangle - \sum_n \left( \int_n^{n+1} f(t) dt \right)^2.$$

This, together with the previous exercise, gives us an expression for the least-squares error for  $f$  from  $V_0$  (at least after taking square roots). 2DO: Generalize to  $m$

**Exercise 5.7: Projecting on  $W_0$**

Show that

$$\text{proj}_{W_0}(f) = \sum_{n=0}^{N-1} \left( \int_n^{n+1/2} f(t) dt - \int_{n+1/2}^{n+1} f(t) dt \right) \psi_{0,n}(t) \quad (5.23)$$

for any  $f$ . Show also that the second part of Proposition 5.12 follows from this.

**Exercise 5.8: When  $N$  is odd**

When  $N$  is odd, the (first stage in a) DWT is defined as the change of coordinates from  $(\phi_{1,0}, \phi_{1,1}, \dots, \phi_{1,N-1})$  to

$$(\phi_{0,0}, \psi_{0,0}, \phi_{0,1}, \psi_{0,1}, \dots, \phi_{0,(N-1)/2}, \psi_{(N-1)/2}, \phi_{0,(N+1)/2}).$$

Since all functions are assumed to have period  $N$ , we have that

$$\phi_{0,(N+1)/2} = \frac{1}{\sqrt{2}}(\phi_{1,N-1} + \phi_{1,N}) = \frac{1}{\sqrt{2}}(\phi_{1,0} + \phi_{1,N-1}).$$

From this relation one can find the last column in the change of coordinate matrix from  $\phi_0$  to  $(\phi_1, \psi_1)$ , i.e. the IDWT matrix. In particular, when  $N$  is odd, we see that the last column in the IDWT matrix circulates to the upper right corner. In terms of coordinates, we thus have that

$$c_{1,0} = \frac{1}{\sqrt{2}}(c_{0,0} + w_{0,0} + c_{0,(N+1)/2}) \quad c_{1,N-1} = \frac{1}{\sqrt{2}}c_{0,(N+1)/2}. \quad (5.24)$$

a) If  $N = 3$ , the DWT matrix equals

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

and the inverse of this is

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & -1 \\ 1 & -1 & -1 \\ 0 & 0 & 2 \end{pmatrix}.$$

Explain from this that, when  $N$  is odd, the DWT matrix can be constructed by adding a column on the form  $\frac{1}{\sqrt{2}}(-1, -1, 0, \dots, 0, 2)$  to the DWT matrices we had for  $N$  even (in the last row zeros are also added). In terms of the coordinates, we thus have the additional formulas

$$\begin{aligned} c_{0,0} &= \frac{1}{\sqrt{2}}(c_{1,0} + c_{1,1} - c_{1,N-1}) \\ w_{0,0} &= \frac{1}{\sqrt{2}}(c_{1,0} - c_{1,1} - c_{1,N-1}) \\ c_{0,(N+1)/2} &= \frac{1}{\sqrt{2}}2c_{1,N-1}. \end{aligned} \tag{5.25}$$

b) Explain that the DWT matrix is orthogonal if and only if  $N$  is even. Also explain that it is only the last column which spoils the orthogonality.

### 5.3 Implementation of the DWT and examples

The DWT is straightforward to implement: Simply iterate Equation (5.19) for  $m, m-1, \dots, 1$ . For each iteration we will use a *kernel function* which takes as input the coordinates  $(c_{m,0}, c_{m,1}, \dots)$ , and returns the coordinates  $(c_{m-1,0}, w_{m-1,0}, c_{m-1,1}, w_{m-1,1}, \dots)$ , i.e. computes one stage of the DWT, but with a different order of the coordinates than in the basis  $(\phi_m, \psi_m)$ . This turns out to be the natural ordering for computing the DWT in-place. As an example, the kernel function for the Haar wavelet can be implemented as follows (for simplicity this first version of the code assumes that  $N$  is even):

```
def dwt_kernel_haar(x, bd_mode):
    x /= sqrt(2)
    for k in range(2, len(x) - 1, 2):
        a, b = x[k] + x[k+1], x[k] - x[k+1]
        x[k], x[k+1] = a, b
```

The code above accepts two-dimensional data, just as our function `FFTImpl`. Thus, the function may be applied simultaneously to all channels in a sound. The reason for using a general kernel function will be apparent later, when we

change to different types of wavelets. It is not meant that you call this kernel function directly. Instead every time you apply the DWT call the function

```
DWTImpl(x, m, wave_name, bd_mode='symm', dual=False, transpose=False)
```

$x$  is the input to the DWT, and  $m$  is the number of levels. The three last parameters will be addressed later in the book (the `bd_mode`-parameter addresses how the boundary should be handled). The function also sets meaningful default values for the three last parameters, so that you mostly only need to provide the three first parameters.

We will later construct other wavelets, and we will distinguish them by using different names. This is the purposes of the `wave_name` parameter. This parameter is sent to a function called `find_kernel` which looks up a kernel function by name (`find_kernel` also uses the `dual` and `transpose` parameters to take a decision on which kernel to choose). The Haar wavelet is identified with the name "Haar". When this is input to `DWTImpl`, `find_kernel` returns the `dwt_kernel_haar` kernel. The kernel is then used as input to the following function:

```
def DWTImpl_internal(x, m, f, bd_mode):
    for res in range(m):
        f(x[0::2**res], bd_mode)
    reorganize_coeffs_forward(x, m)
```

The code is applied to all columns if the data is two-dimensional, and we see that the kernel function is invoked one time for each resolution. To reorder coordinates in the same order as  $(\phi_m, \psi_m)$ , note that the coordinates from  $\phi_m$  above end up at indices  $k2^m$ , where  $m$  represents the current stage, and  $k$  runs through the indices. The function `reorganize_coeffs_forward` uses this to reorder the coordinates (you will be spared the details in this implementation). Although the DWT requires this reorganization, this may not be required in practice. In Exercise 5.27 we go through some aspects of this implementation. The implementation is not recursive, as the for-loop runs through the different stages.

In this implementation, note that the first levels require the most operations, since the latter levels leave an increasing part of the coordinates unchanged. Note also that the change of coordinates matrix is a very sparse matrix: At each level a coordinate can be computed from only two of the other coordinates, so that this matrix has only two nonzero elements in each row/column. The algorithm clearly shows that there is no need to perform a full matrix multiplication to perform the change of coordinates.

There is a similar setup for the IDWT:

```
IDWTImpl(x, m, wave_name, bd_mode='symm', dual=False, transpose=False)
```

If the `wave_name`-parameter is set to "Haar", also this function will use the `find_kernel` function to look up another kernel function, `idwt_kernel_haar`

(when  $N$  is even, this uses the exact same code as `dwt_kernel_haar`. For  $N$  is odd, see Exercises 5.8 and 5.26). This is then sent as input to

```
def IDWTImpl_internal(x, m, f, bd_mode):
    reorganize_coeffs_reverse(x, m)
    for res in range(m - 1, -1, -1):
        f(x[0::2**res], bd_mode)
```

Here the steps are simply performed in the reverse order, and by iterating Equation (5.20).

In the next sections we will consider other cases where the underlying function  $\phi$  may be something else, and not necessarily piecewise constant. It will turn out that much of the analysis we have done makes sense for other functions  $\phi$  as well, giving rise to other structures which we also will refer to as wavelets. The wavelet resulting from piecewise constant functions is thus simply one example out of many, and it is commonly referred to as the *Haar wavelet*. Let us round off this section with some important examples.

### Example 5.9: Computing the DWT by hand

In some cases, the DWT can be computed by hand, keeping in mind its definition as a change of coordinates. As an example, consider the simple vector  $\mathbf{x}$  of length  $2^{10} = 1024$  defined by

$$x_n = \begin{cases} 1 & \text{for } n < 512 \\ 0 & \text{for } n \geq 512, \end{cases}$$

and let us compute the 10-level DWT of this vector by first visualizing the function with these coordinates. Since  $m = 10$  here, we should view  $\mathbf{x}$  as coordinates in the basis  $\phi_{10}$  of a function  $f(t) \in V_{10}$ . This is  $f(t) = \sum_{n=0}^{511} \phi_{10,n}$ , and since  $\phi_{10,n}$  is supported on  $[2^{-10}n, 2^{-10}(n+1))$ , the support of  $f$  has width  $512 \times 2^{-10} = 1/2$  (512 translates, each with width  $2^{-10}$ ). Moreover, since  $\phi_{10,n}$  is  $2^{10/2} = 2^5 = 32$  on  $[2^{-10}n, 2^{-10}(n+1))$  and 0 elsewhere, it is clear that

$$f(t) = \begin{cases} 32 & \text{for } 0 \leq t < 1/2 \\ 0 & \text{for } 0t \geq 1/2. \end{cases}$$

This is by definition a function in  $V_1$ :  $f$  must in fact be a multiple of  $\phi_{1,0}$ , since this also is supported on  $[0, 1/2)$ . We can thus write  $f(t) = c\phi_{1,0}(t)$  for some  $c$ . We can find  $c$  by setting  $t = 0$ . This gives that  $32 = 2^{1/2}c$  (since  $f(0) = 32$ ,  $\phi_{1,0}(0) = 2^{1/2}$ ), so that  $c = 32/\sqrt{2}$ . This means that  $f(t) = \frac{32}{\sqrt{2}}\phi_{1,0}(t)$ ,  $f$  is in  $V_1$ , and with coordinates  $(32/\sqrt{2}, 0, \dots, 0)$  in  $\phi_1$ .

When we run a 10-level DWT we make a change of coordinates from  $\phi_{10}$  to  $(\phi_0, \psi_0, \dots, \psi_9)$ . The first 9 levels give us the coordinates in  $(\phi_1, \psi_1, \psi_2, \dots, \psi_9)$ , and these are  $(32/\sqrt{2}, 0, \dots, 0)$  from what we showed. It remains thus only to perform the last level in the DWT, i.e. perform the change of coordinates from  $\phi_1$  to  $(\phi_0, \psi_0)$ . Since  $\phi_{1,0} = \frac{1}{\sqrt{2}}(\phi_{0,0} + \psi_{0,0})$ , so that we get

$$f(t) = \frac{32}{\sqrt{2}}\phi_{1,0}(t) = \frac{32}{\sqrt{2}}\frac{1}{\sqrt{2}}(\phi_{0,0} + \psi_{0,0}) = 16\phi_{0,0} + 16\psi_{0,0}.$$

From this we see that the coordinate vector of  $f$  in  $(\phi_0, \psi_0, \dots, \psi_9)$ , i.e. the 10-level DWT of  $x$ , is  $(16, 16, 0, 0, \dots, 0)$ . Note that here  $V_0$  and  $W_0$  are both 1-dimensional, since  $V_{10}$  was assumed to be of dimension  $2^{10}$  (in particular,  $N = 1$ ).

It is straightforward to verify what we found using the algorithm above:

```
x = hstack([ones(512), zeros(512)])
DWTImpl(x, 10, 'Haar')
print x
```

The reason why the method from this example worked was that the vector we started with had a simple representation in the wavelet basis, actually it equaled the coordinates of a basis function in  $\phi_1$ . Usually this is not the case, and our only possibility then is to run the DWT on a computer.

### Example 5.10: DWT on sound

Let us plot the samples of our audio sample file, and compare them with the first order DWT. Both are shown in Figure 5.10.

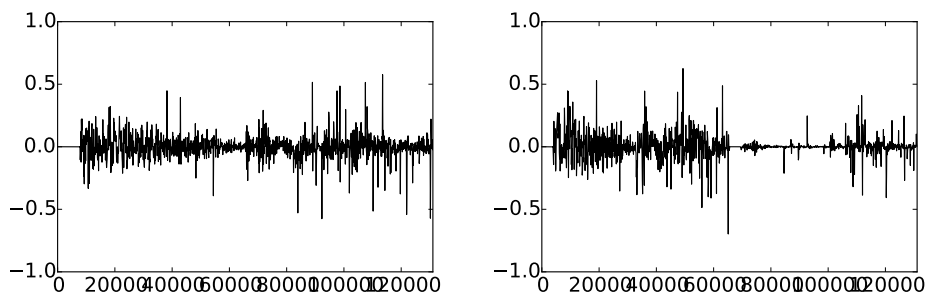


Figure 5.10: The  $2^{17}$  first sound samples (left) and the DWT coefficients (right) of the sound `castanets.wav`.

The first part of the DWT plot represents the low resolution part, the second the detail.

Since  $\phi(2^m t - n) \in V_m$  oscillates more quickly than  $\phi(t - n) \in V_0$ , one is lead to believe that coefficients from lower order resolution spaces correspond to lower frequencies. The functions  $\phi_{m,n}$  do not correspond to pure tones in the setting of wavelets, however, but let us nevertheless listen to sound from the different resolution spaces. The code base includes a function `forw_comp_rev_DWT` which runs an  $m$ -level DWT on the first samples of the audio sample file, extracts the detail or the low-resolution approximation, and runs an IDWT to reconstruct the sound. Since the returned values may lie outside the legal range  $[-1, 1]$ , the values are normalized at the end.

To listen to the low-resolution approximation, write

```
x, fs = forw_comp_rev_DWT(m, 'Haar')
play(x, fs)
```

It is instructive to run this code for different values of  $m$ . For  $m = 2$  we clearly hear a degradation in the sound. For  $m = 4$  and above most of the sound is unrecognizable, since too much of the detail is omitted. To be more precise, when listening to the sound by throwing away detail from  $W_0, W_1, \dots, W_{m-1}$ , we are left with a  $2^{-m}$  share of the data.

Let us also consider the detail. For  $m = 1$  this can be played as follows

```
x, fs = forw_comp_rev_DWT(1, 'Haar', 0)
play(x, fs)
```

We see that the detail is quite significant, so that the first order wavelet approximation does not give a very good approximation. For  $m = 2$  the detail can be played as follows

```
x, fs = forw_comp_rev_DWT(2, 'Haar', 0)
play(x, fs)
```

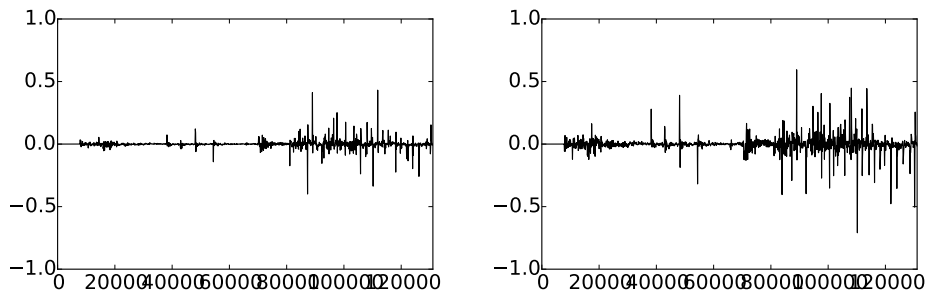


Figure 5.11: The detail in our audio sample file, for  $m = 1$  (left) and  $m = 2$  (right).

The errors are shown in Figure 5.11. The error is larger when two levels of the DWT are performed, as one would suspect. It is also seen that the error is larger in the part of the file where there are bigger variations. Since more and more information is contained in the detail components as we increase  $m$ , we here see the opposite effect: The sound gradually improves in quality as we increase  $m$ .

The previous example illustrates that wavelets as well may be used to perform operations on sound. As we will see later, however, our main application for wavelets will be images, where they have found a more important role than for sound. Images typically display variations which are less abrupt than the ones found in sound. Just as the functions above had smaller errors in the



corresponding resolution spaces than the sound had, images are thus more suited for use with wavelets. The main idea behind why wavelets are so useful comes from the fact that the detail, i.e., wavelet coefficients corresponding to the spaces  $W_k$ , are often very small. After a DWT one is therefore often left with a couple of significant coefficients, while most of the coefficients are small. The approximation from  $V_0$  can be viewed as a good approximation, even though it contains much less information. This gives another reason why wavelets are popular for images: Detailed images can be very large, but when they are downloaded to a web browser, the browser can very early show a low-resolution of the image, while waiting for the rest of the details in the image to be downloaded. When we later look at how wavelets are applied to images, we will need to handle one final hurdle, namely that images are two-dimensional.

### Example 5.11: DWT on the samples of a mathematical function

Above we plotted the DWT coefficients of a sound, as well as the detail/error. We can also experiment with samples generated from a mathematical function. Figure 5.12 plots the error for different functions, with  $N = 1024$ .

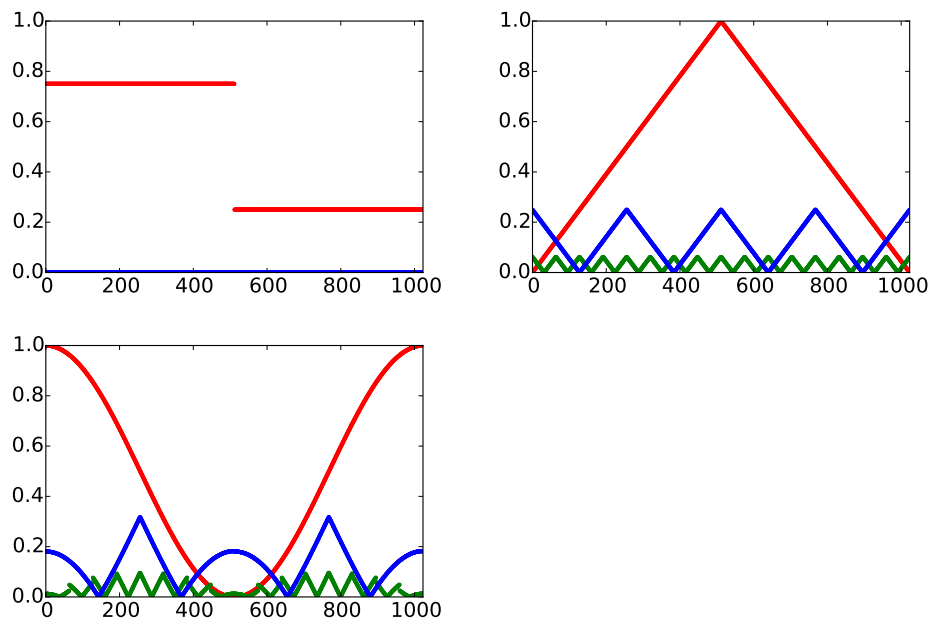


Figure 5.12: The error (i.e. the contribution from  $W_0 \oplus W_1 \oplus \dots \oplus W_{m-1}$ ) for  $N = 1024$  when  $f$  is a square wave, the linear function  $f(t) = 1 - 2|1/2 - t/N|$ , and the trigonometric function  $f(t) = 1/2 + \cos(2\pi t/N)/2$ , respectively. The detail is indicated for  $m = 6$  and  $m = 8$ .

In these cases, we see that we require large  $m$  before the detail/error becomes significant. We see also that there is no error for the square wave. The reason is that the square wave is a piecewise constant function, so that it can be represented exactly by the  $\phi$ -functions. For the other functions, however, this is not the case, so we here get an error.

### Example 5.12: Computing the wavelet coefficients

For the functions we plotted in the previous example we used the functions `DWTImpl`, `IDWTImpl` to plot the error, but it is also possible to compute the wavelet coefficients  $w_{m,n}$  exactly. You will be asked to do this in exercises 5.23 and 5.24. To exemplify the general procedure for this, consider the function  $f(t) = 1 - t/N$ . This decreases linearly from 1 to 0 on  $[0, N]$ , so that it is not piecewise constant, and does not lie in any of the spaces  $V_m$ . We can instead consider  $\text{proj}_{V_m} f \in V_m$ , and apply the DWT to this. Let us compute the  $\psi_m$ -coordinates  $w_{m,n}$  of  $\text{proj}_{V_m} f$  in the orthonormal basis  $(\phi_0, \psi_0, \psi_1, \dots, \psi_{m-1})$ . The orthogonal decomposition theorem says that

$$w_{m,n} = \langle f, \psi_{m,n} \rangle = \int_0^N f(t) \psi_{m,n}(t) dt = \int_0^N (1 - t/N) \psi_{m,n}(t) dt.$$

Using the definition of  $\psi_{m,n}$  we see that this can also be written as

$$2^{m/2} \int_0^N (1 - t/N) \psi(2^m t - n) dt = 2^{m/2} \left( \int_0^N \psi(2^m t - n) dt - \int_0^N \frac{t}{N} \psi(2^m t - n) dt \right).$$

Using Observation 5.14 we get that  $\int_0^N \psi(2^m t - n) dt = 0$ , so that the first term above vanishes. Moreover,  $\psi_{m,n}$  is nonzero only on  $[2^{-m}n, 2^{-m}(n+1))$ , and is 1 on  $[2^{-m}n, 2^{-m}(n+1/2))$ , and  $-1$  on  $[2^{-m}(n+1/2), 2^{-m}(n+1))$ . We therefore get

$$\begin{aligned} w_{m,n} &= -2^{m/2} \left( \int_{2^{-m}n}^{2^{-m}(n+1/2)} \frac{t}{N} dt - \int_{2^{-m}(n+1/2)}^{2^{-m}(n+1)} \frac{t}{N} dt \right) \\ &= -2^{m/2} \left( \left[ \frac{t^2}{2N} \right]_{2^{-m}n}^{2^{-m}(n+1/2)} - \left[ \frac{t^2}{2N} \right]_{2^{-m}(n+1/2)}^{2^{-m}(n+1)} \right) \\ &= -2^{m/2} \left( \left( \frac{2^{-2m}(n+1/2)^2}{2N} - \frac{2^{-2m}n^2}{2N} \right) - \left( \frac{2^{-2m}(n+1)^2}{2N} - \frac{2^{-2m}(n+1/2)^2}{2N} \right) \right) \\ &= -2^{m/2} \left( -\frac{2^{-2m}n^2}{2N} + \frac{2^{-2m}(n+1/2)^2}{N} - \frac{2^{-2m}(n+1)^2}{2N} \right) \\ &= -\frac{2^{-3m/2}}{2N} (-n^2 + 2(n+1/2)^2 - (n+1)^2) = \frac{1}{N2^{2+3m/2}}. \end{aligned}$$

We see in particular that  $w_{m,n} \rightarrow 0$  when  $m \rightarrow \infty$ . Also, all coordinates were equal, i.e.  $w_{m,0} = w_{m,1} = w_{m,2} = \dots$ . It is not too hard to convince oneself that this equality has to do with the fact that  $f$  is linear. We see also that there were a lot of computations even in this very simple example. For most functions we therefore usually do not compute  $w_{m,n}$  symbolically, but instead run implementations like `DWTImpl`, `IDWTImpl` on a computer.

### Exercise 5.13: Implement IDWT for The Haar wavelet

Write a function `idwt_kernel_haar` which uses the formulas (5.24) to implement the IDWT, similarly to how the function `dwt_kernel_haar` implemented the DWT using the formulas (5.25).

### Exercise 5.14: Computing projections 4

Generalize Exercise 5.4 to the projections from  $V_{m+1}$  onto  $V_m$  and  $W_m$ .

### Exercise 5.15: Scaling a function

Show that  $f(t) \in V_m$  if and only if  $g(t) = f(2t) \in V_{m+1}$ .

### Exercise 5.16: Direct sums

Let  $C_1, C_2, \dots, C_n$  be independent vector spaces, and let  $T_i : C_i \rightarrow C_i$  be linear transformations. The direct sum of  $T_1, T_2, \dots, T_n$ , written as  $T_1 \oplus T_2 \oplus \dots \oplus T_n$ , denotes the linear transformation from  $C_1 \oplus C_2 \oplus \dots \oplus C_n$  to itself defined by

$$T_1 \oplus T_2 \oplus \dots \oplus T_n(\mathbf{c}_1 + \mathbf{c}_2 + \dots + \mathbf{c}_n) = T_1(\mathbf{c}_1) + T_2(\mathbf{c}_2) + \dots + T_n(\mathbf{c}_n)$$

when  $\mathbf{c}_1 \in C_1, \mathbf{c}_2 \in C_2, \dots, \mathbf{c}_n \in C_n$ . Similarly, when  $A_1, A_2, \dots, A_n$  are square matrices,  $A_1 \oplus A_2 \oplus \dots \oplus A_n$  is defined as the block matrix where the blocks along the diagonal are  $A_1, A_2, \dots, A_n$ , and where all other blocks are 0. Show that, if  $\mathcal{B}_i$  is a basis for  $C_i$  then

$$[T_1 \oplus T_2 \oplus \dots \oplus T_n]_{(\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_n)} = [T_1]_{\mathcal{B}_1} \oplus [T_2]_{\mathcal{B}_2} \oplus \dots \oplus [T_n]_{\mathcal{B}_n},$$

Here two new concepts are used: a direct sum of matrices, and a direct sum of linear transformations.

### Exercise 5.17: Eigenvectors of direct sums

Assume that  $T_1$  and  $T_2$  are matrices, and that the eigenvalues of  $T_1$  are equal to those of  $T_2$ . What are the eigenvalues of  $T_1 \oplus T_2$ ? Can you express the eigenvectors of  $T_1 \oplus T_2$  in terms of those of  $T_1$  and  $T_2$ ?

**Exercise 5.18: Invertibility of direct sums**

Assume that  $A$  and  $B$  are square matrices which are invertible. Show that  $A \oplus B$  is invertible, and that  $(A \oplus B)^{-1} = A^{-1} \oplus B^{-1}$ .

**Exercise 5.19: Multiplying direct sums**

Let  $A, B, C, D$  be square matrices of the same dimensions. Show that  $(A \oplus B)(C \oplus D) = (AC) \oplus (BD)$ .

**Exercise 5.20: Finding  $N$** 

Assume that you run an  $m$ -level DWT on a vector of length  $r$ . What value of  $N$  does this correspond to? Note that an  $m$ -level DWT performs a change of coordinates from  $\phi_m$  to  $(\phi_0, \psi_0, \psi_1, \dots, \psi_{m-2}, \psi_{m-1})$ .

**Exercise 5.21: Different DWTs for similar vectors**

In Figure 5.13 we have plotted the DWT's of two vectors  $\mathbf{x}_1$  and  $\mathbf{x}_2$ . In both vectors we have 16 ones followed by 16 zeros, and this pattern repeats cyclically so that the length of both vectors is 256. The only difference is that the second vector is obtained by delaying the first vector with one element.

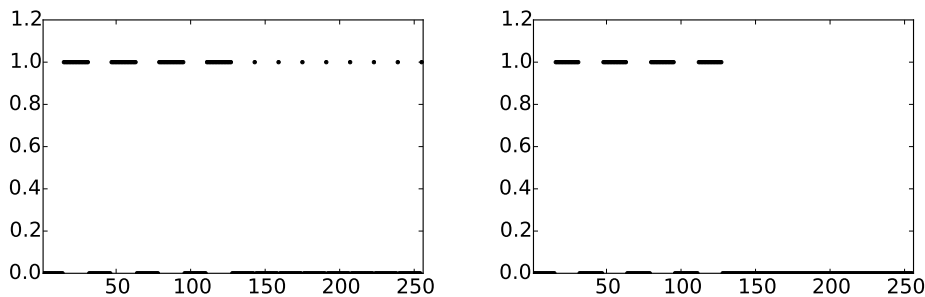


Figure 5.13: 2 vectors  $\mathbf{x}_1$  and  $\mathbf{x}_2$  which seem equal, but where the DWT's are very different.

You see that the two DWT's are very different: For the first vector we see that there is much detail present (the second part of the plot), while for the second vector there is no detail present. Attempt to explain why this is the case. Based on your answer, also attempt to explain what can happen if you change the point of discontinuity for the piecewise constant function in the left part of Figure 5.11 to something else.

**Exercise 5.22: Construct a sound**

Attempt to construct a (nonzero) sound where the low resolution approximations equal the sound itself for  $m = 1$ ,  $m = 2$ .

**Exercise 5.23: Exact computation of wavelet coefficients 1**

Compute the wavelet detail coefficients analytically for the functions in Example 5.11, i.e. compute the quantities  $w_{m,n} = \int_0^N f(t)\psi_{m,n}(t)dt$  similarly to how this was done in Example 5.12.

**Exercise 5.24: Exact computation of wavelet coefficients 2**

Compute the wavelet detail coefficients analytically for the functions  $f(t) = \left(\frac{t}{N}\right)^k$ , i.e. compute the quantities  $w_{m,n} = \int_0^N \left(\frac{t}{N}\right)^k \psi_{m,n}(t)dt$  similarly to how this was done in Example 5.12. How do these compare with the coefficients from the Exercise 5.23?

**Exercise 5.25: Computing the DWT of a simple vector**

Suppose that we have the vector  $\mathbf{x}$  with length  $2^{10} = 1024$ , defined by  $x_n = 1$  for  $n$  even,  $x_n = -1$  for  $n$  odd. What will be the result if you run a 10-level DWT on  $\mathbf{x}$ ? Use the function `DWTImpl` to verify what you have found.

**Hint.** We defined  $\psi$  by  $\psi(t) = (\phi_{1,0}(t) - \phi_{1,1}(t))/\sqrt{2}$ . From this connection it follows that  $\psi_{9,n} = (\phi_{10,2n} - \phi_{10,2n+1})/\sqrt{2}$ , and thus  $\phi_{10,2n} - \phi_{10,2n+1} = \sqrt{2}\psi_{9,n}$ . Try to couple this identity with the alternating sign you see in  $\mathbf{x}$ .

**Exercise 5.26: The Haar wavelet when  $N$  is odd**

Use the results from Exercise 5.8 to rewrite the implementations `dwt_kernel_haar` and `idwt_kernel_haar` so that they also work in the case when  $N$  is odd.

**Exercise 5.27: in-place DWT**

Show that the coordinates in  $\phi_m$  after an in-place  $m$ -level DWT end up at indices  $k2^m$ ,  $k = 0, 1, 2, \dots$ . Show similarly that the coordinates in  $\psi_m$  after an in-place  $m$ -level DWT end up at indices  $2^{m-1} + k2^m$ ,  $k = 0, 1, 2, \dots$ . Find these indices in the code for the function `reorganize_coefficients`.

## 5.4 A wavelet based on piecewise linear functions

Unfortunately, piecewise constant functions are too simple to provide good approximations. In this section we are going to extend the construction of wavelets to piecewise linear functions. The advantage is that piecewise linear functions are better for approximating smooth functions and data than piecewise constants, which should translate into smaller components (errors) in the detail spaces in many practical situations. As an example, this would be useful if we are interested in compression. In this new setting it turns out that we lose the

orthonormality we had for the Haar wavelet. On the other hand, we will see that the new scaling functions and mother wavelets are symmetric functions. We will later see that this implies that the corresponding DWT and IDWT have simple implementations with higher precision. Our experience from deriving Haar wavelets will guide us in the construction of piecewise linear wavelets. The first task is to define the new resolution spaces.

**Definition 5.18.** *Resolution spaces of piecewise linear functions.*

The space  $V_m$  is the subspace of continuous functions on  $\mathbb{R}$  which are periodic with period  $N$ , and linear on each subinterval of the form  $[n2^{-m}, (n+1)2^{-m})$ .

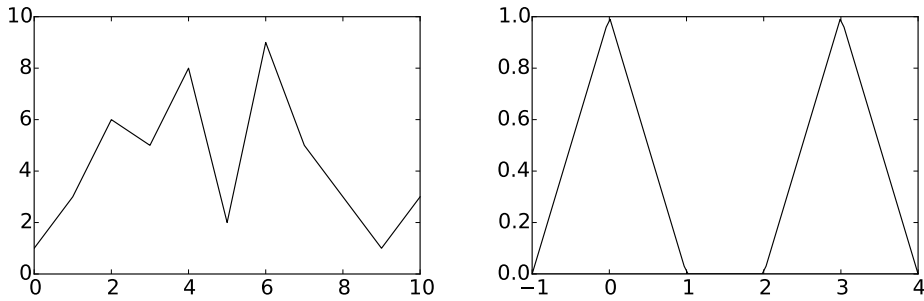


Figure 5.14: A piecewise linear function and the two functions  $\phi(t)$  and  $\phi(t-3)$ .

Any  $f \in V_m$  is uniquely determined by its values in the points  $\{2^{-m}n\}_{n=0}^{2^m N-1}$ . The linear mapping which sends  $f$  to these samples is thus an isomorphism from  $V_m$  onto  $R^{N2^m}$ , so that the dimension of  $V_m$  is  $N2^m$ . The left plot in Figure 5.14 shows an example of a piecewise linear function in  $V_0$  on the interval  $[0, 10]$ . We note that a piecewise linear function in  $V_0$  is completely determined by its value at the integers, so the functions that are 1 at one integer and 0 at all others are particularly simple and therefore interesting, see the right plot in Figure 5.14. These simple functions are all translates of each other and can therefore be built from one scaling function, as is required for a multiresolution analysis.

**Lemma 5.19.** *The function  $\phi$ .*

Let the function  $\phi$  be defined by

$$\phi(t) = \begin{cases} 1 - |t|, & \text{if } -1 \leq t \leq 1; \\ 0, & \text{otherwise;} \end{cases} \quad (5.26)$$

and for any  $m \geq 0$  set

$$\phi_{m,n}(t) = 2^{m/2} \phi(2^m t - n) \quad \text{for } n = 0, 1, \dots, 2^m N - 1,$$

and  $\phi_m = \{\phi_{m,n}\}_{n=0}^{2^m N-1}$ .  $\phi_m$  is a basis for  $V_m$ , and  $\phi_{0,n}(t)$  is the function in  $V_0$  with smallest support that is nonzero at  $t = n$ .

*Proof.* It is clear that  $\phi_{m,n} \in V_m$ , and

$$\phi_{m,n'}(n2^{-m}) = 2^{m/2}\phi(2^m(2^{-m}n) - n') = 2^{m/2}\phi(n - n').$$

Since  $\phi$  is zero at all nonzero integers, and  $\phi(0) = 1$ , we see that  $\phi_{m,n'}(n2^{-m}) = 2^{m/2}$  when  $n' = n$ , and 0 if  $n' \neq n$ . Let  $L_m : V_m \rightarrow R^{N2^m}$  be the isomorphism mentioned above which sends  $f \in V_m$  to the samples in the points  $\{2^{-m}n\}_{n=0}^{2^m N-1}$ . Our calculation shows that  $L_m(\phi_{m,n}) = 2^{m/2}e_n$ . Since  $L_m$  is an isomorphism it follows that  $\phi_m = \{\phi_{m,n}\}_{n=0}^{2^m N-1}$  is a basis for  $V_m$ .

Suppose that the function  $g \in V_0$  has smaller support than  $\phi_{0,n}$ , but is nonzero at  $t = n$ . We must have that  $L_0(g) = ce_n$  for some  $c$ , since  $g$  is zero on the integers different from  $n$ . But then  $g$  is a multiple of  $\phi_{0,n}$ , so that it is the function in  $V_0$  with smallest support that is nonzero at  $t = n$ .  $\square$

The function  $\phi$  and its translates and dilates are often referred to as hat functions for obvious reasons. Note that the new function  $\phi$  is nonzero for small negative  $x$ -values, contrary to the  $\phi$  we defined in Chapter 5. If we plotted the function on  $[0, N)$ , we would see the nonzero parts at the beginning and end of this interval, due to the period  $N$ , but we will mostly plot on an interval around zero, since such an interval captures the entire support of the function. Also for the piecewise linear wavelet the coordinates of a basis function is given by the samples:

**Lemma 5.20.** *Writing in terms of the samples.*

A function  $f \in V_m$  may be written as

$$f(t) = \sum_{n=0}^{2^m N-1} f(n/2^m)2^{-m/2}\phi_{m,n}(t). \quad (5.27)$$

An essential property also here is that the spaces are nested.

**Lemma 5.21.** *Resolution spaces are nested.*

The piecewise linear resolution spaces are nested,

$$V_0 \subset V_1 \subset \dots \subset V_m \subset \dots$$

*Proof.* We only need to prove that  $V_0 \subset V_1$  since the other inclusions are similar. But this is immediate since any function in  $V_0$  is continuous, and linear on any subinterval in the form  $[n/2, (n+1)/2)$ .  $\square$

In the piecewise constant case, we saw in Lemma 5.3 that the scaling functions were automatically orthogonal since their supports did not overlap. This is not the case in the linear case, but we could orthogonalise the basis  $\phi_m$  with the Gram-Schmidt process from linear algebra. The disadvantage is that we lose the nice local behaviour of the scaling functions and end up with basis functions that are nonzero over all of  $[0, N]$ . And for most applications, orthogonality is not essential; we just need a basis. The next step in the derivation of wavelets is to find formulas that let us express a function given in the basis  $\phi_0$  for  $V_0$  in terms of the basis  $\phi_1$  for  $V_1$ .

**Lemma 5.22.** *The two-scale equation.*

The functions  $\phi_{0,n}$  satisfy the relation

$$\phi_{0,n} = \frac{1}{\sqrt{2}} \left( \frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1} \right). \quad (5.28)$$

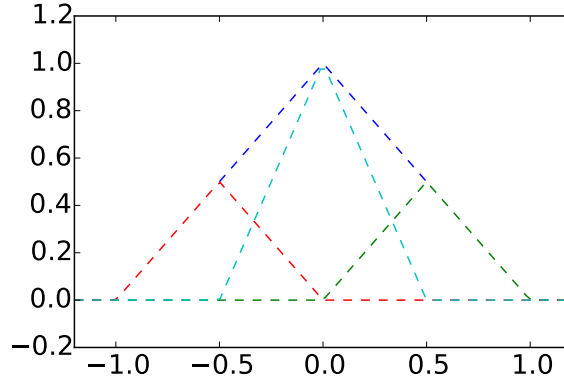


Figure 5.15: How  $\phi(t)$  can be decomposed as a linear combination of  $\phi_{1,-1}$ ,  $\phi_{1,0}$ , and  $\phi_{1,1}$ .

*Proof.* Since  $\phi_{0,n}$  is in  $V_0$  it may be expressed in the basis  $\phi_1$  with formula (5.27),

$$\phi_{0,n}(t) = 2^{-1/2} \sum_{k=0}^{2N-1} \phi_{0,n}(k/2) \phi_{1,k}(t).$$

The relation (5.28) now follows since

$$\phi_{0,n}((2n-1)/2) = \phi_{0,n}((2n+1)/2) = 1/2, \quad \phi_{0,n}(2n/2) = 1,$$

and  $\phi_{0,n}(k/2) = 0$  for all other values of  $k$ .  $\square$

The relationship given by Equation (5.28) is shown in Figure 5.15.

### 5.4.1 Detail spaces and wavelets

The next step in our derivation of wavelets for piecewise linear functions is the definition of the detail spaces. We need to determine a space  $W_0$  that is linearly independent from  $V_0$ , and so that  $V_1 = V_0 \oplus W_0$ . In the case of piecewise constant functions we started with a function  $g_1$  in  $V_1$ , computed the least squares approximation  $g_0$  in  $V_0$ , and then defined the error function  $e_0 = g_1 - g_0$ , with  $e_0 \in W_0$  and  $W_0$  as the orthogonal complement of  $V_0$  in  $V_1$ .



It turns out that this strategy is less appealing in the case of piecewise linear functions. The reason is that the functions  $\phi_{0,n}$  are not orthogonal anymore (see Exercise 5.32). Due to this we have no simple, orthogonal basis for the set of piecewise linear functions, so that the orthogonal decomposition theorem fails to give us the projection onto  $V_0$  in a simple way. It is therefore no reason to use the orthogonal complement of  $V_0$  in  $V_1$  as our error space, since it is hard to write a piecewise linear function as a sum of two other piecewise linear functions which are orthogonal. Instead of using projections to find low-resolution approximations, and orthogonal complements to find error functions, we will attempt the following simple approximation method:

**Definition 5.23.** *Alternative projection.*

Let  $g_1$  be a function in  $V_1$  given by

$$g_1 = \sum_{n=0}^{2N-1} c_{1,n} \phi_{1,n}. \quad (5.29)$$

The approximation  $g_0 = P(g_1)$  in  $V_0$  is defined as the unique function in  $V_0$  which has the same values as  $g_1$  at the integers, i.e.

$$g_0(n) = g_1(n), \quad n = 0, 1, \dots, N-1. \quad (5.30)$$

It is easy to show that  $P(g_1)$  actually is different from the projection of  $g_1$  onto  $V_0$ : If  $g_1 = \phi_{1,1}$ , then  $g_1$  is zero at the integers, and then clearly  $P(g_1) = 0$ . But in Exercise 5.31 you will be asked to compute the projection onto  $V_0$  using different means than the orthogonal decomposition theorem, and the result will be seen to be nonzero. It is also very easy to see that the coordinates of  $g_0$  in  $\phi_0$  can be obtained by dropping every second coordinate of  $g_0$  in  $\phi_1$ . To be more precise, the following holds:

**Lemma 5.24.** *Expression for the alternative projection.*

We have that

$$P(\phi_{1,n}) = \begin{cases} \sqrt{2}\phi_{0,n/2}, & \text{if } n \text{ is an even integer;} \\ 0, & \text{otherwise.} \end{cases}$$

Once this approximation method is determined, it is straightforward to determine the detail space as the space of error functions.

**Lemma 5.25.** *Resolution spaces.*

Define

$$W_0 = \{f \in V_1 \mid f(n) = 0, \quad \text{for } n = 0, 1, \dots, N-1\},$$

and

$$\psi(t) = \frac{1}{\sqrt{2}}\phi_{1,1}(t) \quad \psi_{m,n}(t) = 2^{m/2}\psi(2^m t - n). \quad (5.31)$$

Suppose that  $g_1 \in V_1$  and that  $g_0 = P(g_1)$ . Then

- the error  $e_0 = g_1 - g_0$  lies in  $W_0$ ,
- $\psi_0 = \{\psi_{0,n}\}_{n=0}^{N-1}$  is a basis for  $W_0$ .
- $V_0$  and  $W_0$  are linearly independent, and  $V_1 = V_0 \oplus W_0$ .

*Proof.* Since  $g_0(n) = g_1(n)$  for all integers  $n$ ,  $e_0(n) = (g_1 - g_0)(n) = 0$ , so that  $e_0 \in W_0$ . This proves the first statement.

For the second statement, note first that

$$\begin{aligned} \psi_{0,n}(t) &= \psi(t-n) = \frac{1}{\sqrt{2}}\phi_{1,1}(t-n) = \phi(2(t-n)-1) \\ &= \phi(2t-(2n+1)) = \frac{1}{\sqrt{2}}\phi_{1,2n+1}(t). \end{aligned} \quad (5.32)$$

$\psi_0$  is thus a linearly independent set of dimension  $N$ , since it corresponds to a subset of  $\phi_1$ . Since  $\phi_{1,2n+1}$  is nonzero only on  $(n, n+1)$ , it follows that all of  $\psi_0$  lies in  $W_0$ . Clearly then  $\psi_0$  is also a basis for  $W_0$ , since  $W_0$  also has dimension  $N$  (its image under  $L_1$  consists of points where every second component is zero).

Consider finally a linear combination from  $\phi_0$  and  $\psi_0$  which gives zero:

$$\sum_{n=0}^{N-1} a_n \phi_{0,n} + \sum_{n=0}^{N-1} b_n \psi_{0,n} = 0.$$

If we evaluate this at  $t = k$ , we see that  $\psi_{0,n}(k) = 0$ ,  $\phi_{0,n}(k) = 0$  when  $n \neq k$ , and  $\phi_{0,k}(k) = 1$ . When we evaluate at  $k$  we thus get  $a_k$ , which must be zero. If we then evaluate at  $t = k + 1/2$  we get in a similar way that all  $b_n = 0$ , and it follows that  $V_0$  and  $W_0$  are linearly independent. That  $V_1 = V_0 \oplus W_0$  follows from the fact that  $V_1$  has dimension  $2N$ , and  $V_0$  and  $W_0$  both have dimension  $N$ .  $\square$

We can define  $W_m$  in a similar way for  $m > 0$ , and generalize the lemma to  $W_m$ . We can thus state the following analog to Theorem 5.16 for writing  $g_m \in V_m$  as a sum of a low-resolution approximation  $g_{m-1} \in V_{m-1}$ , and a detail/error component  $e_{m-1} \in W_{m-1}$ .

**Theorem 5.26.** *Decomposing  $V_m$ .*

The space  $V_m$  can be decomposed as the direct sum  $V_m = V_{m-1} \oplus W_{m-1}$  where

$$W_{m-1} = \{f \in V_m \mid f(n/2^{m-1}) = 0, \text{ for } n = 0, 1, \dots, 2^{m-1}N - 1\}.$$

$W_m$  has the base  $\psi_m = \{\psi_{m,n}\}_{n=0}^{2^m N - 1}$ , and  $V_m$  has the two bases

$$\phi_m = \{\phi_{m,n}\}_{n=0}^{2^m N - 1}, \text{ and } (\phi_{m-1}, \psi_{m-1}) = (\{\phi_{m-1,n}\}_{n=0}^{2^{m-1} N - 1}, \{\psi_{m-1,n}\}_{n=0}^{2^{m-1} N - 1}).$$

With this result we can define the DWT and the IDWT with their stages as before, but the matrices themselves are now different. For the IDWT (i.e.  $P_{\phi_1 \leftarrow (\phi_0, \psi_0)}$ ), the columns in the matrix can be found from equations (5.28) and (5.32), i.e.

$$\begin{aligned}\phi_{0,n} &= \frac{1}{\sqrt{2}} \left( \frac{1}{2} \phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2} \phi_{1,2n+1} \right) \\ \psi_{0,n} &= \frac{1}{\sqrt{2}} \phi_{1,2n+1}.\end{aligned}\tag{5.33}$$

This states that

$$G = P_{\phi_m \leftarrow c_m} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1/2 & 1 & 1/2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 1/2 & 0 & 0 & 0 & \cdots & 0 & 1/2 & 1 \end{pmatrix}\tag{5.34}$$

In general we will call a matrix on the form

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda & 1 & \lambda & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \lambda & 0 & 0 & 0 & \cdots & 0 & \lambda & 1 \end{pmatrix}\tag{5.35}$$

an *elementary lifting matrix of odd type*, and denote it by  $B_\lambda$ . This results from the identity matrix by adding  $\lambda$  times the preceding and succeeding rows to the odd-indexed rows. Since the even-indexed rows are left untouched, the inverse is clearly obtained by subtracting  $\lambda$  times the preceding and succeeding rows to the odd-indexed rows, i.e.  $(B_\lambda)^{-1} = B_{-\lambda}$ . This means that the matrix for the DWT is also easily found, and

$$H = P_{c_m \leftarrow \phi_m} = \sqrt{2} B_{-1/2}\tag{5.36}$$

$$G = P_{\phi_m \leftarrow c_m} = \frac{1}{\sqrt{2}} B_{1/2}.\tag{5.37}$$

In the exercises you will be asked to implement a function `lifting_odd_symm` which computes  $B_\lambda$ . Using this the DWT kernel transformation for the piecewise linear wavelet can be applied to a vector  $\mathbf{x}$  as follows.

```
x *= sqrt(2)
lifting_odd_symm(-0.5, x, symm)
```

The IDWT kernel transformation is computed similarly. Functions `dwt_kernel_pw10`, `idwt_kernel_pw10` which perform these steps are included in the code base. The 0 stands for 0 vanishing moments. We defined vanishing moments after Observation 5.14, and we will have more to say about vanishing moments later.

### Example 5.28: DWT on sound

Using the new kernels, let us plot the listen to the new low resolution approximations, as well as plot and listen to the detail, as we did in Example 5.10. First we listen to the low-resolution approximation.

```
x, fs = forw_comp_rev_DWT(m, 'pw10')
play(x, fs)
```

There is a new and undesired effect when we increase  $m$  here: The castanet sound seems to grow strange. The sounds from the castanets are perhaps the sound with the highest frequencies.

Now for the detail. For  $m = 1$  this can be played as follows

```
x, fs = forw_comp_rev_DWT(1, 'pw10', 0)
play(x, fs)
```

For  $m = 2$  the detail can be played as follows

```
x, fs = forw_comp_rev_DWT(2, 'pw10', 0)
play(x, fs)
```

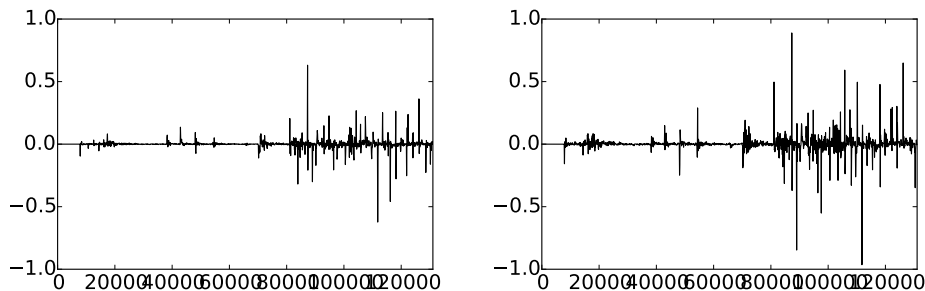


Figure 5.16: The detail in our audio sample file for the piecewise linear wavelet, for  $m = 1$  (left) and  $m = 2$  (right).

The errors are shown in Figure 5.16. When comparing with Example 5.10 we see much of the same, but it seems here that the error is bigger than before. In the next section we will try to explain why this is the case, and construct another wavelet based on piecewise linear functions which remedies this.

### Example 5.29: DWT on the samples of a mathematical function

Let us also repeat Example 5.11, where we plotted the detail/error at different resolutions for the samples of a mathematical function.

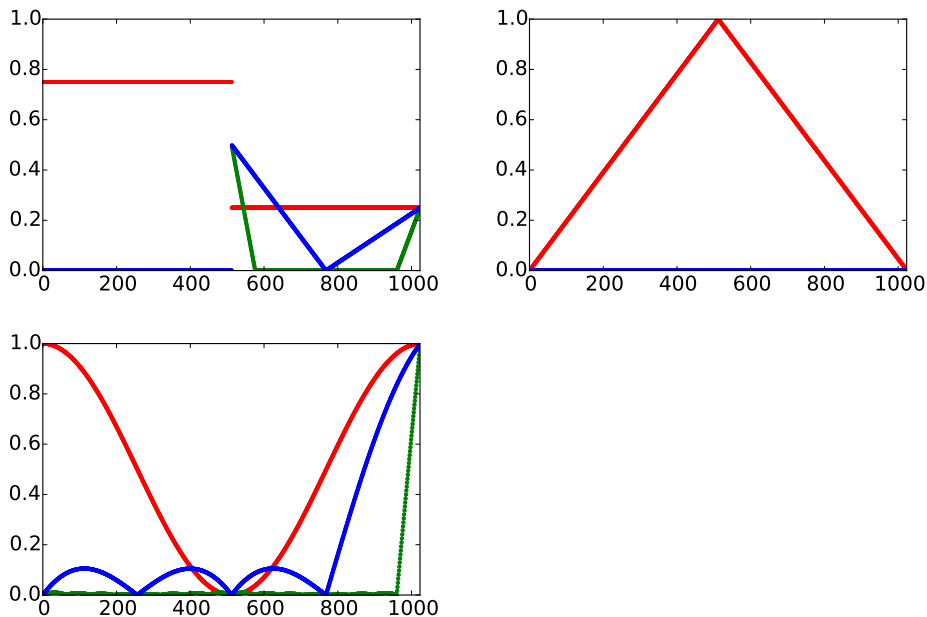


Figure 5.17: The error (i.e. the contribution from  $W_0 \oplus W_1 \oplus \dots \oplus W_{m-1}$ ) for  $N = 1025$  when  $f$  is a square wave, the linear function  $f(t) = 1 - 2|1/2 - t/N|$ , and the trigonometric function  $f(t) = 1/2 + \cos(2\pi t/N)/2$ , respectively. The detail is indicated for  $m = 6$  and  $m = 8$ .

Figure 5.17 shows the new plot. With the square wave we see now that there is an error. The reason is that a piecewise constant function can not be represented exactly by piecewise linear functions, due to discontinuity. For the second function we see that there is no error. The reason is that this function is piecewise linear, so there is no error when we represent the function from the space  $V_0$ . With the third function, however, we see an error.

### Exercise 5.30: The vector of samples is the coordinate vector $\mathbf{2}$

Show that, for  $f \in V_0$  we have that  $[f]_{\phi_0} = (f(0), f(1), \dots, f(N-1))$ . This shows that, also for the piecewise linear wavelet, there is no loss of information in working with the samples of  $f$  rather than  $f$  itself.

**Exercise 5.31: Computing projections**

In this exercise we will show how the projection of  $\phi_{1,1}$  onto  $V_0$  can be computed. We will see from this that it is nonzero, and that its support is the entire  $[0, N]$ . Let  $f = \text{proj}_{V_0} \phi_{1,1}$ , and let  $x_n = f(n)$  for  $0 \leq n < N$ . This means that, on  $(n, n+1)$ ,  $f(t) = x_n + (x_{n+1} - x_n)(t - n)$ .

- a) Show that  $\int_n^{n+1} f(t)^2 dt = (x_n^2 + x_n x_{n+1} + x_{n+1}^2)/3$ .  
 b) Show that

$$\int_0^{1/2} (x_0 + (x_1 - x_0)t) \phi_{1,1}(t) dt = 2\sqrt{2} \left( \frac{1}{12} x_0 + \frac{1}{24} x_1 \right)$$

$$\int_{1/2}^1 (x_0 + (x_1 - x_0)t) \phi_{1,1}(t) dt = 2\sqrt{2} \left( \frac{1}{24} x_0 + \frac{1}{12} x_1 \right).$$

- c) Use the fact that

$$\int_0^N (\phi_{1,1}(t) - \sum_{n=0}^{N-1} x_n \phi_{0,n}(t))^2 dt$$

$$= \int_0^1 \phi_{1,1}(t)^2 dt - 2 \int_0^{1/2} (x_0 + (x_1 - x_0)t) \phi_{1,1}(t) dt - 2 \int_{1/2}^1 (x_0 + (x_1 - x_0)t) \phi_{1,1}(t) dt$$

$$+ \sum_{n=0}^{N-1} \int_n^{n+1} (x_n + (x_{n+1} - x_n)t)^2 dt$$

and a) and b) to find an expression for  $\|\phi_{1,1}(t) - \sum_{n=0}^{N-1} x_n \phi_{0,n}(t)\|^2$ .

d) To find the minimum least squares error, we can set the gradient of the expression in c) to zero, and thus find the expression for the projection of  $\phi_{1,1}$  onto  $V_0$ . Show that the values  $\{x_n\}_{n=0}^{N-1}$  can be found by solving the equation  $S\mathbf{x} = \mathbf{b}$ , where  $S = \frac{1}{3}\{1, 4, 1\}$  is an  $N \times N$  symmetric filter, and  $\mathbf{b}$  is the vector with components  $b_0 = b_1 = \sqrt{2}/2$ , and  $b_k = 0$  for  $k \geq 2$ .

e) Solve the system in d. for some values of  $N$  to verify that the projection of  $\phi_{1,1}$  onto  $V_0$  is nonzero, and that its support covers the entire  $[0, N]$ .

**Exercise 5.32: Non-orthogonality for the piecewise linear wavelet**

Show that

$$\langle \phi_{0,n}, \phi_{0,n} \rangle = \frac{2}{3} \quad \langle \phi_{0,n}, \phi_{0,n\pm 1} \rangle = \frac{1}{6} \quad \langle \phi_{0,n}, \phi_{0,n\pm k} \rangle = 0 \text{ for } k > 1.$$

As a consequence, the  $\{\phi_{0,n}\}_n$  are neither orthogonal, nor have norm 1.

**Exercise 5.33: Implement elementary lifting steps of odd type**

Write a function

```
lifting_odd_symm(lambda, x, bd_mode)
```

which applies an elementary lifting matrix of odd type (Equation (5.35)) to  $\mathbf{x}$ . Assume that  $N$  is even. The parameter `bd_mode` should do nothing, as we will return to this parameter later. The function should not perform matrix multiplication, and apply as few multiplications as possible.

**Exercise 5.34: Wavelets based on polynomials**

The convolution of two functions defined on  $(-\infty, \infty)$  is defined by

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt.$$

Show that we can obtain the piecewise linear  $\phi$  we have defined as  $\phi = \chi_{[-1/2, 1/2]} * \chi_{[-1/2, 1/2]}$  (recall that  $\chi_{[-1/2, 1/2]}$  is the function which is 1 on  $[-1/2, 1/2]$  and 0 elsewhere). This gives us a nice connection between the piecewise constant scaling function (which is similar to  $\chi_{[-1/2, 1/2]}$ ) and the piecewise linear scaling function in terms of convolution.

## 5.5 Alternative wavelet based on piecewise linear functions

For the scaling function used for piecewise linear functions,  $\{\phi(t-n)\}_{0 \leq n < N}$  were not orthogonal anymore, contrary to the case for piecewise constant functions. We were still able to construct what we could call resolution spaces and detail spaces. We also mentioned that having many vanishing moments is desirable for a mother wavelet, and that the scaling function used for piecewise constant functions had one vanishing moment. It is easily checked, however, that the mother wavelet we now introduced for piecewise linear functions (i.e.  $\psi(t) = \frac{1}{\sqrt{2}}\phi_{1,1}(t)$ ) has no vanishing moments. Therefore, this is not a very good choice of mother wavelet. We will attempt the following adjustment strategy to construct an alternative mother wavelet  $\hat{\psi}$  which has two vanishing moments, i.e. one more than the Haar wavelet.

**Idea 5.27.** *Adjusting the wavelet construction.*

Adjust the wavelet construction in Theorem 5.26 to

$$\hat{\psi} = \psi - \alpha\phi_{0,0} - \beta\phi_{0,1} \tag{5.38}$$

and choose  $\alpha, \beta$  so that

$$\int_0^N \hat{\psi}(t) dt = \int_0^N t\hat{\psi}(t) dt = 0, \quad (5.39)$$

and define  $\psi_m = \{\hat{\psi}_{m,n}\}_{n=0}^{N2^m-1}$ , and  $W_m$  as the space spanned by  $\psi_m$ .

We thus have two free variables  $\alpha, \beta$  in Equation (5.38), to enforce the two conditions in Equation (5.39). In Exercise 5.38 you are taken through the details of solving this as two linear equations in the two unknowns  $\alpha$  and  $\beta$ , and this gives the following result:

**Lemma 5.28.** *The new function  $\psi$ .*

The function

$$\hat{\psi}(t) = \psi(t) - \frac{1}{4}(\phi_{0,0}(t) + \phi_{0,1}(t)) \quad (5.40)$$

satisfies the conditions (5.39).

Using Equation (5.28), which stated that

$$\phi_{0,n} = \frac{1}{\sqrt{2}} \left( \frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1} \right), \quad (5.41)$$

we get

$$\begin{aligned} \hat{\psi}_{0,n} &= \psi_{0,n} - \frac{1}{4}(\phi_{0,n} + \phi_{0,n+1}) \\ &= \frac{1}{\sqrt{2}}\phi_{1,2n+1} - \frac{1}{4}\frac{1}{\sqrt{2}} \left( \frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1} \right) \\ &\quad - \frac{1}{4}\frac{1}{\sqrt{2}} \left( \frac{1}{2}\phi_{1,2n+1} + \phi_{1,2n+2} + \frac{1}{2}\phi_{1,2n+3} \right) \\ &= \frac{1}{\sqrt{2}} \left( -\frac{1}{8}\phi_{1,2n-1} - \frac{1}{4}\phi_{1,2n} + \frac{3}{4}\phi_{1,2n+1} - \frac{1}{4}\phi_{1,2n+2} - \frac{1}{8}\phi_{1,2n+3} \right) \end{aligned} \quad (5.42)$$

In summary we have

$$\begin{aligned} \phi_{0,n} &= \frac{1}{\sqrt{2}} \left( \frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1} \right) \\ \hat{\psi}_{0,n} &= \frac{1}{\sqrt{2}} \left( -\frac{1}{8}\phi_{1,2n-1} - \frac{1}{4}\phi_{1,2n} + \frac{3}{4}\phi_{1,2n+1} - \frac{1}{4}\phi_{1,2n+2} - \frac{1}{8}\phi_{1,2n+3} \right), \end{aligned} \quad (5.43)$$

The new function  $\hat{\psi}$  is plotted in Figure 5.18.

We see that  $\hat{\psi}$  has support  $(-1, 2)$ , and consist of four linear segments glued together. This is in contrast with the old  $\psi$ , which was simpler in that it had the shorter support  $(0, 1)$ , and consisted of only two linear segments glued together.



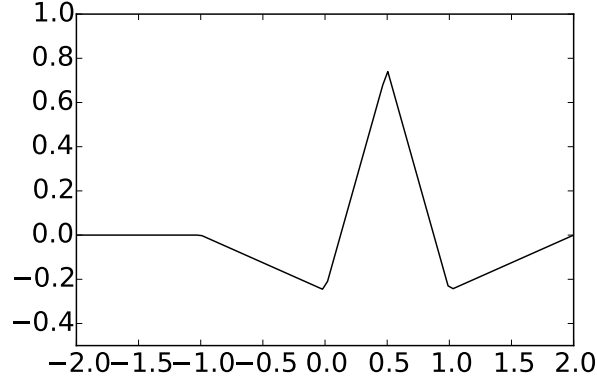


Figure 5.18: The function  $\hat{\psi}$  we constructed as an alternative wavelet for piecewise linear functions.

It may therefore seem surprising that  $\hat{\psi}$  is better suited for approximating functions than  $\psi$ . This is indeed a more complex fact, which may not be deduced by simply looking at plots of the functions.

The DWT in this new setting is the change of coordinates from  $\phi_m$  to

$$\hat{\mathcal{C}}_m = \{\phi_{m-1,0}, \hat{\psi}_{m-1,0}, \phi_{m-1,1}, \hat{\psi}_{m-1,1}, \dots, \phi_{m-1,2^{m-1}N-1}, \hat{\psi}_{m-1,2^{m-1}N-1}\}.$$

Equation (5.40) states that

$$P_{\mathcal{C}_m \leftarrow \hat{\mathcal{C}}_m} = \begin{pmatrix} 1 & -1/4 & 0 & 0 & \cdots & 0 & 0 & -1/4 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1/4 & 1 & -1/4 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1/4 & 1 & -1/4 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

(Column  $j$  for  $j$  even equals  $\mathbf{e}_j$ , since the basis functions  $\phi_{0,n}$  are not altered).

In general we will call a matrix on the form

$$\begin{pmatrix} 1 & \lambda & 0 & 0 & \cdots & 0 & 0 & \lambda \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda & 1 & \lambda & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda & 1 & \lambda \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \quad (5.44)$$

an *elementary lifting matrix of even type*, and denote it by  $A_\lambda$ . Using Equation (5.34) we can write

$$G = P_{\phi_m \leftarrow \hat{c}_m} = P_{\phi_m \leftarrow c_m} P_{c_m \leftarrow \hat{c}_m}, = \frac{1}{\sqrt{2}} B_{1/2} A_{-1/4}$$

This gives us a factorization of the IDWT in terms of lifting matrices. The inverse of elementary lifting matrices of even type can be found similarly to how we found the inverse of elementary lifting matrices of odd type, i.e.  $(A_\lambda)^{-1} = A_{-\lambda}$ .

This means that the matrix for the DWT is easily found also in this case, and

$$H = P_{\hat{c}_m \leftarrow \phi_m} = \sqrt{2} A_{1/4} B_{-1/2} \quad (5.45)$$

$$G = P_{\phi_m \leftarrow \hat{c}_m} = \frac{1}{\sqrt{2}} B_{1/2} A_{-1/4}. \quad (5.46)$$

Note that equations (5.43) also computes the matrix  $G$ , but we will rather use these factorizations, since the elementary lifting operations are already implemented in the exercises. We will also explain later why such a factorization is attractive in terms of saving computations. In the exercises you will be asked to implement a function `lifting_even_symm` which computes  $A_\lambda$ . Using this the DWT kernel transformation for the alternative piecewise linear wavelet can be applied to a vector  $\mathbf{x}$  as follows.

```
x *= sqrt(2)
lifting_odd_symm(-0.5, x, symm)
lifting_even_symm(0.25, x, symm)
```

The IDWT kernel transformation is computed similarly. Functions `dwt_kernel_pw12`, `idwt_kernel_pw12` which perform these steps are included in the code base (2 stands for 2 vanishing moments).

### Example 5.35: DWT on sound

Using the new kernels, let us also here listen to the low resolution approximations and the detail. First the low-resolution approximation:

```
x, fs = forw_comp_rev_DWT(m, 'pw12')
play(x, fs)
```

The new, undesired effect in the castanets from Example 5.28 now seem to be gone. The detail for  $m = 1$  this can be played as follows

```
x, fs = forw_comp_rev_DWT(1, 'pw12', 0)
play(x, fs)
```

For  $m = 2$  the detail can be played as follows

```
x, fs = forw_comp_rev_DWT(2, 'pw12', 0)
play(x, fs)
```

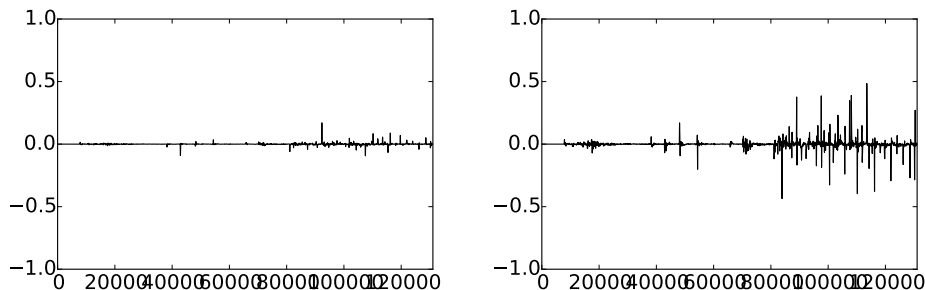


Figure 5.19: The detail in our audio sample file for the alternative piecewise linear wavelet, for  $m = 1$  (left) and  $m = 2$  (right).

The errors are shown in Figure 5.19. Again, when comparing with Example 5.10 we see much of the same. It is difficult to see an improvement from this figure. However, this figure also clearly shows a smaller error than the piecewise linear wavelet. A partial explanation is that the wavelet we now have constructed has two vanishing moments, while the other had not.

### Example 5.36: DWT on the samples of a mathematical function

Let us also repeat Exercise 5.11 for our alternative wavelet, where we plotted the detail/error at different resolutions, for the samples of a mathematical function.

Figure 5.20 shows the new plot. Again for the square wave there is an error, which seems to be slightly lower than for the previous wavelet. For the second function we see that there is no error, as before. The reason is the same as before, since the function is piecewise linear. With the third function there is an error. The error seems to be slightly lower than for the previous wavelet, which fits well with the fact that this new wavelet has a bigger number of vanishing moments.

### Exercise 5.37: Implement elementary lifting steps of even type

Write a function

```
lifting_even_symm(lambda, x, bd_mode)
```

which applies an elementary lifting matrix of even type (Equation (5.44)) to  $\mathbf{x}$ . As before, assume that  $N$  is even, and that the parameter `bd_mode` does nothing.

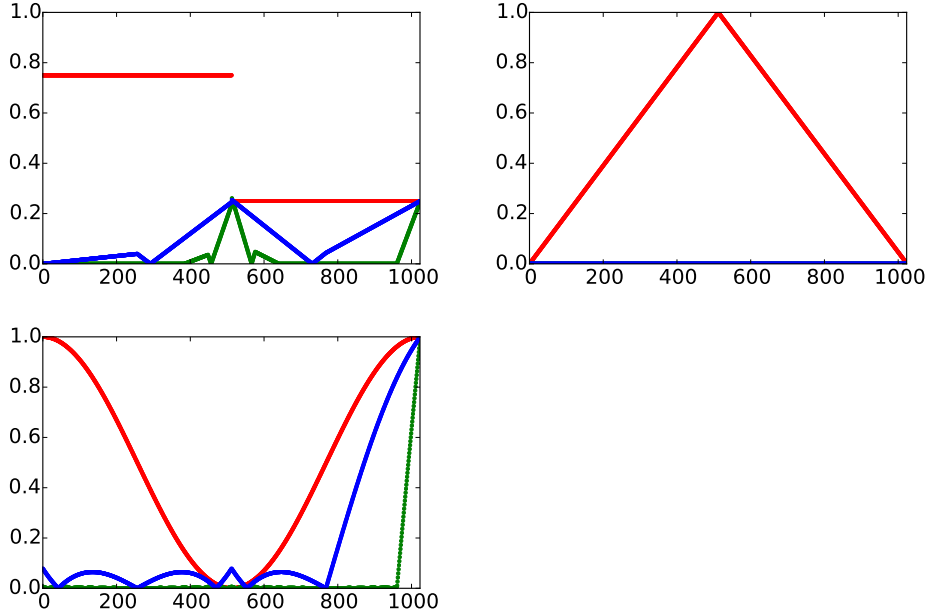


Figure 5.20: The error (i.e. the contribution from  $W_0 \oplus W_1 \oplus \dots \oplus W_{m-1}$ ) for  $N = 1025$  when  $f$  is a square wave, the linear function  $f(t) = 1 - 2|1/2 - t/N|$ , and the trigonometric function  $f(t) = 1/2 + \cos(2\pi t/N)/2$ , respectively. The detail is indicated for  $m = 6$  and  $m = 8$ .

### Exercise 5.38: Two vanishing moments

In this exercise we will show that there is a unique function on the form given by Equation (5.38) which has two vanishing moments.

a) Show that, when  $\hat{\psi}$  is defined by Equation (5.38), we have that

$$\hat{\psi}(t) = \begin{cases} -\alpha t - \alpha & \text{for } -1 \leq t < 0 \\ (2 + \alpha - \beta)t - \alpha & \text{for } 0 \leq t < 1/2 \\ (\alpha - \beta - 2)t - \alpha + 2 & \text{for } 1/2 \leq t < 1 \\ \beta t - 2\beta & \text{for } 1 \leq t < 2 \\ 0 & \text{for all other } t \end{cases}$$

b) Show that

$$\int_0^N \hat{\psi}(t) dt = \frac{1}{2} - \alpha - \beta, \quad \int_0^N t \hat{\psi}(t) dt = \frac{1}{4} - \beta.$$

c) Explain why there is a unique function on the form given by Equation (5.38) which has two vanishing moments, and that this function is given by Equation (5.40).

**Exercise 5.39: Implement finding  $\psi$  with vanishing moments**

In the previous exercise we ended up with a lot of calculations to find  $\alpha, \beta$  in Equation (5.38). Let us try to make a program which does this for us, and which also makes us able to generalize the result.

a) Define

$$a_k = \int_{-1}^1 t^k(1 - |t|)dt, \quad b_k = \int_0^2 t^k(1 - |t - 1|)dt, \quad e_k = \int_0^1 t^k(1 - 2|t - 1/2|)dt,$$

for  $k \geq 0$ . Explain why finding  $\alpha, \beta$  so that we have two vanishing moments in Equation (5.38) is equivalent to solving the following equation:

$$\begin{pmatrix} a_0 & b_0 \\ a_1 & b_1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \end{pmatrix}$$

Write a program which sets up and solves this system of equations, and use this program to verify the values for  $\alpha, \beta$  we previously have found.

**Hint.** you can integrate functions in Python with the function `quad` in the package `scipy.integrate`. As an example, the function  $\phi(t)$ , which is nonzero only on  $[-1, 1]$ , can be integrated as follows:

```
res, err = quad(lambda t: t**k*(1-abs(t)), -1, 1)
```

b) The procedure where we set up a matrix equation in a) allows for generalization to more vanishing moments. Define

$$\hat{\psi} = \psi_{0,0} - \alpha\phi_{0,0} - \beta\phi_{0,1} - \gamma\phi_{0,-1} - \delta\phi_{0,2}. \quad (5.47)$$

We would like to choose  $\alpha, \beta, \gamma, \delta$  so that we have 4 vanishing moments. Define also

$$g_k = \int_{-2}^0 t^k(1 - |t + 1|)dt, \quad d_k = \int_1^3 t^k(1 - |t - 2|)dt$$

for  $k \geq 0$ . Show that  $\alpha, \beta, \gamma, \delta$  must solve the equation

$$\begin{pmatrix} a_0 & b_0 & g_0 & d_0 \\ a_1 & b_1 & g_1 & d_1 \\ a_2 & b_2 & g_2 & d_2 \\ a_3 & b_3 & g_3 & d_3 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix},$$

and solve this with your computer.

c) Plot the function defined by (5.47), which you found in b).

**Hint.** If  $\mathbf{t}$  is the vector of  $t$ -values, and you write

$$(\mathbf{t} \geq 0) * (\mathbf{t} \leq 1) * (1 - 2 * \text{abs}(\mathbf{t} - 0.5))$$

you get the points  $\phi_{1,1}(t)$ .

d) Explain why the coordinate vector of  $\hat{\psi}$  in the basis  $(\phi_0, \psi_0)$  is

$$[\hat{\psi}]_{(\phi_0, \psi_0)} = (-\alpha, -\beta, -\delta, 0, \dots, 0 - \gamma) \oplus (1, 0, \dots, 0).$$

**Hint.** The placement of  $-\gamma$  may seem a bit strange here, and has to do with that  $\phi_{0,-1}$  is not one of the basis functions  $\{\phi_{0,n}\}_{n=0}^{N-1}$ . However, we have that  $\phi_{0,-1} = \phi_{0,N-1}$ , i.e.  $\phi(t+1) = \phi(t-N+1)$ , since we always assume that the functions we work with have period  $N$ .

e) Sketch a more general procedure than the one you found in b), which can be used to find wavelet bases where we have even more vanishing moments.

**Exercise 5.40:  $\psi$  for the Haar wavelet with two vanishing moments**

Let  $\phi(t)$  be the function we used when we defined the Haar-wavelet.

a) Compute  $\text{proj}_{V_0}(f(t))$ , where  $f(t) = t^2$ , and where  $f$  is defined on  $[0, N)$ .

b) Find constants  $\alpha, \beta$  so that  $\hat{\psi}(t) = \psi(t) - \alpha\phi_{0,0}(t) - \beta\phi_{0,1}(t)$  has two vanishing moments, i.e. so that  $\langle \hat{\psi}, 1 \rangle = 0$ , and  $\langle \hat{\psi}, t \rangle = 0$ . Plot also the function  $\hat{\psi}$ .

**Hint.** Start with computing the integrals  $\int \psi(t)dt, \int t\psi(t)dt, \int \phi_{0,0}(t)dt, \int \phi_{0,1}(t)dt,$  and  $\int t\phi_{0,0}(t)dt, \int t\phi_{0,1}(t)dt$ .

c) Express  $\phi$  and  $\hat{\psi}$  with the help of functions from  $\phi_1$ , and use this to write down the change of coordinate matrix from  $(\phi_0, \hat{\psi}_0)$  to  $\phi_1$ .

**Exercise 5.41: More vanishing moments for the Haar wavelet**

It is also possible to add more vanishing moments to the Haar wavelet. Define

$$\hat{\psi} = \psi_{0,0} - a_0\phi_{0,0} - \dots - a_{k-1}\phi_{0,k-1}.$$

Define also  $c_{r,l} = \int_l^{l+1} t^r dt$ , and  $e_r = \int_0^1 t^r \psi(t)dt$ .

a) Show that  $\hat{\psi}$  has  $k$  vanishing moments if and only if  $a_0, \dots, a_{k-1}$  solves the equation

$$\begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,k-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ c_{k-1,0} & c_{k-1,1} & \cdots & c_{k-1,k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{k-1} \end{pmatrix} \quad (5.48)$$

b) Write a function `vanishingmomshaar` which takes  $k$  as input, solves Equation (5.48), and returns the vector  $\mathbf{a} = (a_0, a_1, \dots, a_{k-1})$ .

### Exercise 5.42: Listening experiments

Run the function `forw_comp_rev_DWT` for different  $m$  for the Haar wavelet, the piecewise linear wavelet, and the alternative piecewise linear wavelet, but listen to the detail components  $W_0 \oplus W_1 \oplus \dots \oplus W_{m-1}$  instead. Describe the sounds you hear for different  $m$ , and try to explain why the sound seems to get louder when you increase  $m$ .

## 5.6 Multiresolution analysis: A generalization

Let us summarize the properties of the spaces  $V_m$ . In both our examples we showed that they were nested, i.e.

$$V_0 \subset V_1 \subset V_2 \subset \dots \subset V_m \subset \dots$$

We also showed that continuous functions could be approximated arbitrarily well from  $V_m$ , as long as  $m$  was chosen large enough. Moreover, the space  $V_0$  is closed under all translates, at least if we view the functions in  $V_0$  as periodic with period  $N$ . In the following we will always identify a function with this periodic extension, just as we did in Fourier analysis. When performing this identification, we also saw that  $f(t) \in V_m$  if and only if  $g(t) = f(2t) \in V_{m+1}$ . We have therefore shown that the scaling functions we have considered fit into the following general framework.

**Definition 5.29.** *Multiresolution analysis.*

A Multiresolution analysis, or MRA, is a nested sequence of function spaces

$$V_0 \subset V_1 \subset V_2 \subset \dots \subset V_m \subset \dots, \quad (5.49)$$

called resolution spaces, so that

- Any function can be approximated arbitrarily well from  $V_m$ , as long as  $m$  is large enough,
- $f(t) \in V_0$  if and only if  $f(2^m t) \in V_m$ ,
- $f(t) \in V_0$  if and only if  $f(t - n) \in V_0$  for all  $n$ .
- There is a function  $\phi$ , called a scaling function, so that  $\boldsymbol{\phi} = \{\phi(t - n)\}_{0 \leq n < N}$  is a basis for  $V_0$ .

When  $\boldsymbol{\phi}$  is an orthonormal basis we say that the MRA is *orthonormal*.

The wavelet of piecewise constant functions was an orthonormal MRA, while the wavelets for piecewise linear functions were not. Although the definition above states that any function can be approximated with MRA's, in practice one needs to restrict to certain functions: Certain pathological functions may be difficult to approximate. In the literature one typically requires that the function is in  $L^2(\mathbb{R})$ , and also that the scaling function and the spaces  $V_m$  are in  $L^2(\mathbb{R})$ . MRA's are much used, and one can find a wide variety of functions  $\phi$ , not only piecewise constant functions, which give rise to MRA's.

In the examples we have considered we also chose a mother wavelet. The term wavelet is used in very general terms. However, the term mother wavelet is quite concrete, and is what gives rise to the theory of wavelets. This was necessary in order to efficiently decompose the  $g_m \in V_m$  into a low resolution approximation  $g_{m-1} \in V_{m-1}$ , and a detail/error  $e_{m-1}$  in a detail space we called  $W_{m-1}$ . We have freedom in how we define these detail spaces, as well as how we define a mother wavelet whose translates span the detail space (in general we choose a mother wavelet which simplifies the computation of the decomposition  $g_m = g_{m-1} + e_{m-1}$ , but we will see later that it also is desirable to choose a  $\psi$  with other properties). Once we agree on the detail spaces and the mother wavelet, we can perform a change of coordinates to find detail and low resolution approximations. We thus have the following general recipe.

**Idea 5.30.** *Recipe for constructing wavelets.*

In order to construct MRA's which are useful for practical purposes, we need to do the following:

- Find a function  $\phi$  which can serve as the scaling function for an MRA,
- Find a function  $\psi$  so that  $\psi = \{\psi(t-n)\}_{0 \leq n < N}$  and  $\phi = \{\phi(t-n)\}_{0 \leq n < N}$  together form an orthonormal basis for  $V_1$ . The function  $\psi$  is also called a mother wavelet.

With  $V_0$  the space spanned by  $\phi = \{\phi(t-n)\}_{0 \leq n < N}$ , and  $W_0$  the space spanned by  $\psi = \{\psi(t-n)\}_{0 \leq n < N}$ ,  $\phi$  and  $\psi$  should be chosen so that we easily can compute the decomposition of  $g_1 \in V_1$  into  $g_0 + e_0$ , where  $g_0 \in V_0$  and  $e_0 \in W_0$ . If we can achieve this, the Discrete Wavelet Transform is defined as the change of coordinates from  $\phi_1$  to  $(\phi_0, \psi_0)$ .

More generally, if

$$f(t) = \sum_n c_{m,n} \phi_{m,n} = \sum_n c_{0,n} \phi_{0,n} + \sum_{m' < m, n} w_{m',n} \psi_{m',n},$$

then the  $m$ -level DWT is defined by  $\text{DWT}(\mathbf{c}_m) = (\mathbf{c}_0, \mathbf{w}_0, \dots, \mathbf{w}_{m-1})$ . It is useful to interpret  $m$  as frequency,  $n$  as time, and  $w_{m,n}$  as the contribution at frequency  $m$  and time  $n$ . In this sense, wavelets provide a *time-frequency representation* of signals. This is what can make them more useful than Fourier analysis, which only provides frequency representations.



While there are in general many possible choices of detail spaces, in the case of an orthonormal wavelet we saw that it was natural to choose the detail space  $W_{m-1}$  as the orthogonal complement of  $V_{m-1}$  in  $V_m$ , and obtain the mother wavelet by projecting the scaling function onto the detail space. Thus, for orthonormal MRA's, the low-resolution approximation and the detail can be obtained by computing projections, and the least squares approximation of  $f$  from  $V_m$  can be computed as

$$\text{proj}_{V_m}(f) = \sum_n \langle f, \phi_{m,n} \rangle \phi_{m,n}(t).$$

**Working with the samples of  $f$  rather than  $f$  itself: The first crime of wavelets.** In Exercise 5.1 we saw that for the piecewise constant wavelet the coordinate vector of  $f$  in  $\Phi_m$  equaled the sample vector of  $f$ . In Exercise 5.30 we saw that the same held for the piecewise linear wavelet. The general statement is false, however: The coordinate vector of  $f$  in  $\Phi_0$  may not equal the samples  $(f(0), f(1), \dots)$ , so that  $\sum_n f(n)\phi_{0,n}$  and  $f$  are two different functions.

In most applications, a function is usually only available through its samples. In many books on wavelets, one starts with these samples, and computes their DWT. This means that the underlying function is  $\sum_n f(n)\phi_{0,n}$ , and since this is different from  $f$  in general, we compute something completely different than we want. This shows that many books apply a wrong procedure when computing the DWT. This kind of error is also called *the first crime of wavelets*.

So, how bad is this crime? We will address this with two results. First we will see how the samples are related to the wavelet coefficients. Then we will see how the function  $\sum_s f(s/2^m)\phi_{m,s}(t)$  is related to  $f$  (wavelet crime assumes equality).

**Theorem 5.31.** *Relation between samples and wavelet coefficients.*

Assume that  $\tilde{\phi}$  has compact support and is absolutely integrable, i.e.  $\int_0^N |\tilde{\phi}(t)| dt < \infty$ . Assume also that  $f$  is continuous and has wavelet coefficients  $c_{m,n}$ . Then we have that

$$\lim_{m \rightarrow \infty} 2^{m/2} c_{m, n2^{m-m'}} = f(n/2^{m'}) \int_0^N \tilde{\phi}(t) dt.$$

*Proof.* Since  $\tilde{\phi}$  has compact support,  $\tilde{\phi}_{m, n2^{m-m'}}$  will be supported on a small interval close to  $n/2^{m'}$  for large  $m$ . Since  $f$  is continuous, given  $\epsilon > 0$  we can choose  $m$  so large that  $f = f(n/2^{m'}) + r(t)$ , where  $|r(t)| < \epsilon$  on this interval. But then

$$\begin{aligned}
 c_{m,n2^{m-m'}} &= \int_0^N f(t) \tilde{\phi}_{m,n2^{m-m'}}(t) dt \\
 &= f(n/2^{m'}) \int_0^N \tilde{\phi}_{m,n2^{m-m'}}(t) dt + \int_0^N r(t) \tilde{\phi}_{m,n2^{m-m'}}(t) dt \\
 &\leq 2^{-m/2} f(n/2^{m'}) \int_0^N \tilde{\phi}(t) dt + \epsilon \int_0^N |\tilde{\phi}_{m,n2^{m-m'}}(t)| dt \\
 &= 2^{-m/2} f(n/2^{m'}) \int_0^N \tilde{\phi}(t) dt + 2^{-m/2} \epsilon \int_0^N |\tilde{\phi}(t)| dt.
 \end{aligned}$$

From this it follows that  $\lim_{m \rightarrow \infty} 2^{m/2} c_{m,n2^{m-m'}} = f(n/2^{m'}) \int_0^N \tilde{\phi}(t) dt$ , since  $\epsilon$  was arbitrary, and  $\tilde{\phi}(t)$  was assumed to be absolutely integrable.  $\square$

This result has an important application. It turns out that there is usually no way to find analytical expressions for the scaling function and the mother wavelet. Their coordinates in  $(\phi_0, \psi_0)$  are simple, however, since there is only one non-zero coordinate:

- The coordinates of  $\phi$  in  $(\phi_0, \psi_0)$  is  $(1, 0, \dots, 0)$ , where there are  $2^m N - 1$  zeros.
- The coordinates of  $\psi$  in  $(\phi_0, \psi_0)$  is  $(0, \dots, 0, 1, 0, \dots, 0)$ , where there are  $2^{m-1} N$  zeros at the beginning.

If we know that  $\phi$  and  $\psi$  are continuous, we can apply an  $m$  stage IDWT to these coordinates and use Theorem 5.31 to find arbitrarily good estimates to the samples  $\phi(n/2^{m'})$ ,  $\psi(n/2^{m'})$ . The coordinates we find have to be scaled with  $2^{m/2}$  in order for the values to be of comparable size. Also, this algorithm will miss the actual samples by a factor of  $\int_0^N \tilde{\phi}(t) dt$ . Nevertheless, the graphs will be similar. The algorithm is also called the cascade algorithm.

**Definition 5.32.** *The cascade algorithm.*

The *cascade algorithm* applies a change of coordinates for the functions  $\phi$ ,  $\psi$  from the basis  $(\phi_0, \psi_0, \psi_1 \dots)$  to the basis  $\phi_m$ , and uses the new coordinates as an approximation to the function values of these functions.

Now for the second result.

**Theorem 5.33.** *Using the samples.*

If  $f$  is continuous and  $\phi$  has compact support, for all  $t$  with a finite bit expansion (i.e.  $t = n/2^{m'}$  for some integers  $n$  and  $m'$ ) we have that

$$\lim_{m \rightarrow \infty} 2^{-m/2} \sum_{s=0}^{2^m N - 1} f(s/2^m) \phi_{m,s}(t) = f(t) \sum_s \phi(s).$$

This says that, up to the constant factor  $c = \sum_n \phi(n)$ , the functions  $f_m \in V_m$  with coordinates  $2^{-m/2}(f(0/2^m), f(1/2^m), \dots)$  in  $\Phi_m$  converge pointwise to  $f$  as  $m \rightarrow \infty$  (even though the samples of  $f_m$  may not equal those of  $f$ ).

*Proof.* With  $t = n/2^{m'}$ , for  $m > m'$  we have that

$$\phi_{m,s}(t) = \phi_{m,s}(n2^{m-m'}/2^m) = 2^{m/2}\phi(2^m n2^{m-m'}/2^m - s) = 2^{m/2}\phi(n2^{m-m'} - s).$$

We thus have that

$$\sum_{s=0}^{2^m N-1} 2^{-m/2} f(s/2^m) \phi_{m,s}(t) = \sum_{s=0}^{2^m N-1} f(s/2^m) \phi(n2^{m-m'} - s).$$

In the sum finitely many  $s$  close to  $n2^{m-m'}$  contribute (due to finite support), and then  $s/2^m \approx t$ . Due to continuity of  $f$  this sum thus converges to  $f(t) \sum_s \phi(s)$ , and the proof is done.  $\square$

Let us see how we can implement the cascade algorithm. As input to the algorithm we must have the number of levels  $m$ , and the kernel to use for the IDWT. Also we need to know an interval  $[a, b]$  so large that it contains the supports of  $\phi, \psi$  (we will see later how we can compute this supports). Else we can not obtain complete plots of the functions.  $a$  and  $b$  thus also need to be input to the algorithm. We now set  $N = b - a$ . The space  $\phi_m$  is then of dimension  $(b - a)2^m$ , so the cascade algorithm needs a coordinate vector of this size as starting point. the coordinates of  $\phi$  in the  $(b - a)2^m$ -dimensional basis  $(\phi_0, \psi_0, \psi_1 \dots)$  is  $(1, 0, \dots, 0)$  while the coordinates of  $\psi$  in the same basis is

$$\left( \underbrace{0, \dots, 0}_{b-a \text{ times}}, 1, 0, \dots, 0 \right).$$

Our algorithm can take as input whether we want to plot the  $\phi$  or the  $\psi$  function (and thereby choose among these sets of coordinates), and also the value of the dual parameter, which we will return to. The following algorithm can be used for all this

```
def cascade_alg(m, a, b, wave_name, scaling, dual):
    coords = zeros((b-a)*2**m)
    if scaling:
        coords[0] = 1
    else:
        coords[b - a] = 1

    t = linspace(a, b, (b-a)*2**m)
    IDWTImpl(coords, m, wave_name, 'per', dual)
    coords = concatenate([coords[(b*2**m):((b-a)*2**m)], \
                          coords[0:(b*2**m)]])
    plt.figure()
    plt.plot(t, 2**(m/2.)*coords, 'k-')
```

**Example 5.43: Implementing the cascade algorithm**

One thing should be noted in the function `cascade_alg`. As the scaling function of the piecewise linear wavelet, it may be that the function is nonzero for small, negative values. If we plot the function over  $[0, N]$ , we would see two disconnected segments - one to the left, and one to the right. In the code we shift the values so that the graph appears as one connected segment.

We will use  $a = -2$  and  $b = 6$  in what follows, since  $[-2, 6]$  will turn out to contain all supports. We will also use  $m = 10$  levels in the cascade algorithm. The following code then runs the cascade algorithm for the three wavelets we have considered, to reproduce all previous scaling functions and mother wavelets.

```
cascade_alg(10, -2, 6, 'Haar', True, False)
cascade_alg(10, -2, 6, 'Haar', False, False)
```

```
cascade_alg(10, -2, 6, 'pw10', True, False)
cascade_alg(10, -2, 6, 'pw10', False, False)
```

```
cascade_alg(10, -2, 6, 'pw12', True, False)
cascade_alg(10, -2, 6, 'pw12', False, False)
```

## 5.7 Summary

We started this chapter by motivating the theory of wavelets as a different function approximation scheme, which solved some of the shortcomings of Fourier series. While one approximates functions with trigonometric functions in Fourier theory, with wavelets one instead approximates a function in several stages, where one at each stage attempts to capture information at a given resolution, using a function prototype. This prototype is localized in time, contrary to the Fourier basis functions, and this makes the theory of wavelets suitable for time-frequency representations of signals. We used an example based on Google Earth to illustrate that the wavelet-based scheme can represent an image at different resolutions in a scalable way, so that passing from one resolution to another simply amounts to adding some detail information to the lower resolution version of the image. This also made wavelets useful for compression, since the images at different resolutions can serve as compressed versions of the image.

We defined the simplest wavelet, the Haar wavelet, which is a function approximation scheme based on piecewise constant functions, and deduced its properties. We defined the Discrete Wavelet Transform (DWT) as a change of coordinates corresponding to the function spaces we defined. This transform is the crucial object to study when it comes to more general wavelets also, since it is the object which makes wavelets useful for computation. In the following chapters, we will see that reordering of the source and target bases of the

DWT will aid in expressing connections between wavelets and filters, and in constructing optimized implementations of the DWT.

We then defined another wavelet, which corresponded to a function approximation scheme based on piecewise linear functions, instead of piecewise constant functions. There were several differences with the new wavelet when compared to the previous one. First of all, the basis functions were not orthonormal, and we did not attempt to make them orthonormal. The resolution spaces we now defined were not defined in terms of orthogonal bases, and we had some freedom on how we defined the detail spaces, since they are not defined as orthogonal complements anymore. Similarly, we had some freedom on how we define the mother wavelet, and we mentioned that we could define it so that it is more suitable for approximation of functions, by adding what we called vanishing moments.

From these examples of wavelets and their properties we made a generalization to what we called a multiresolution analysis (MRA). In an MRA we construct successively refined spaces of functions that may be used to approximate functions arbitrarily well. We will continue in the next chapter to construct even more general wavelets, within the MRA framework.

The book [29] goes through developments for wavelets in detail. While wavelets have been recognized for quite some time, it was with the important work of Daubechies [12, 13] that they found new arenas in the 80's. Since then they found important applications. The main application we will focus on in later chapters is image processing.

#### What you should have learned in this chapter.

- Definition of resolution spaces ( $V_m$ ), detail spaces ( $W_m$ ), scaling function ( $\phi$ ), and mother wavelet ( $\psi$ ) for the wavelet based on piecewise constant functions.
- The nesting of resolution spaces, and how one can project from one resolution space onto a lower order resolution space, and onto its orthogonal complement.
- The definition of the Discrete Wavelet Transform as a change of coordinates, and how this can be written down from relations between basis functions.
- Definition of the  $m$ -level Discrete Wavelet Transform.
- Implementation of the Haar wavelet transform and its inverse.
- Experimentation with wavelets on sound.
- Definition of scaling function, mother wavelet, resolution spaces, and detail spaces for the wavelet of piecewise linear functions.
- How one alters the mother wavelet for piecewise linear functions, in order to add a vanishing moment.
- Definition of a multiresolution analysis.

## Chapter 6

# The filter representation of wavelets

Previously we saw that analog filters restricted to the Fourier spaces gave rise to digital filters. These digital filters sent the samples of the input function to the samples of the output function, and are easily implementable, in contrast to the analog filters. We have also seen that wavelets give rise to analog filters. This leads us to believe that the DWT also can be implemented in terms of digital filters. In this chapter we will prove that this is in fact the case.

There are some differences between the Fourier and wavelet settings, however:

- The DWT is not constructed by looking at the samples of a function, but rather by looking at coordinates in a given basis.
- The function spaces we work in (i.e.  $V_m$ ) are different from the Fourier spaces.
- The DWT gave rise to two different types of analog filters: The filter defined by Equation (7.16) for obtaining  $c_{m,n}$ , and the filter defined by Equation (7.17) for obtaining  $w_{m,n}$ . We want both to correspond to digital filters.

Due to these differences, the way we realize the DWT in terms of filters will be a bit different. Despite the differences, this chapter will make it clear that the output of a DWT can be interpreted as the combined output of two different filters, and each filter will have an interpretation in terms of frequency representations. We will also see that the IDWT has a similar interpretation in terms of filters.

In this chapter we will also see that expressing the DWT in terms of filters will also enable us to define more general transforms, where even more filters are used. It is fruitful to think about each filter as concentrating on a particular frequency range, and that these transforms thus simply splits the input into different frequency bands. Such transforms have important applications to the

processing and compression of sound, and we will show that the much used MP3 standard for compression of sound takes use of such transforms.

## 6.1 The filters of a wavelet transformation

We will make the connection with digital filters by looking again at the different examples of wavelet bases we have seen: In each case we saw that every second row/column in the kernel transformations  $G = P_{\phi_m \leftarrow c_m}$  and  $H = P_{c_m \leftarrow \phi_m}$  repeated, as in a circulant matrix. The matrices were not exactly circulant Toeplitz matrices, however, since there are two different columns repeating. The change of coordinate matrices occuring in the stages in a DWT are thus not digital filters, but they seem to be related. Let us start by giving these new matrices names:

**Definition 6.1.** *MRA-matrices.*

An  $N \times N$ -matrix  $T$ , with  $N$  even, is called an MRA-matrix if the columns are translates of the first two columns in alternating order, in the same way as the columns of a circulant Toeplitz matrix.

From our previous calculations it is clear that, once  $\phi$  and  $\psi$  are given through an MRA, the corresponding change of coordinate matrices will always be MRA-matrices. The MRA-matrices is our connection between filters and wavelets. Let us make the following definition:

**Definition 6.2.**  *$H_0$  and  $H_1$ .*

We denote by  $H_0$  the (unique) filter with the same first row as  $H$ , and by  $H_1$  the (unique) filter with the same second row as  $H$ .  $H_0$  and  $H_1$  are also called the *DWT filter components*.

Using this definition it is clear that

$$(Hc_m)_k = \begin{cases} (H_0c_m)_k & \text{when } k \text{ is even} \\ (H_1c_m)_k & \text{when } k \text{ is odd,} \end{cases}$$

since the left hand side depends only on row  $k$  in the matrix  $H$ , and this is equal to row  $k$  in  $H_0$  (when  $k$  is even) or row  $k$  in  $H_1$  (when  $k$  is odd). This means that  $Hc_m$  can be computed with the help of  $H_0$  and  $H_1$  as follows:

**Theorem 6.3.** *DWT expressed in terms of filters.*

Let  $c_m$  be the coordinates in  $\phi_m$ , and let  $H_0, H_1$  be defined as above. Any stage in a DWT can be implemented in terms of filters as follows:

- Compute  $H_0c_m$ . The even-indexed entries in the result are the coordinates  $c_{m-1}$  in  $\phi_{m-1}$ .
- Compute  $H_1c_m$ . The odd-indexed entries in the result are the coordinates  $w_{m-1}$  in  $\psi_{m-1}$ .

This gives an important connection between wavelets and filters: The DWT corresponds to applying two filters,  $H_0$  and  $H_1$ , and the result from the DWT is produced by assembling half of the coordinates from each. Keeping only every second coordinate is called *downsampling* (with a factor of two). Had we not performed downsampling, we would have ended up with twice as many coordinates as we started with. Downsampling with a factor of two means that we end up with the same number of samples as we started with. We also say that the output of the two filters is *critically sampled*. Due to the critical sampling, it is inefficient to compute the full application of the filters. We will return to the issue of making efficient implementations of critically sampled filter banks later.

We can now complement Figure 5.9 by giving names to the arrows as follows:

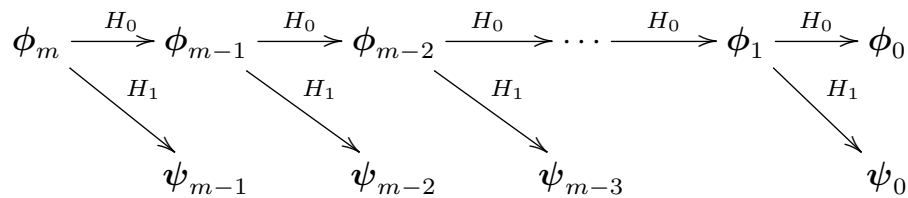


Figure 6.1: Detailed illustration of a wavelet transform.

Let us make a similar analysis for the IDWT, and let us first make the following definition:

**Definition 6.4.**  $G_0$  and  $G_1$ .

We denote by  $G_0$  the (unique) filter with the same first column as  $G$ , and by  $G_1$  the (unique) filter with the same second column as  $G$ .  $G_0$  and  $G_1$  are also called the *IDWT filter components*.

These filters are uniquely determined, since any filter is uniquely determined from one of its columns. We can now write



$$\begin{aligned}
 \mathbf{c}_m &= G \begin{pmatrix} c_{m-1,0} \\ w_{m-1,0} \\ c_{m-1,1} \\ w_{m-1,1} \\ \dots \\ c_{m-1,2^{m-1}N-1} \\ w_{m-1,2^{m-1}N-1} \end{pmatrix} = G \left( \begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \dots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \dots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix} \right) \\
 &= G \begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \dots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + G \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \dots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix} \\
 &= G_0 \begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \dots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + G_1 \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \dots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix}.
 \end{aligned}$$

Here we have split a vector into its even-indexed and odd-indexed elements, which correspond to the coefficients from  $\phi_{m-1}$  and  $\psi_{m-1}$ , respectively. In the last equation, we replaced with  $G_0, G_1$ , since the multiplications with  $G$  depend only on the even and odd columns in that matrix (due to the zeros inserted), and these columns are equal in  $G_0, G_1$ . We can now state the following characterization of the inverse Discrete Wavelet transform:

**Theorem 6.5.** *IDWT expressed in terms of filters.*

Let  $G_0, G_1$  be defined as above. Any stage in an IDWT can be implemented in terms of filters as follows:

$$\mathbf{c}_m = G_0 \begin{pmatrix} c_{m-1,0} \\ 0 \\ c_{m-1,1} \\ 0 \\ \dots \\ c_{m-1,2^{m-1}N-1} \\ 0 \end{pmatrix} + G_1 \begin{pmatrix} 0 \\ w_{m-1,0} \\ 0 \\ w_{m-1,1} \\ \dots \\ 0 \\ w_{m-1,2^{m-1}N-1} \end{pmatrix}. \quad (6.1)$$

Making a new vector where zeroes have been inserted in this way is also called *upsampling* (with a factor of two). We can now also complement Figure 5.9 for the IDWT with named arrows. This has been done in Figure 6.2

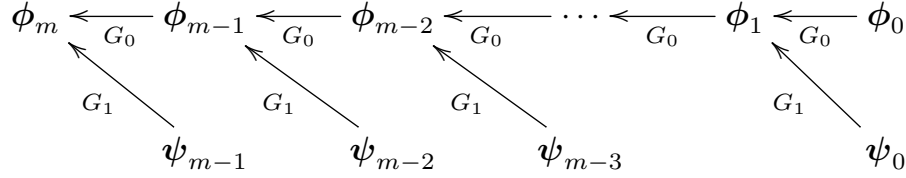


Figure 6.2: Detailed illustration of an IDWT.

Note that the filters  $G_0, G_1$  were defined in terms of the columns of  $G$ , while the filters  $H_0, H_1$  were defined in terms of the rows of  $H$ . This difference is seen from the computations above to come from that the change of coordinates one way splits the coordinates into two parts, while the inverse change of coordinates performs the opposite. Let us summarize what we have found as follows.

**Fact 6.6.** *Computing DWT/IDWT through filters.*

The DWT can be computed with the help of two filters  $H_0, H_1$ , as explained in Theorem 6.3. Any linear transformation computed from two filters  $H_0, H_1$  in this way is called a *forward filter bank transform*. The IDWT can be computed with the help of two filters  $G_0, G_1$  as explained in Theorem 6.5. Any linear transformation computed from two filters  $G_0, G_1$  in this way is called a *reverse filter bank transform*.

In Chapter 8 we will go through how any forward and reverse filter bank transform can be implemented, once we have the filters  $H_0, H_1, G_0$ , and  $G_1$ . When we are in a wavelet setting, the filter coefficients in these four filters can be found from the relations between the bases  $\phi_1$  and  $(\phi_0, \psi_0)$ . The filters  $H_0, H_1, G_0, G_1$  can also be constructed from outside a wavelet setting, i.e. that they do not originate from change of coordinate matrices between certain function bases. The important point is that the matrices invert each other, but in a signal processing setting it may also be meaningful to allow for the reverse transform not to invert the forward transform exactly. This corresponds to some loss of information when we attempt to reconstruct the original signal using the reverse transform. A small such loss can, as we will see at the end of this chapter, be acceptable.

Note that Figure 6.1 and 6.2 do not indicate the additional downsampling and upsampling steps described in Theorem 6.3 and 6.5. If we indicate downsampling with  $\downarrow_2$ , and upsampling with  $\uparrow_2$ , the algorithms given in Theorem 6.3 and 6.5 can be summarized as in Figure 6.3.

Here  $\oplus$  represents summing the elements which point inwards to the plus sign. In this figure, the left side represents the DWT, the right side the IDWT. In the literature, wavelet transforms are more often illustrated in this way using filters, since it makes all steps involved in the process more clear. This type of figure also opens for generalization. We will shortly look into this.

There are several reasons why it is smart to express a wavelet transformation in terms of filters. First of all, it enables us to reuse theoretical results from

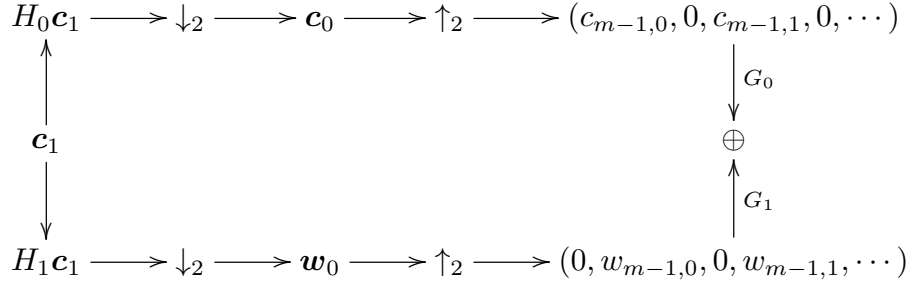


Figure 6.3: Detailed illustration of a DWT.

the world of filters in the world of wavelets, and to give useful interpretations of the wavelet transform in terms of frequencies. Secondly, and perhaps most important, it enables us to reuse efficient implementations of filters in order to compute wavelet transformations. A lot of work has been done in order to establish efficient implementations of filters, due to their importance.

In Example 5.10 we argued that the elements in  $V_{m-1}$  correspond to frequencies at lower frequencies than those in  $V_m$ , since  $V_0 = \text{Span}(\{\phi_{0,n}\}_n)$  should be interpreted as content of lower frequency than the  $\phi_{1,n}$ , with  $W_0 = \text{Span}(\{\psi_{0,n}\}_n)$  the remaining high frequency detail. To elaborate more on this, we have that

$$\phi(t) = \sum_{n=0}^{2N-1} (G_0)_{n,0} \phi_{1,n}(t) \tag{6.2}$$

$$\psi(t) = \sum_{n=0}^{2N-1} (G_1)_{n-1,1} \phi_{1,n}(t), \tag{6.3}$$

where  $(G_k)_{i,j}$  are the entries in the matrix  $G_k$ . Similar equations are true for  $\phi(t - k), \psi(t - k)$ . Due to Equation (6.2), the filter  $G_0$  should have lowpass characteristics, since it extracts the information at lower frequencies. Similarly,  $G_1$  should have highpass characteristics due to Equation (6.3).

**Example 6.1: The Haar wavelet**

For the Haar wavelet we saw that, in  $G$ , the matrix

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{6.4}$$

repeated along the diagonal. The filters  $G_0$  and  $G_1$  can be found directly from these columns:

$$G_0 = \{1/\sqrt{2}, 1/\sqrt{2}\}$$

$$G_1 = \{1/\sqrt{2}, -1/\sqrt{2}\}.$$

We have seen these filters previously:  $G_0$  is a moving average filter of two elements (up to multiplication with a constant). This is a lowpass filter.  $G_1$  is a bass-reducing filter, which is a high-pass filter. Up to a constant, this is also an approximation to the derivative. Since  $G_1$  is constructed from  $G_0$  by adding an alternating sign to the filter coefficients, we know from before that  $G_1$  is the high-pass filter corresponding to the low-pass filter  $G_0$ , so that the frequency response of the second is given by a shift of frequency with  $\pi$  in the first. The frequency responses are

$$\lambda_{G_0}(\omega) = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}e^{-i\omega} = \sqrt{2}e^{-i\omega/2} \cos(\omega/2)$$

$$\lambda_{G_1}(\omega) = \frac{1}{\sqrt{2}}e^{i\omega} - \frac{1}{\sqrt{2}} = \sqrt{2}ie^{i\omega/2} \sin(\omega/2).$$

By considering the filters where the rows are as in Equation (6.4), it is clear that

$$H_0 = \{1/\sqrt{2}, 1/\sqrt{2}\}$$

$$H_1 = \{-1/\sqrt{2}, 1/\sqrt{2}\},$$

so that the frequency responses for the DWT have the same lowpass/highpass characteristics.

**Example 6.2: Wavelet for piecewise linear functions**

For the wavelet for piecewise linear functions we looked at in the previous section, Equation (5.34) gives that

$$G_0 = \frac{1}{\sqrt{2}}\{1/2, \underline{1}, 1/2\}$$

$$G_1 = \frac{1}{\sqrt{2}}\{\underline{1}\}. \tag{6.5}$$

$G_0$  is again a filter we have seen before: Up to multiplication with a constant, it is the treble-reducing filter with values from row 2 of Pascal’s triangle. We see something different here when compared to the Haar wavelet, in that the filter  $G_1$  is not the highpass filter corresponding to  $G_0$ . The frequency responses are now

$$\begin{aligned}\lambda_{G_0}(\omega) &= \frac{1}{2\sqrt{2}}e^{i\omega} + \frac{1}{\sqrt{2}} + \frac{1}{2\sqrt{2}}e^{-i\omega} = \frac{1}{\sqrt{2}}(\cos \omega + 1) \\ \lambda_{G_1}(\omega) &= \frac{1}{\sqrt{2}}.\end{aligned}$$

$\lambda_{G_1}(\omega)$  thus has magnitude  $\frac{1}{\sqrt{2}}$  at all points. Comparing with Figure 6.5 we see that here also the frequency response has a zero at  $\pi$ . The frequency response seems also to be flatter around  $\pi$ . For the DWT we have that

$$\begin{aligned}H_0 &= \sqrt{2}\{\underline{1}\} \\ H_1 &= \sqrt{2}\{-1/2, \underline{1}, -1/2\}.\end{aligned}\tag{6.6}$$

Even though  $G_1$  was not the highpass filter corresponding to  $G_0$ , we see that, up to a constant,  $H_1$  is (it is a bass-reducing filter with values taken from row 2 of Pascals triangle).

**Example 6.3: The alternative piecewise linear wavelet**

We previously wrote down the first two columns in  $P_{\phi_m \leftarrow c_m}$  for the alternative piecewise linear wavelet. This gives us that the filters  $G_0$  and  $G_1$  are

$$\begin{aligned}G_0 &= \frac{1}{\sqrt{2}}\{1/2, \underline{1}, 1/2\} \\ G_1 &= \frac{1}{\sqrt{2}}\{-1/8, -1/4, \underline{3/4}, -1/4, -1/8\}.\end{aligned}\tag{6.7}$$

Here  $G_0$  was as for the wavelet of piecewise linear functions since we use the same scaling function.  $G_1$  was changed, however. Clearly,  $G_1$  now has highpass characteristics, while the lowpass characteristic of  $G_0$  has been preserved.

The filters  $G_0, G_1, H_0, H_1$  are particularly important in applications: Apart from the scaling factors  $1/\sqrt{2}, \sqrt{2}$  in front, we see that the filter coefficients are all dyadic fractions, i.e. they are on the form  $\beta/2^j$ . Arithmetic operations with dyadic fractions can be carried out exactly on a computer, due to representations as binary numbers in computers. These filters are thus important in applications, since they can be used as transformations for lossless coding. The same argument can be made for the Haar wavelet, but this wavelet had one less vanishing moment.

Note that the role of  $H_1$  as the high-pass filter corresponding to  $G_0$  is the case in both previous examples. We will prove in the next chapter that this is a much more general result which holds for all wavelets, not only for the orthonormal ones.

### 6.1.1 The dual filter bank transform and the dual parameter

Since the reverse transform inverts the forward transform,  $GH = I$ . If we transpose this expression we get that  $H^T G^T = I$ . Clearly  $H^T$  is a reverse filter bank transform with filters  $(H_0)^T, (H_1)^T$ , and  $G^T$  is a forward filter bank transform with filters  $(G_0)^T, (G_1)^T$ . Due to their usefulness, these transforms have their own name:

**Definition 6.7.** *Dual filter bank transforms.*

Assume that  $H_0, H_1$  are the filters of a forward filter bank transform, and that  $G_0, G_1$  are the filters of a reverse filter bank transform. By the *dual transforms* we mean the forward filter bank transform with filters  $(G_0)^T, (G_1)^T$ , and the reverse filter bank transform with filters  $(H_0)^T, (H_1)^T$ .

In other words, if  $H$  and  $G$  are the kernel transformations of the DWT and the IDWT, respectively, the kernel transformations of the dual DWT and the dual IDWT are  $G^T$  and  $H^T$ , respectively. In Section 5.3 we used a parameter `dual` in our call to the DWT and IDWT kernel functions. This parameter can now be explained as follows:

**Fact 6.8.** *The `dual`-parameter in DWT kernel functions..*

- If the `dual` parameter is false, the DWT is computed as the forward filter bank transform with filters  $H_0, H_1$ , and the IDWT is computed as the reverse filter bank transform with filters  $G_0, G_1$ .
- If the `dual` parameter is true, the DWT is computed as the forward filter bank transform with filters  $(G_0)^T, (G_1)^T$ , and the IDWT is computed as the reverse filter bank transform with filters  $(H_0)^T, (H_1)^T$ .

This means that we can differ between the DWT, IDWT, and their duals as follows.

```
DWTImpl(x, m, wave_name, True, False) # DWT
IDWTImpl(x, m, wave_name, True, False) # IDWT
DWTImpl(x, m, wave_name, True, True) # Dual DWT
IDWTImpl(x, m, wave_name, True, True) # Dual IDWT
```

Note that, even though the reverse filter bank transform  $G$  can be associated with certain function bases, it is not clear if the reverse filter bank transform  $H^T$  also can be associated with such bases. We will see in the next chapter that such bases can in many cases be found. We will also denote these bases as *dual bases*.

The construction of the dual wavelet transform was function-free - we have no reason to believe that they correspond to scaling functions and mother wavelets. In the next chapter we will show that such dual scaling functions and dual mother wavelets exist in many cases. We can set the `dual` parameter to `True` in

the implementation of the cascade algorithm in Example 5.43 to see how the functions must look. In Figure 6.4 we have plotted the result. We see that these functions look very irregular. Also, they are very different from the original scaling function and mother wavelet. We will later argue that this is bad, it would be much better if  $\phi \approx \tilde{\phi}$  and  $\psi \approx \tilde{\psi}$ .

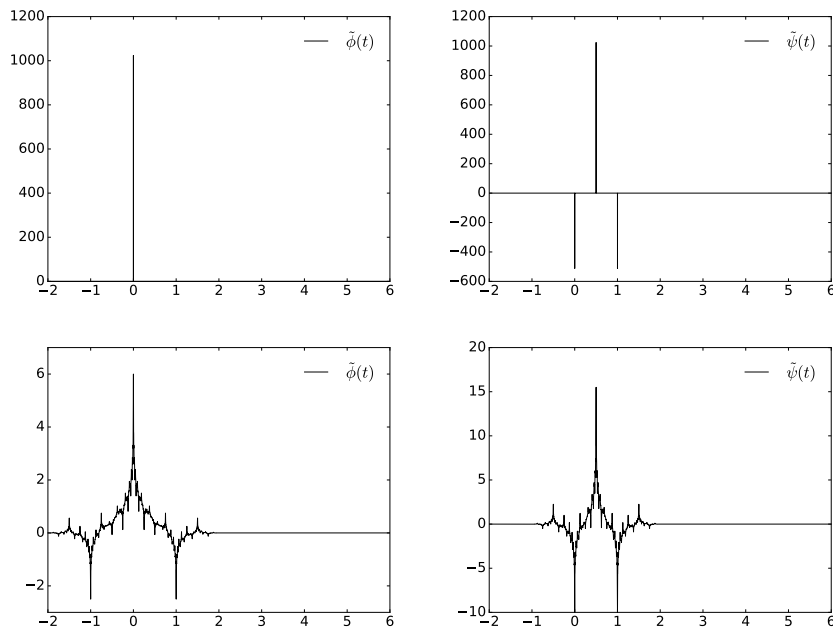


Figure 6.4: Dual functions for the two piecewise linear wavelets.

In the construction of the alternative piecewise linear wavelet we actually made a DWT/IDWT implementation before we found the filter coefficients themselves. But since the filter coefficients of  $G_0$  and  $G_1$  can be found in the columns of the matrix, they can be extracted by applying the IDWT kernel implementation to  $\mathbf{e}_0$  (for the low-pass filter coefficients) and  $\mathbf{e}_1$  (for the high-pass filter coefficients). Finally the frequency responses can be plotted by copying what we did in Section 3.3 The following algorithm does this.

```
def freqresp_alg(wave_name, lowpass, dual):
    idwt_kernel = find_kernel(wave_name, 0, dual, False);
    N = 128
    n = arange(0,N)
    omega = 2*pi*n/float(N)

    g = zeros(N)
    if lowpass:
        g[0] = 1
    else:
        g[1] = 1

    idwt_kernel(g, 'per')
```

```
plt.figure()
plt.plot(omega, abs(fft.fft(g)), 'k-')
```

If the parameter `dual` is set to `True`, the dual filters  $(H_0)^T$  and  $(H_1)^T$  are plotted instead. If the filters have real coefficients,  $|\lambda_{H_i^T}(\omega)| = |\lambda_{H_i}(\omega)|$ , so the correct frequency responses are shown.

### Example 6.4: Plotting the frequency responses

In order to verify the low-pass/high-pass characteristics of  $G_0$  and  $G_1$ , let us plot the frequency responses of the wavelets we have considered. To plot  $\lambda_{G_0}(\omega)$  and  $\lambda_{G_1}(\omega)$  for the Haar wavelet, we can write

```
freqresp_alg('Haar', True, False)
freqresp_alg('Haar', False, False)
```

To plot the same frequency response for the alternative piecewise linear wavelet, we can write

```
freqresp_alg('pw12', True, False)
freqresp_alg('pw12', False, False)
```

The resulting frequency responses are shown in Figure 6.5. Low-pass/high-pass characteristics are clearly seen here.

## 6.1.2 The support of the scaling function and the mother wavelet

The scaling functions and mother wavelets we encounter will turn out to always be functions with compact support. An interesting consequence of equations (6.2) and (6.3) is that we can find the size of these supports from the number of filter coefficients in  $G_0$  and  $G_1$ . In the following we will say that the support of a filter is  $[E, F]$  if  $t_{-E}, \dots, t_F$  are the only nonzero filter coefficients.

**Theorem 6.9.** *Support size.*

Assume that  $G_0$  has support  $[M_0, M_1]$ ,  $G_1$  has support  $[N_0, N_1]$ . Then the support of  $\phi$  is  $[M_0, M_1]$ , and the support of  $\psi$  is  $[(M_0 + N_0 + 1)/2, (M_1 + N_1 + 1)/2]$ .

*Proof.* Let  $[m_0, m_1]$  be the support of  $\phi$ . Then  $\phi_{1,n}$  clearly has support  $[(m_0 + n)/2, (m_1 + n)/2]$ . In the equation

$$\phi(t) = \sum_{n=M_0}^{M_1} (G_0)_n \phi_{1,n},$$

the function with the leftmost support is  $\phi_{1,M_0}$ , while the one with the rightmost one is  $\phi_{1,M_1}$ . These supports are  $[(m_0 + M_0)/2, (m_1 + M_1)/2]$ . In order for the supports of the two sides to match we clearly must have



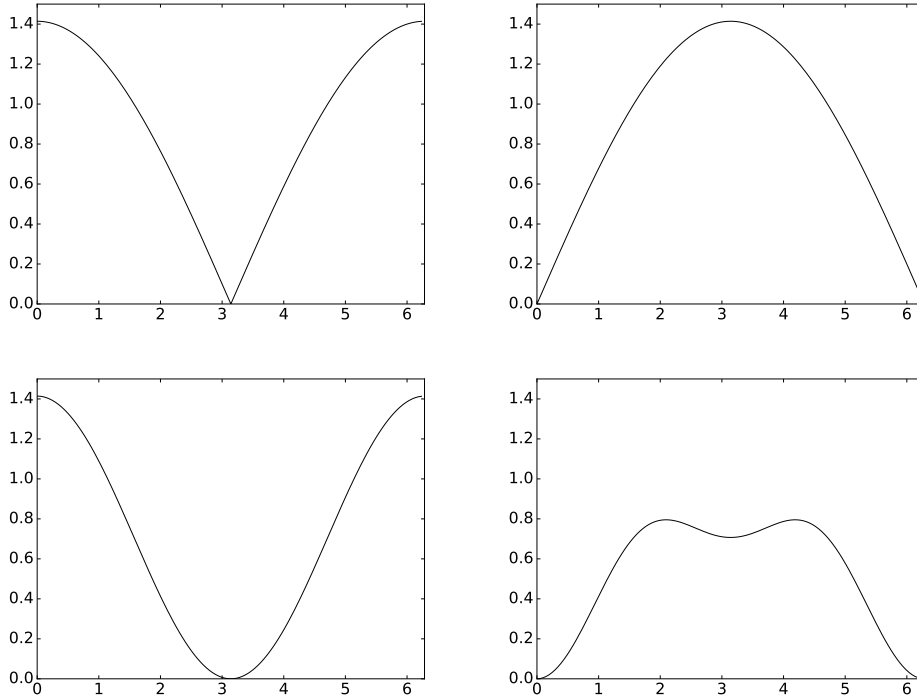


Figure 6.5: The frequency responses  $\lambda_{G_0}(\omega)$  and  $\lambda_{G_1}(\omega)$  for the Haar wavelet (top), and for the alternative piecewise linear wavelet (bottom).

$$m_0 = (m_0 + M_0)/2 \qquad m_1 = (m_1 + M_1)/2.$$

It follows that  $m_0 = M_0$  and  $m_1 = M_1$ , so that the support of  $\phi$  is  $[M_0, M_1]$ . Similarly, let  $[n_0, n_1]$  be the support of  $\psi$ . We have that

$$\psi(t) = \sum_{n=N_0+1}^{N_1+1} (G_1)_{n-1} \phi_{1,n},$$

and we get in the same way

$$n_0 = (m_0 + N_0 + 1)/2 \qquad n_1 = (m_1 + N_1 + 1)/2.$$

It follows that  $n_0 = (M_0 + N_0 + 1)/2$ .  $n_1 = (M_1 + N_1 + 1)/2$ , so that the support of  $\psi$  is  $((M_0 + N_0 + 1)/2, (M_1 + N_1 + 1)/2)$ .  $\square$

There are two special cases of the above we will run into.

**Wavelets with symmetric filters.** The results then say the support of  $\phi$  is  $[-M_1, M_1]$  (i.e. symmetric around 0), and the support of  $\psi$  is  $1/2 + [-(M_1 + N_1)/2, (M_1 + N_1)/2]$ , i.e. symmetric around  $1/2$ . The wavelet with most such filter coefficients we will consider has 7 and 9 filter coefficients, respectively, so that the support of  $\phi$  is  $[-3, 3]$ , and the support of  $\psi$  is  $[-3, 4]$ . This is why we have plotted these functions over  $[-4, 4]$ , so that the entire function can be seen. For the alternative piecewise linear wavelet the same argument gives that support of  $\phi$  is  $[-1, 1]$ , and the support of  $\psi$  is  $[-1, 2]$  (which we already knew from Figure 5.18). For the piecewise linear wavelet the support of  $\psi$  is deduced to be  $[0, 1]$ .

**Orthonormal wavelets.** For these wavelets it will turn that  $G_0$  has filter coefficients evenly distributed around  $1/2$ , and  $G_1$  has equally many, and evenly distributed around  $-1/2$ . It is straightforward to check that the filters for the Haar wavelet are of this kind, and this will turn out to be the simplest case of an orthonormal wavelet. For such supports Theorem 6.9 says that both supports are symmetric around  $1/2$ , and that both  $\phi$ ,  $\psi$ ,  $G_0$  and  $G_1$  have the same support lengths. This can also be verified from the plots for the Haar wavelet. We will only consider orthonormal wavelets with at most 8 filter coefficients. This number of filter coefficients is easily seen to give the support  $[-3, 4]$ , which is why we have used  $[-4, 4]$  as a common range when we plot functions on this form.

### 6.1.3 Symmetric extensions and the `bd_mode` parameter.

Continuous functions  $f : [0, N] \rightarrow \mathbb{R}$  are approximated well from  $V_m$ , at least if the wavelet has one vanishing moment. The periodic extension of  $f$  is not continuous, however, if  $f(0) \neq f(N)$ . We can instead form the symmetric extension  $\check{f}$  of  $f$ , as given by Definition 1.22, which is defined on  $[0, 2N]$ , and which can be periodically extended to a continuous function. We make a smaller error if we restrict to the approximation of  $\check{f}$  from  $V_m$ , when compared to that of  $f$  from  $V_m$ .

The input to the DWT is given in terms of a vector  $\mathbf{c}$ , however, so that our  $f$  is really the function  $\sum_n c_{m,n} \phi_{m,n}$ . If  $\phi$  has compact support, then clearly the function  $\sum_n \check{c}_{m,n} \phi_{m,n}$  retains the same values as the symmetric extension (except near the boundary), when  $\check{\mathbf{c}}$  is some kind of symmetric extension of the vector  $\mathbf{c}$ . We are free to decide how vectors are symmetrically extended near the boundary. In the theory of wavelets, the following symmetric extension strategy for vectors is used.

**Definition 6.10.** *Symmetric extension of a vector.*

By the *symmetric extension* of  $\mathbf{x} \in \mathbb{R}^N$ , we mean  $\check{\mathbf{x}} \in \mathbb{R}^{2N-2}$  defined by

$$\check{\mathbf{x}}_k = \begin{cases} x_k & 0 \leq k < N \\ x_{2N-2-k} & N \leq k < 2N-2 \end{cases} \quad (6.8)$$

This is different from the symmetric extension given by Definition 4.1. Note that  $(\check{f}(0), \check{f}(1), \dots, \check{f}(N-1), \check{f}(N), \check{f}(N+1), \dots, \check{f}(2N-1)) \in \mathbb{R}^{2N}$  is now the symmetric extension of  $(f(0), f(1), \dots, f(N))$ , so that this way of defining symmetric extensions is perhaps the most natural when it comes to sampling which includes the boundaries.

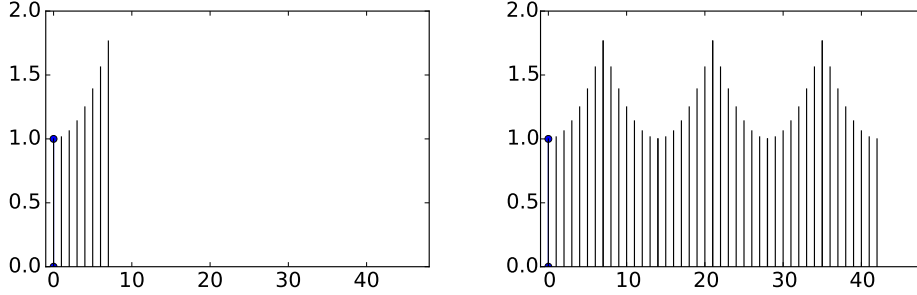


Figure 6.6: A vector and its symmetric extension. Note that the period of the vector is now  $2N - 2$ , while it was  $2N$  for the vector shown in Figure 4.1.

Consider applying the DWT to a symmetric extension  $\check{\mathbf{x}}$  of length  $2N - 2$ . Assume that all filters are symmetric  $(2N - 2) \times (2N - 2)$  filters. According to Chapter 4, they preserve vectors symmetric around 0 and  $N - 1$ , so that there exist  $N \times N$ -matrices  $(H_0)_r, (H_1)_r, (G_0)_r,$  and  $(G_1)_r$ , so that for all  $\mathbf{x} \in \mathbb{R}^N$ ,  $H_i \check{\mathbf{x}} = (H_i)_r \check{\mathbf{x}}$  and  $G_i \check{\mathbf{x}} = (G_i)_r \check{\mathbf{x}}$ . In particular, the first  $N$  entries in  $H_i \check{\mathbf{x}}$  are  $(H_i)_r \mathbf{x}$ . The first  $N$  entries of  $H \check{\mathbf{x}}$  are thus obtained by assembling the even-indexed entries from  $(H_0)_r \mathbf{x}$ , and the odd-indexed entries from  $(H_1)_r \mathbf{x}$ .

Since the difference between  $N - 1 - k$  and  $N - 1 + k$  is even, and since  $H_i \check{\mathbf{x}}$  is a symmetric extension, for one  $i$  we have that

$$(H \check{\mathbf{x}})_{N-1-k} = (H_i \check{\mathbf{x}})_{N-1-k} = (H_i \check{\mathbf{x}})_{N-1+k} = (H \check{\mathbf{x}})_{N-1+k}.$$

It follows that  $H$  preserves the same type of symmetric extensions, i.e. there exists an  $N \times N$ -matrix  $H_r$  so that  $H \check{\mathbf{x}} = H_r \mathbf{x}$ . Moreover, the entries in  $H_r \mathbf{x}$  are assembled from the entries in  $(H_i)_r \mathbf{x}$ , in the same way as the entries in  $H \check{\mathbf{x}}$  are assembled from the entries in  $H_i \check{\mathbf{x}}$ .

Note also that setting every second element to zero in a symmetric extension only creates a new symmetric extension, so that  $G$  also preserves symmetric extensions. It follows that there exist  $N \times N$ -matrices  $G_r, (G_0)_r, (G_1)_r$  so that  $G \check{\mathbf{x}} = G_r \mathbf{x}$ , and so that the entries  $0, \dots, N - 1$  in the output of  $G_r$  are obtained by combining  $(G_0)_r$  and  $(G_1)_r$  as in Theorem 6.5.

**Theorem 6.11.** *Symmetric filters and symmetric extensions.*

If the filters  $H_0, H_1, G_0,$  and  $G_1$  in a wavelet transform are symmetric, then the DWT/IDWT preserve symmetric extensions (as defined in Definition 6.10). Also, applying the filters  $H_0, H_1, G_0,$  and  $G_1$  to  $\check{\mathbf{x}} \in \mathbb{R}^{2N-2}$  in the DWT/IDWT is equivalent to applying  $(H_0)_r, (H_1)_r, (G_0)_r,$  and  $(G_1)_r$  to  $\mathbf{x} \in \mathbb{R}^N$  as described in theorems 6.3 and 6.5.

In our implementations, we factored  $H$  and  $G$  in terms of elementary lifting matrices. Note that the filters in these matrices are symmetric so that, after applying  $A\check{\mathbf{x}} = A_r\mathbf{x}$  repeatedly,

$$\prod_i A_{\lambda_{2^i}} B_{\lambda_{2^{i+1}}} \check{\mathbf{x}} = \prod_i (A_{\lambda_{2^i}})_r (B_{\lambda_{2^{i+1}}})_r \mathbf{x}$$

It follows that  $(\prod_i A_{\lambda_{2^i}} B_{\lambda_{2^{i+1}}})_r = \prod_i (A_{\lambda_{2^i}})_r (B_{\lambda_{2^{i+1}}})_r$ . It is straightforward to find expressions for  $(A_\lambda)_r$  and  $(B_\lambda)_r$  (Exercise 6.7). DWT and IDWT implementations can thus apply symmetric extensions by replacing  $A_\lambda$  and  $B_\lambda$  with  $(A_\lambda)_r$  and  $(B_\lambda)_r$ .

The purpose of the `bd_mode` parameter is to control how we should handle the boundary of the signal, in particular whether symmetric extensions should be performed. This parameter is passed to `lifting_even_symm` and `lifting_odd_symm`, which are the building blocks in our kernel functions. If this parameter equals 'symm' (this is the default value), the methods apply symmetric extensions by using  $(A_\lambda)_r/(B_\lambda)_r$  rather than  $A_\lambda/B_\lambda$ . If the parameter equals 'per', a periodic extension at the boundaries is made.

**Fact 6.12.** *The `bd_mode`-parameter in DWT kernel functions.*

Assume that the filters  $H_0, H_1, G_0,$  and  $G_1$  are symmetric. If the `bd_mode` parameter is "symm", the symmetric versions  $(H_0)_r, (H_1)_r, (G_0)_r,$  and  $(G_1)_r$  are applied in the DWT and IDWT, rather than the filters  $H_0, H_1, G_0,$  and  $G_1$  themselves. If the 'parameter is "per", the filters  $H_0, H_1, G_0,$  and  $G_1$  are applied.

**Exercise 6.5: Implement the dual filter bank transforms**

- a) Show that  $A_\lambda^T = B_\lambda$  and  $B_\lambda^T = A_\lambda$ , i.e. that the transpose of an elementary lifting matrix of even/odd type is an elementary lifting matrix of odd/even type.
- b) Let  $H$  be the kernel of the DWT, and assume that we have a factorization of it in terms of elementary lifting matrices. Use a) how that the dual DWT is obtained from the DWT by replacing each  $A_\lambda$  with  $B_{-\lambda}$ , and  $B_\lambda$  with  $A_{-\lambda}$  in this factorization.
- c) Previously we expressed the DWT and the IDWT of the piecewise linear wavelets in terms of elementary liftings. Use b) to write down the dual DWT and IDWT of these two wavelets in terms of lifting matrices. Verify your answer by going through the code in the functions `dwt_kernel_pw10`, `idwt_kernel_pw10`, `dwt_kernel_pw12`, and `idwt_kernel_pw12` where the dual parameter is set to true..

**Exercise 6.6: Transpose of the DWT and IDWT**

Explain why

- The transpose of the DWT can be computed with an IDWT with the kernel of the dual IDWT

- The transpose of the dual DWT can be computed with an IDWT with the kernel of the IDWT
- The transpose of the IDWT can be computed with a DWT with the kernel of the dual DWT
- The transpose of the dual IDWT can be computed with a DWT with the kernel of the DWT

**Exercise 6.7: Reduced matrices for elementary lifting**

Show that the reduced matrices for elementary lifting are

$$(A_\lambda)_r = \begin{pmatrix} 1 & 2\lambda & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda & 1 & \lambda & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda & 1 & \lambda \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \tag{6.9}$$

$$(B_\lambda)_r = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda & 1 & \lambda & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 2\lambda & 1 \end{pmatrix}. \tag{6.10}$$

Also, change the implementations of `liftingstevensymm` and `liftingstoddsymm` so that these expressions are used (rather than  $A_\lambda, B_\lambda$ ) when the parameter `symm` is set to `True`.

**Exercise 6.8: Prove expression for  $S_r$**

Show that, with

$$S = \begin{pmatrix} S_1 & S_2 \\ S_3 & S_4 \end{pmatrix} \in \mathbb{R}^{2N-2} \times \mathbb{R}^{2N-2}$$

a symmetric filter, with  $S_1 \in \mathbb{R}^N \times \mathbb{R}^N, S_2 \in \mathbb{R}^N \times \mathbb{R}^{N-2}$ , we have that

$$S_r = S_1 + \begin{pmatrix} 0 & (S_2)^f & 0 \end{pmatrix}.$$

Use the proof of Theorem 4.9 as a guide.

**Exercise 6.9: Orthonormal basis for the symmetric extensions**

In this exercise we will establish an orthonormal basis for the symmetric extensions, as defined by Definition 6.10. This parallels Theorem 4.6.

a) Explain why, if  $\mathbf{x} \in \mathbb{R}^{2N-2}$  is a symmetric extension (according to Definition 4.1), then  $(\hat{\mathbf{x}})_n = z_n e^{-\pi i n}$ , where  $\mathbf{z}$  is a real vectors which satisfies  $z_n = z_{2N-2-n}$

b) Show that

$$\left\{ \mathbf{e}_0, \left\{ \frac{1}{\sqrt{2}}(\mathbf{e}_i + \mathbf{e}_{2N-2-i}) \right\}_{n=1}^{N-2}, \mathbf{e}_{N-1} \right\} \tag{6.11}$$

is an orthonormal basis for the vectors on the form  $\hat{\mathbf{x}}$  with  $\mathbf{x} \in \mathbb{R}^{2N-2}$  a symmetric extension.

c) Show that

$$\begin{aligned} & \frac{1}{\sqrt{2N-2}} \cos\left(2\pi \frac{0}{2N-2} k\right) \\ & \left\{ \frac{1}{\sqrt{N-1}} \cos\left(2\pi \frac{n}{2N-2} k\right) \right\}_{n=1}^{N-2} \\ & \frac{1}{\sqrt{2N-2}} \cos\left(2\pi \frac{N-1}{2N-2} k\right) \end{aligned} \tag{6.12}$$

is an orthonormal basis for the symmetric extensions in  $\mathbb{R}^{2N-2}$ .

d) Assume that  $S$  is symmetric. Show that the vectors listed in (6.12) are eigenvectors for  $S_r$ , when the vectors are viewed as vectors in  $\mathbb{R}^N$ , and that they are linearly independent. This shows that  $S_r$  is diagonalizable.

**Exercise 6.10: Diagonalizing  $S_r$**

Let us explain how the matrix  $S_r$  can be diagonalized, similarly to how we previously diagonalized using the DCT. In Exercise 6.9 we showed that the vectors

$$\left\{ \cos\left(2\pi \frac{n}{2N-2} k\right) \right\}_{n=0}^{N-1} \tag{6.13}$$

in  $\mathbb{R}^N$  is a basis of eigenvectors for  $S_r$  when  $S$  is symmetric.  $S_r$  itself is not symmetric, however, so that this basis can not possibly be orthogonal ( $S$  is symmetric if and only if it is orthogonally diagonalizable). However, when the

vectors are viewed in  $\mathbb{R}^{2N-2}$  we showed in Exercise 6.9c) an orthogonality statement which can be written as

$$\sum_{k=0}^{2N-3} \cos\left(2\pi \frac{n_1}{2N-2} k\right) \cos\left(2\pi \frac{n_2}{2N-2} k\right) = (N-1) \times \begin{cases} 2 & \text{if } n_1 = n_2 \in \{0, N-1\} \\ 1 & \text{if } n_1 = n_2 \notin \{0, N-1\} \\ 0 & \text{if } n_1 \neq n_2 \end{cases} \quad (6.14)$$

a) Show that

$$\begin{aligned} & (N-1) \times \begin{cases} 1 & \text{if } n_1 = n_2 \in \{0, N-1\} \\ \frac{1}{2} & \text{if } n_1 = n_2 \notin \{0, N-1\} \\ 0 & \text{if } n_1 \neq n_2 \end{cases} \\ &= \frac{1}{\sqrt{2}} \cos\left(2\pi \frac{n_1}{2N-2} \cdot 0\right) \frac{1}{\sqrt{2}} \cos\left(2\pi \frac{n_2}{2N-2} \cdot 0\right) \\ & \quad + \sum_{k=1}^{N-2} \cos\left(2\pi \frac{n_1}{2N-2} k\right) \cos\left(2\pi \frac{n_2}{2N-2} k\right) \\ & \quad + \frac{1}{\sqrt{2}} \cos\left(2\pi \frac{n_1}{2N-2} (N-1)\right) \frac{1}{\sqrt{2}} \cos\left(2\pi \frac{n_2}{2N-2} (N-1)\right). \end{aligned}$$

**Hint.** Use that  $\cos x = \cos(2\pi - x)$  to pair the summands  $k$  and  $2N-2-k$ .

Now, define the vector  $\mathbf{d}_n^{(1)}$  as

$$d_{n,N} \left( \frac{1}{\sqrt{2}} \cos\left(2\pi \frac{n}{2N-2} \cdot 0\right), \left\{ \cos\left(2\pi \frac{n}{2N-2} k\right) \right\}_{k=1}^{N-2}, \frac{1}{\sqrt{2}} \cos\left(2\pi \frac{n}{2N-2} (N-1)\right) \right),$$

and define  $d_{0,N}^{(1)} = d_{N-1,N}^{(1)} = 1/\sqrt{N-1}$ , and  $d_{n,N}^{(1)} = \sqrt{2/(N-1)}$  when  $n > 1$ .

The orthogonal  $N \times N$  matrix where the rows are  $\mathbf{d}_n^{(1)}$  is called the DCT-I, and we will denote it by  $D_N^{(1)}$ . DCT-I is also much used, just as the DCT-II of Chapter 4. The main difference from the previous cosine vectors is that  $2N$  has been replaced by  $2N-2$ .

b) Explain that the vectors  $\mathbf{d}_n^{(1)}$  are orthonormal, and that the matrix

$$\sqrt{\frac{2}{N-1}} \begin{pmatrix} 1/\sqrt{2} & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1/\sqrt{2} \end{pmatrix} \left( \cos\left(2\pi \frac{n}{2N-2} k\right) \right) \begin{pmatrix} 1/\sqrt{2} & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1/\sqrt{2} \end{pmatrix}$$

is orthogonal.

c) Explain from b) that  $\left(\cos\left(2\pi\frac{n}{2N-2}k\right)\right)^{-1}$  can be written as

$$\frac{2}{N-1} \begin{pmatrix} 1/2 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1/2 \end{pmatrix} \left(\cos\left(2\pi\frac{n}{2N-2}k\right)\right) \begin{pmatrix} 1/2 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1/2 \end{pmatrix}$$

With the expression we found in c)  $S_r$  can now be diagonalized as

$$\left(\cos\left(2\pi\frac{n}{2N-2}k\right)\right) D \left(\cos\left(2\pi\frac{n}{2N-2}k\right)\right)^{-1}.$$

**Exercise 6.11: Compute filters and frequency responses 1**

Write down the corresponding filters  $G_0$  og  $G_1$  for Exercise 5.40. Plot their frequency responses, and characterize the filters as low-pass- or high-pass filters.

**Exercise 6.12: Symmetry of MRA matrices vs. symmetry of filters 1**

Find two symmetric filters, so that the corresponding MRA-matrix, constructed with alternating rows from these two filters, is not a symmetric matrix.

**Exercise 6.13: Symmetry of MRA matrices vs. symmetry of filters 2**

Assume that an MRA-matrix is symmetric. Are the corresponding filters  $H_0$ ,  $H_1$ ,  $G_0$ ,  $G_1$  also symmetric? If not, find a counterexample.

**Exercise 6.14: Finding  $H_0$ ,  $H_1$  from  $H$**

Assume that one stage in a DWT is given by the MRA-matrix

$$H = \begin{pmatrix} 1/5 & 1/5 & 1/5 & 0 & 0 & 0 & \cdots & 0 & 1/5 & 1/5 \\ -1/3 & 1/3 & -1/3 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & -1/3 & 1/3 & -1/3 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Write down the compact form for the corresponding filters  $H_0$ ,  $H_1$ , and compute and plot the frequency responses. Are the filters symmetric?



**Exercise 6.15: Finding  $G_0, G_1$  from  $G$**

Assume that one stage in the IDWT is given by the MRA-matrix

$$G = \begin{pmatrix} 1/2 & -1/4 & 0 & 0 & \cdots \\ 1/4 & 3/8 & 1/4 & 1/16 & \cdots \\ 0 & -1/4 & 1/2 & -1/4 & \cdots \\ 0 & 1/16 & 1/4 & 3/8 & \cdots \\ 0 & 0 & 0 & -1/4 & \cdots \\ 0 & 0 & 0 & 1/16 & \cdots \\ 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots \\ 1/4 & 1/16 & 0 & 0 & \cdots \end{pmatrix}$$

Write down the compact form for the filters  $G_0, G_1$ , and compute and plot the frequency responses. Are the filters symmetric?

**Exercise 6.16: Finding  $H$  from  $H_0, H_1$**

Assume that  $H_0 = \{1/16, 1/4, 3/8, 1/4, 1/16\}$ , and  $H_1 = \{-1/4, 1/2, -1/4\}$ . Plot the frequency responses of  $H_0$  and  $H_1$ , and verify that  $H_0$  is a lowpass filter, and that  $H_1$  is a highpass filter. Also write down the change of coordinate matrix  $P_{c_1 \leftarrow \phi_1}$  for the wavelet corresponding to these filters.

**Exercise 6.17: Finding  $G$  from  $G_0, G_1$**

Assume that  $G_0 = \frac{1}{3}\{1, 1, 1\}$ , and  $G_1 = \frac{1}{5}\{1, -1, 1, -1, 1\}$ . Plot the frequency responses of  $G_0$  and  $G_1$ , and verify that  $G_0$  is a lowpass filter, and that  $G_1$  is a highpass filter. Also write down the change of coordinate matrix  $P_{\phi_1 \leftarrow c_1}$  for the wavelet corresponding to these filters.

**Exercise 6.18: Computing by hand**

In Exercise 5.21 we computed the DWT of two very simple vectors  $x_1$  and  $x_2$ , using the Haar wavelet.

- a) Compute  $H_0x_1, H_1x_1, H_0x_2,$  and  $H_1x_2$ , where  $H_0$  and  $H_1$  are the filters used by the Haar wavelet.
- b) Compare the odd-indexed elements in  $H_1x_1$  with the odd-indexed elements in  $H_1x_2$ . From this comparison, attempt to find an explanation to why the two vectors have very different detail components.

**Exercise 6.19: Comment code**

Suppose that we run the following algorithm on the sound represented by the vector  $x$ :

```

c = (x[0::2] + x[1::2])/sqrt(2)
w = (x[0::2] - x[1::2])/sqrt(2)

newx = concatenate([c, w])
newx /= abs(newx).max()
play(newx,44100)

```

- a) Comment the code and explain what happens. Which wavelet is used? What do the vectors  $\mathbf{c}$  and  $\mathbf{w}$  represent? Describe the sound you believe you will hear.
- b) Assume that we add lines in the code above which sets the elements in the vector  $\mathbf{w}$  to 0 before we compute the inverse operation. What will you hear if you play the new sound you then get?

### Exercise 6.20: Computing filters and frequency responses

Let us return to the piecewise linear wavelet from Exercise 5.39.

- a) With  $\hat{\psi}$  as defined as in b) in Exercise 5.39, compute the coordinates of  $\hat{\psi}$  in the basis  $\phi_1$  (i.e.  $[\hat{\psi}]_{\phi_1}$ ) with  $N = 8$ , i.e. compute the IDWT of

$$[\hat{\psi}]_{(\phi_0, \psi_0)} = (-\alpha, -\beta, -\delta, 0, 0, 0, 0, -\gamma) \oplus (1, 0, 0, 0, 0, 0, 0, 0),$$

which is the coordinate vector you computed in d) in Exercise 5.39. For this, you should use the function `IDWTImp1`, with the kernel of the piecewise linear wavelet without symmetric extension as input. Explain that this gives you the filter coefficients of  $G_1$ .

- b) Plot the frequency response of  $G_1$ .

### Exercise 6.21: Computing filters and frequency responses

#### 2

Repeat the previous exercise for the Haar wavelet as in Exercise 5.41, and plot the corresponding frequency responses for  $k = 2, 4, 6$ .

### Exercise 6.22: Implementing with symmetric extension

In Exercise 3.9 we implemented a symmetric filter applied to a vector, i.e. when a periodic extension is assumed. The corresponding function was called `filterS(t, x)`, and used the function `numpy.convolve`.

- a) Reimplement the function `filterS` so that it also takes a third parameter `symm`. If `symm` is false a periodic extension of  $\mathbf{x}$  should be performed (i.e. filtering as we have defined it, and as the previous version of `filterS` performs it). If `symm` is true, symmetric extensions should be used (as given by Definition 6.10).
- b) Implement functions

```
dwt_kernel_filters(H0, H1, G0, G1, x, bd_mode)
idwt_kernel_filters(H0, H1, G0, G1, x, bd_mode)
```

which return the DWT and IDWT kernels using theorems 6.3 and 6.5, respectively. This function thus bases itself on that the filters of the wavelet are known. The functions should call the function `filterS` from a). Recall also the definition of the parameter `dual` from this section.

With the functions defined in b) you can now define standard DWT and IDWT kernels in the following way, once the filters are known.

```
dwt_kernel = lambda x, bd_mode: dwt_kernel_filters(H0, H1, G0, G1, x, bd_mode)
idwt_kernel = lambda x, bd_mode: idwt_kernel_filters(H0, H1, G0, G1, x, bd_mode)
```

## 6.2 Properties of the filter bank transforms of a wavelet

We have now described the DWT/IDWT as linear transformations  $G, H$  so that  $GH = I$ , and where two filters  $G_0, G_1$  characterize  $G$ , two filters  $H_0, H_1$  characterize  $H$ .  $G$  and  $H$  are not Toeplitz matrices, however, so they are not filters. Since filters produce the same output frequency from an input frequency, we must have that  $G$  and  $H$  produce other (undesired) frequencies in the output than those that are present in the input. We will call this phenomenon *aliasing*. In order for  $GH = I$ , the undesired frequencies must cancel each other, so that we end up with what we started with. Thus,  $GH$  must have what we will refer to as *alias cancellation*. This is the same as saying that  $GH$  is a filter. In order for  $GH = I$ , alias cancellation is not enough: We also need that the amount at the given frequency is unchanged, i.e. that  $GH\phi_n = \phi_n$  for any Fourier basis vector  $\phi_n$ . We then say that we have *perfect reconstruction*. Perfect reconstruction is always the case for wavelets by construction, but in signal processing many interesting examples  $(G_0, G_1, H_0, H_1)$  exist, for which we do not have perfect reconstruction. Historically, forward and reverse filter bank transforms have been around long before they appeared in a wavelet context. Operations where  $GH\phi_n = c_n\phi_n$  for all  $n$  may also be useful, in particular when  $c_n$  is close to 1 for all  $n$ . If  $c_n$  is real for all  $n$ , we say that we have *no phase distortion*. If we have no phase distortion, the output from  $GH$  has the same phase, even if we do not have perfect reconstruction. Such “near-perfect reconstruction systems” have also been around long before many perfect reconstruction wavelet systems were designed. In signal processing, these transforms also exist in more general variants, and we will define these later. Let us summarize as follows.

**Definition 6.13.** *Alias cancellation, phase distortion, and perfect reconstruction.*

We say that we have *alias cancellation* if, for any  $n$ ,

$$GH\phi_n = c_n\phi_n,$$

for some constant  $c_n$  (i.e.  $GH$  is a filter). If all  $c_n$  are real, we say that we *no phase distortion*. If  $GH = I$  (i.e.  $c_n = 1$  for all  $n$ ) we say that we have *perfect reconstruction*. If all  $c_n$  are close to 1, we say that we have *near-perfect reconstruction*.

In signal processing, one also says that we have perfect- or near-perfect reconstruction when  $GH$  equals  $E_d$ , or is close to  $E_d$  (i.e. the overall result is a delay). The reason why a delay occurs has to do with that the transforms are used in real-time processing, for which we may not be able to compute the output at a given time instance before we know some of the following samples. Clearly the delay is unproblematic, since one can still reconstruct the input from the output. We will encounter a useful example of near-perfect reconstruction soon in the MP3 standard.

Let us now find a criterium for alias cancellation: When do we have that  $GH e^{2\pi i r k/N}$  is a multiplum of  $e^{2\pi i r k/N}$ , for any  $r$ ? We first remark that

$$H(e^{2\pi i r k/N}) = \begin{cases} \lambda_{H_0,r} e^{2\pi i r k/N} & k \text{ even} \\ \lambda_{H_1,r} e^{2\pi i r k/N} & k \text{ odd.} \end{cases}$$

The frequency response of  $H(e^{2\pi i r k/N})$  is

$$\begin{aligned} & \sum_{k=0}^{N/2-1} \lambda_{H_0,r} e^{2\pi i r(2k)/N} e^{-2\pi i(2k)n/N} + \sum_{k=0}^{N/2-1} \lambda_{H_1,r} e^{2\pi i r(2k+1)/N} e^{-2\pi i(2k+1)n/N} \\ &= \sum_{k=0}^{N/2-1} \lambda_{H_0,r} e^{2\pi i(r-n)(2k)/N} + \sum_{k=0}^{N/2-1} \lambda_{H_1,r} e^{2\pi i(r-n)(2k+1)/N} \\ &= (\lambda_{H_0,r} + \lambda_{H_1,r} e^{2\pi i(r-n)/N}) \sum_{k=0}^{N/2-1} e^{2\pi i(r-n)k/(N/2)}. \end{aligned}$$

Clearly,  $\sum_{k=0}^{N/2-1} e^{2\pi i(r-n)k/(N/2)} = N/2$  if  $n = r$  or  $n = r + N/2$ , and 0 else. The frequency response is thus the vector

$$\frac{N}{2}(\lambda_{H_0,r} + \lambda_{H_1,r})\mathbf{e}_r + \frac{N}{2}(\lambda_{H_0,r} - \lambda_{H_1,r})\mathbf{e}_{r+N/2},$$

so that

$$H(e^{2\pi i r k/N}) = \frac{1}{2}(\lambda_{H_0,r} + \lambda_{H_1,r})e^{2\pi i r k/N} + \frac{1}{2}(\lambda_{H_0,r} - \lambda_{H_1,r})e^{2\pi i(r+N/2)k/N}. \quad (6.15)$$

Let us now turn to the reverse filter bank transform. We can write

$$\begin{aligned} (e^{2\pi i r \cdot 0/N}, 0, e^{2\pi i r \cdot 2/N}, 0, \dots, e^{2\pi i r(N-2)/N}, 0) &= \frac{1}{2}(e^{2\pi i r k/N} + e^{2\pi i(r+N/2)k/N}) \\ (0, e^{2\pi i r \cdot 1/N}, 0, e^{2\pi i r \cdot 3/N}, \dots, 0, e^{2\pi i r(N-1)/N}) &= \frac{1}{2}(e^{2\pi i r k/N} - e^{2\pi i(r+N/2)k/N}). \end{aligned}$$

This means that

$$\begin{aligned}
 G(e^{2\pi irk/N}) &= G_0 \left( \frac{1}{2} \left( e^{2\pi irk/N} + e^{2\pi i(r+N/2)k/N} \right) \right) + G_1 \left( \frac{1}{2} \left( e^{2\pi irk/N} - e^{2\pi i(r+N/2)k/N} \right) \right) \\
 &= \frac{1}{2} (\lambda_{G_0,r} e^{2\pi irk/N} + \lambda_{G_0,r+N/2} e^{2\pi i(r+N/2)k/N}) + \frac{1}{2} (\lambda_{G_1,r} e^{2\pi irk/N} - \lambda_{G_1,r+N/2} e^{2\pi i(r+N/2)k/N}) \\
 &= \frac{1}{2} (\lambda_{G_0,r} + \lambda_{G_1,r}) e^{2\pi irk/N} + \frac{1}{2} (\lambda_{G_0,r+N/2} - \lambda_{G_1,r+N/2}) e^{2\pi i(r+N/2)k/N}. \quad (6.16)
 \end{aligned}$$

Now, if we combine equations (6.15) and (6.16), we get

$$\begin{aligned}
 GH(e^{2\pi irk/N}) &= \frac{1}{2} (\lambda_{H_0,r} + \lambda_{H_1,r}) G(e^{2\pi irk/N}) + \frac{1}{2} (\lambda_{H_0,r} - \lambda_{H_1,r}) G(e^{2\pi i(r+N/2)k/N}) \\
 &= \frac{1}{2} (\lambda_{H_0,r} + \lambda_{H_1,r}) \left( \frac{1}{2} (\lambda_{G_0,r} + \lambda_{G_1,r}) e^{2\pi irk/N} + \frac{1}{2} (\lambda_{G_0,r+N/2} - \lambda_{G_1,r+N/2}) e^{2\pi i(r+N/2)k/N} \right) \\
 &\quad + \frac{1}{2} (\lambda_{H_0,r} - \lambda_{H_1,r}) \left( \frac{1}{2} (\lambda_{G_0,r+N/2} + \lambda_{G_1,r+N/2}) e^{2\pi i(r+N/2)k/N} + \frac{1}{2} (\lambda_{G_0,r} - \lambda_{G_1,r}) e^{2\pi irk/N} \right) \\
 &= \frac{1}{4} ((\lambda_{H_0,r} + \lambda_{H_1,r})(\lambda_{G_0,r} + \lambda_{G_1,r}) + (\lambda_{H_0,r} - \lambda_{H_1,r})(\lambda_{G_0,r} - \lambda_{G_1,r})) e^{2\pi irk/N} \\
 &\quad + \frac{1}{4} ((\lambda_{H_0,r} + \lambda_{H_1,r})(\lambda_{G_0,r+N/2} - \lambda_{G_1,r+N/2}) + (\lambda_{H_0,r} - \lambda_{H_1,r})(\lambda_{G_0,r+N/2} + \lambda_{G_1,r+N/2})) e^{2\pi i(r+N/2)k/N} \\
 &= \frac{1}{2} (\lambda_{H_0,r} \lambda_{G_0,r} + \lambda_{H_1,r} \lambda_{G_1,r}) e^{2\pi irk/N} + \frac{1}{2} (\lambda_{H_0,r} \lambda_{G_0,r+N/2} - \lambda_{H_1,r} \lambda_{G_1,r+N/2}) e^{2\pi i(r+N/2)k/N}.
 \end{aligned}$$

If we also replace with the continuous frequency response, we obtain the following:

**Theorem 6.14.** *Expression for aliasing.*

We have that

$$\begin{aligned}
 GH(e^{2\pi irk/N}) &= \frac{1}{2} (\lambda_{H_0,r} \lambda_{G_0,r} + \lambda_{H_1,r} \lambda_{G_1,r}) e^{2\pi irk/N} \\
 &\quad + \frac{1}{2} (\lambda_{H_0,r} \lambda_{G_0,r+N/2} - \lambda_{H_1,r} \lambda_{G_1,r+N/2}) e^{2\pi i(r+N/2)k/N}. \quad (6.17)
 \end{aligned}$$

In particular, we have alias cancellation if and only if

$$\lambda_{H_0}(\omega) \lambda_{G_0}(\omega + \pi) = \lambda_{H_1}(\omega) \lambda_{G_1}(\omega + \pi). \quad (6.18)$$

We will refer to this as the *alias cancellation condition*. If in addition

$$\lambda_{H_0}(\omega) \lambda_{G_0}(\omega) + \lambda_{H_1}(\omega) \lambda_{G_1}(\omega) = 2, \quad (6.19)$$

we also have perfect reconstruction. We will refer to as the *condition for perfect reconstruction*.

No phase distortion means that we have alias cancellation, and that

$$\lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_1}(\omega)\lambda_{G_1}(\omega) \text{ is real.}$$

Now let us turn to how we can construct wavelets/perfect reconstruction systems from FIR-filters (recall from Chapter 3 that FIR filters were filters with a finite number of filter coefficients). We will have use for some theorems which allow us to construct wavelets from prototype filters. In particular we show that, when  $G_0$  and  $H_0$  are given lowpass filters which satisfy a certain common property, we can define unique (up to a constant) highpass filters  $H_1$  and  $G_1$  so that the collection of these four filters can be used to implement a wavelet. We first state the following general theorem.

**Theorem 6.15.** *Criteria for perfect reconstruction.*

The following statements are equivalent for FIR filters  $H_0, H_1, G_0, G_1$ :

- $H_0, H_1, G_0, G_1$  give perfect reconstruction,
- there exist  $\alpha \in \mathbb{R}$  and  $d \in \mathbb{Z}$  so that

$$(H_1)_n = (-1)^n \alpha^{-1} (G_0)_{n-2d} \quad (6.20)$$

$$(G_1)_n = (-1)^n \alpha (H_0)_{n+2d} \quad (6.21)$$

$$2 = \lambda_{H_0, n} \lambda_{G_0, n} + \lambda_{H_0, n+N/2} \lambda_{G_0, n+N/2} \quad (6.22)$$

Let us translate this to continuous frequency responses. We first have that

$$\begin{aligned} \lambda_{H_1}(\omega) &= \sum_k (H_1)_k e^{-ik\omega} = \sum_k (-1)^k \alpha^{-1} (G_0)_{k-2d} e^{-ik\omega} \\ &= \alpha^{-1} \sum_k (-1)^k (G_0)_k e^{-i(k+2d)\omega} = \alpha^{-1} e^{-2id\omega} \sum_k (G_0)_k e^{-ik(\omega+\pi)} \\ &= \alpha^{-1} e^{-2id\omega} \lambda_{G_0}(\omega + \pi). \end{aligned}$$

We have a similar computation for  $\lambda_{G_1}(\omega)$ . We can thus state the following:

**Theorem 6.16.** *Criteria for perfect reconstruction.*

The following statements are equivalent for FIR filters  $H_0, H_1, G_0, G_1$ :

- $H_0, H_1, G_0, G_1$  give perfect reconstruction,
- there exist  $\alpha \in \mathbb{R}$  and  $d \in \mathbb{Z}$  so that

$$\lambda_{H_1}(\omega) = \alpha^{-1} e^{-2id\omega} \lambda_{G_0}(\omega + \pi) \quad (6.23)$$

$$\lambda_{G_1}(\omega) = \alpha e^{2id\omega} \lambda_{H_0}(\omega + \pi) \quad (6.24)$$

$$2 = \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_0}(\omega + \pi)\lambda_{G_0}(\omega + \pi) \quad (6.25)$$

*Proof.* Let us prove first that equations (6.23)- (6.25) for a FIR filter implies that we have perfect reconstruction. Equations (6.23)-(6.24) mean that the alias cancellation condition (6.18) is satisfied, since

$$\begin{aligned}\lambda_{H_1}(\omega)\lambda_{G_1}(\omega + \pi) &= \alpha^{-1}e^{-2id\omega}\lambda_{G_0}(\omega + \pi)(\alpha)e^{2id(\omega+\pi)}\lambda_{H_0}(\omega) \\ &= \lambda_{H_0}(\omega)\lambda_{G_0}(\omega + \pi).\end{aligned}$$

Inserting this in the perfect reconstruction condition (6.25), we get

$$\begin{aligned}2 &= \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{G_0}(\omega + \pi)\lambda_{H_0}(\omega + \pi) \\ &= \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \alpha^{-1}e^{-2id\omega}\lambda_{G_0}(\omega + \pi)\alpha e^{2id\omega}\lambda_{H_0}(\omega + \pi) \\ &= \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_1}(\omega)\lambda_{G_1}(\omega),\end{aligned}$$

which is Equation (6.19), so that equations (6.23)- (6.25) imply perfect reconstruction. We therefore only need to prove that any set of FIR filters which give perfect reconstruction, also satisfy these equations. Due to the calculation above, it is enough to prove that equations (6.23)-(6.24) are satisfied. The proof of this will wait till Section 8.1, since it uses some techniques we have not introduced yet.  $\square$

When constructing a wavelet it may be that we know one of the two pairs  $(G_0, G_1)$ ,  $(H_0, H_1)$ , and that we would like to construct the other two. This can be achieved if we can find the constants  $d$  and  $\alpha$  from above. If the filters are symmetric we just saw that  $d = 0$ . If  $G_0, G_1$  are known, it follows from from equations (6.20) and (6.21) that

$$1 = \sum_n (G_1)_n (H_1)_n = \sum_n (G_1)_n \alpha^{-1} (-1)^n (G_0)_n = \alpha^{-1} \sum_n (-1)^n (G_0)_n (G_1)_n,$$

so that  $\alpha = \sum_n (-1)^n (G_0)_n (G_1)_n$ . On the other hand, if  $H_0, H_1$  are known instead, we must have that

$$1 = \sum_n (G_1)_n (H_1)_n = \sum_n \alpha (-1)^n (H_0)_n (H_1)_n = \alpha \sum_n (-1)^n (H_0)_n (H_1)_n,$$

so that  $\alpha = 1/(\sum_n (-1)^n (H_0)_n (H_1)_n)$ . Let us use these observations to state the filters for the alternative wavelet of piecewise linear functions, which is the only wavelet we have gone through we have not computed the filters and the frequency response for.

Let us use Theorem 6.16 to compute the filters  $H_0$  and  $H_1$  for the alternative piecewise linear wavelet. These filters are also symmetric, since  $G_0, G_1$  were. We get that

$$\alpha = \sum_n (-1)^n (G_0)_n (G_1)_n = \frac{1}{2} \left( -\frac{1}{2} \left( -\frac{1}{4} \right) + 1 \cdot \frac{3}{4} - \frac{1}{2} \left( -\frac{1}{4} \right) \right) = \frac{1}{2}.$$

We now get

$$\begin{aligned} (H_0)_n &= \alpha^{-1} (-1)^n (G_1)_n = 2(-1)^n (G_1)_n \\ (H_1)_n &= \alpha^{-1} (-1)^n (G_0)_n = 2(-1)^n (G_0)_n, \end{aligned} \quad (6.26)$$

so that

$$\begin{aligned} H_0 &= \sqrt{2} \{-1/8, 1/4, \underline{3/4}, 1/4, -1/8\} \\ H_1 &= \sqrt{2} \{-1/2, \underline{1}, -1/2\}. \end{aligned} \quad (6.27)$$

Note that, even though conditions (6.23) and (6.24) together ensure that the alias cancellation condition is satisfied, alias cancellation can occur also if these conditions are not satisfied. Conditions (6.23) and (6.24) thus give a stronger requirement than alias cancellation. We will be particularly concerned with wavelets where the filters are symmetric, for which we can state the following corollary.

**Corollary 6.17.** *Criteria for perfect reconstruction .*

The following statements are equivalent:

- $H_0, H_1, G_0, G_1$  are the filters of a symmetric wavelet,
- $\lambda_{H_0}(\omega), \lambda_{H_1}(\omega), \lambda_{G_0}(\omega), \lambda_{G_1}(\omega)$  are real functions, and

$$\lambda_{H_1}(\omega) = \alpha^{-1} \lambda_{G_0}(\omega + \pi) \quad (6.28)$$

$$\lambda_{G_1}(\omega) = \alpha \lambda_{H_0}(\omega + \pi) \quad (6.29)$$

$$2 = \lambda_{H_0}(\omega) \lambda_{G_0}(\omega) + \lambda_{H_0}(\omega + \pi) \lambda_{G_0}(\omega + \pi). \quad (6.30)$$

The delay  $d$  is thus 0 for symmetric wavelets.

*Proof.* Since  $H_0$  is symmetric,  $(H_0)_n = (H_0)_{-n}$ , and from equations (6.20) and (6.21) it follows that

$$\begin{aligned} (G_1)_{n-2d} &= (-1)^{n-2d} \alpha (H_0)_n = (-1)^n \alpha^{-1} (H_0)_{-n} \\ &= (-1)^{(-n-2d)} \alpha^{-1} (H_0)_{(-n-2d)+2d} = (G_1)_{-n-2d} \end{aligned}$$

This shows that  $G_1$  is symmetric about both  $-2d$ , in addition to being symmetric about 0 (by assumption). We must thus have that  $d = 0$ , so that  $(H_1)_n = (-1)^n \alpha (G_0)_n$  and  $(G_1)_n = (-1)^n \alpha^{-1} (H_0)_n$ . We now get that



$$\begin{aligned}
 \lambda_{H_1}(\omega) &= \sum_k (H_1)_k e^{-ik\omega} = \alpha^{-1} \sum_k (-1)^k (G_0)_k e^{-ik\omega} \\
 &= \alpha^{-1} \sum_k e^{-ik\pi} (G_0)_k e^{-ik\omega} = \alpha^{-1} \sum_k (G_0)_k e^{-ik(\omega+\pi)} \\
 &= \alpha^{-1} \lambda_{G_0}(\omega + \pi),
 \end{aligned}$$

which proves Equation (6.28). Equation (6.28) follows similarly.  $\square$

In the literature, two particular cases of filter banks have been important. They are both referred to as *Quadrature Mirror Filter banks*, or QMF filter banks, and some confusion exist between the two. Let us therefore make precise definitions of the two.

**Definition 6.18.** *Classical QMF filter banks.*

In the classical definition of a QMF filter banks it is required that  $G_0 = H_0$  and  $G_1 = H_1$  (i.e. the filters in the forward and reverse transforms are equal), and that

$$\lambda_{H_1}(\omega) = \lambda_{H_0}(\omega + \pi). \quad (6.31)$$

It is straightforward to check that, for a classical QMF filter bank, the forward and reverse transforms are equal (i.e.  $G = H$ ). It is easily checked that conditions (6.23) and (6.24) are satisfied with  $\alpha = 1, d = 0$  for a classical QMF filter bank. In particular, the alias cancellation condition is satisfied. The perfect reconstruction condition can be written as

$$2 = \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_1}(\omega)\lambda_{G_1}(\omega) = \lambda_{H_0}(\omega)^2 + \lambda_{H_0}(\omega + \pi)^2. \quad (6.32)$$

Unfortunately, it is impossible to find non-trivial FIR-filters which satisfy this quadrature formula (Exercise 6.23). Therefore, classical QMF filter banks which give perfect reconstruction do not exist. Nevertheless, one can construct such filter banks which give close to perfect reconstruction [23], and this together with the fulfillment of the alias cancellation condition still make them useful. In fact, we will see in Section 8.3 that the MP3 standard take use of such filters, and this explains our previous observation that the MP3 standard does not give perfect reconstruction. Note, however, that if the filters in a classical QMF filter bank are symmetric (so that  $\lambda_{H_0}(\omega)$  is real), we have no phase distortion.

The second type of QMF filter bank is defined as follows.

**Definition 6.19.** *Alternative QMF filter banks.*

In the alternative definition of a QMF filter bank it is required that  $G_0 = (H_0)^T$  and  $G_1 = (H_1)^T$  (i.e. the filter coefficients in the forward and reverse transforms are reverse of oneanother), and that

$$\lambda_{H_1}(\omega) = \overline{\lambda_{H_0}(\omega + \pi)}. \quad (6.33)$$

The perfect reconstruction condition for an alternative QMF filter bank can be written as

$$\begin{aligned} 2 &= \lambda_{H_0}(\omega)\lambda_{G_0}(\omega) + \lambda_{H_1}(\omega)\lambda_{G_1}(\omega) = \lambda_{H_0}(\omega)\overline{\lambda_{H_0}(\omega)} + \overline{\lambda_{H_0}(\omega + \pi)}\lambda_{H_0}(\omega + \pi) \\ &= |\lambda_{H_0}(\omega)|^2 + |\lambda_{H_0}(\omega + \pi)|^2. \end{aligned}$$

We see that the perfect reconstruction property of the two definitions of QMF filter banks only differ in that the latter take absolute values. It turns out that the latter also has many interesting solutions, as we will see in Chapter 7. If we in in condition (6.23) substitute  $G_0 = (H_0)^T$  we get

$$\lambda_{H_1}(\omega) = \alpha^{-1}e^{-2id\omega}\lambda_{G_0}(\omega + \pi) = \alpha^{-1}e^{-2id\omega}\overline{\lambda_{H_0}(\omega + \pi)}.$$

If we set  $\alpha = 1, d = 0$ , we get equality here. A similar computation follows for Condition (6.24). In other words, also alternative QMF filter banks satisfy the alias cancellation condition. In the literature, a wavelet is called *orthonormal* if  $G_0 = (H_0)^T, G_1 = (H_1)^T$ . From our little computation it follows that alternative QMF filter banks with perfect reconstruction are examples of orthonormal wavelets, and correspond to orthonormal wavelets which satisfy  $\alpha = 1, d = 0$ .

For the Haar wavelet it is easily checked that  $G_0 = (H_0)^T, G_1 = (H_1)^T$ , but it does not satisfy the relation  $\lambda_{H_1}(\omega) = \overline{\lambda_{H_0}(\omega + \pi)}$ . Instead it satisfies the relation  $\lambda_{H_1}(\omega) = -\lambda_{H_0}(\omega + \pi)$ . In other words, the Haar wavelet is not an alternative QMF filter bank the way we have defined them. The difference lies only in a sign, however. This is the reason why the Haar wavelet is still listed as an alternative QMF filter bank in the literature. The additional sign leads to orthonormal wavelets which satisfy  $\alpha = -1, d = 0$  instead.

The following is clear for orthonormal wavelets.

**Theorem 6.20.** *Orthogonality of the DWT matrix.*

A DWT matrix is orthogonal (i.e. the IDWT equals the transpose of the DWT) if and only if the filters satisfy  $G_0 = (H_0)^T, G_1 = (H_1)^T$ , i.e. if and only if the MRA equals the dual MRA.

This can be proved simply by observing that, if we transpose the DWT-matrix, Theorem 6.23 says that we get an IDWT matrix with filters  $(H_0)^T, (H_1)^T$ , and this is equal to the IDWT if and only if  $G_0 = (H_0)^T, G_1 = (H_1)^T$ . It follows that QMF filter banks with perfect reconstruction give rise to orthonormal wavelets.

### Exercise 6.23: Finding FIR filters

Show that it is impossible to find a non-trivial FIR-filter which satisfies Equation (6.32).

**Exercise 6.24: The Haar wavelet as an alternative QMF filter bank**

Show that the Haar wavelet satisfies  $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega + \pi)}$ , and  $G_0 = (H_0)^T$ ,  $G_1 = (H_1)^T$ . The Haar wavelet can thus be considered as an alternative QMF filter bank.

**6.3 A generalization of the filter representation, and its use in audio coding**

It turns out that the filter representation, which we now have used for an alternative representation of a wavelet transformation, can be generalized in such a way that it also is useful for audio coding. In this section we will first define this generalization. We will then state how the MP3 standard encodes and decodes audio, and see how our generalization is connected to this. Much literature fails to elaborate on this connection. We will call our generalizations *filter bank transforms*, or simply *filter banks*. Just as for wavelets, filters are applied differently for the forward and reverse transforms. The code for this section can be found in a module called `mp3funcs`.

We start by defining the forward filter bank transform and its filters.

**Definition 6.21.** *Forward filter bank transform.*

Let  $H_0, H_1, \dots, H_{M-1}$  be  $N \times N$ -filters. A *forward filter bank transform*  $H$  produces output  $\mathbf{z} \in \mathbb{R}^N$  from the input  $\mathbf{x} \in \mathbb{R}^N$  in the following way:

- $z_{iM} = (H_0\mathbf{x})_{iM}$  for any  $i$  so that  $0 \leq iM < N$ .
- $z_{iM+1} = (H_1\mathbf{x})_{iM+1}$  for any  $i$  so that  $0 \leq iM + 1 < N$ .
- ...
- $z_{iM+(M-1)} = (H_{M-1}\mathbf{x})_{iM+(M-1)}$  for any  $i$  so that  $0 \leq iM + (M-1) < N$ .

In other words, the output of a forward filter bank transform is computed by applying filters  $H_0, H_1, \dots, H_{M-1}$  to the input, and by downsampling and assembling these so that we obtain the same number of output samples as input samples (also in this more general setting this is called critical sampling).  $H_0, H_1, \dots, H_{M-1}$  are also called *analysis filter components*, the output of filter  $H_i$  is called *channel  $i$  channel*, and  $M$  is called the number of channels. The output samples  $z_{iM+k}$  are also called the *subband samples* of channel  $k$ .

Clearly this definition generalizes the DWT and its analysis filters, since these can be obtained by setting  $M = 2$ . The DWT is thus a 2-channel forward filter bank transform. While the DWT produces the output  $\begin{pmatrix} \mathbf{c}_{m-1} \\ \mathbf{w}_{m-1} \end{pmatrix}$  from the input  $\mathbf{c}_m$ , an  $M$ -channel forward filter bank transform splits the output into  $M$  components, instead of 2. Clearly, in the matrix of a forward filter bank

transform the rows repeat cyclically with period  $M$ , similarly to MRA-matrices. In practice, the filters in a forward filter bank transform are chosen so that they concentrate on specific frequency ranges. This parallels what we saw for the filters of a wavelet, where one concentrated on high frequencies, one on low frequencies. Using a filter bank to split a signal into frequency components is also called *subband coding*. But the filters in a filter bank are usually not ideal bandpass filters. There exist a variety of different filter banks, for many different purposes [47, 39]. In Chapter 7 we will say more on how one can construct filter banks which can be used for subband coding.

Let us now turn to reverse filter bank transforms.

**Definition 6.22.** *Reverse filter bank transforms.*

Let  $G_0, G_1, \dots, G_{M-1}$  be  $N \times N$ -filters. An *reverse filter bank transform*  $G$  produces  $\mathbf{x} \in \mathbb{R}^N$  from  $\mathbf{z} \in \mathbb{R}^N$  in the following way:

Define  $\mathbf{z}_k \in \mathbb{R}^N$  as the vector where  $(\mathbf{z}_k)_{iM+k} = z_{iM+k}$  for all  $i$  so that  $0 \leq iM+k < N$ , and  $(\mathbf{z}_k)_s = 0$  for all other  $s$ .

$$\mathbf{x} = G_0 \mathbf{z}_0 + G_1 \mathbf{z}_1 + \dots + G_{M-1} \mathbf{z}_{M-1}. \quad (6.34)$$

$G_0, G_1, \dots, G_{M-1}$  are also called *synthesis filter components*.

Again, this generalizes the IDWT and its synthesis filters, and the IDWT can be seen as a 2-channel reverse filter bank transform. Also, in the matrix of a reverse filter bank transform, the columns repeat cyclically with period  $M$ , similarly to MRA-matrices. Also in this more general setting the filters  $G_i$  are in general different from the filters  $H_i$ . But we will see that, just as we saw for the Haar wavelet, there are important special cases where the analysis and synthesis filters are equal, and where their frequency responses are simply shifts of one another. It is clear that definitions 6.21 and 6.22 give the diagram for computing forward and reverse filter bank transforms shown in Figure 6.7:

Here  $\downarrow_M$  and  $\uparrow_M$  means that we extract every  $M$ 'th element in the vector, and add  $M-1$  zeros between the elements, respectively, similarly to how we previously defined  $\downarrow_2$  and  $\uparrow_2$ . Comparing Figure 6.3 with Figure 6.7 makes the similarities between wavelet transformations and the transformation used in the MP3 standard very visible: Although the filters used are different, they are subject to the same kind of processing, and can therefore be subject to the same implementations.

In general it may be that the synthesis filters do not invert exactly the analysis filters. If the synthesis system exactly inverts the analysis system, we say that we have a *perfect reconstruction filter bank*. Since the analysis system introduces undesired frequencies in the different channels, these have to cancel in the inverse transform, in order to reconstruct the input exactly.

We will have use for the following simple connection between forward and reverse filter bank transforms, which follows immediately from the definitions.

**Theorem 6.23.** *Connection between forward and reverse filter bank transforms.*

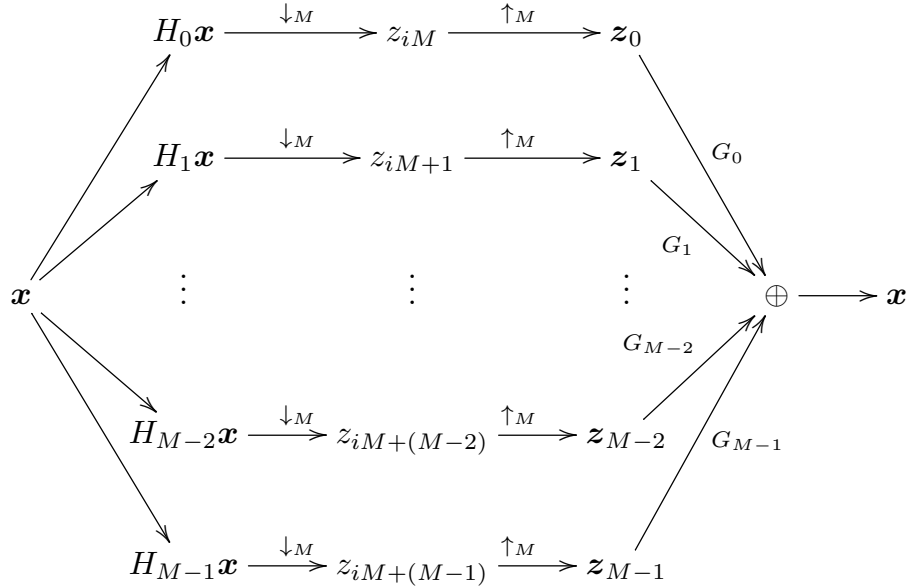


Figure 6.7: Illustration of forward and reverse filter bank transforms.

Assume that  $H$  is a forward filter bank transform with filters  $H_0, \dots, H_{N-1}$ . Then  $H^T$  is a reverse filter bank transform with filters  $G_0 = (H_0)^T, \dots, G_{N-1} = (H_{N-1})^T$ .

### 6.3.1 Forward filter bank transform in the MP3 standard

Now, let us turn to the MP3 standard. The MP3 standard document states that it applies a filter bank, and explains the following procedure for applying this filter bank, see p. 67 of the standard document (the procedure is slightly modified with mathematical terminology adapted to this book):

- Input 32 audio samples at a time.
- Build an input sample vector  $X \in \mathbb{R}^{512}$ , where the 32 new samples are placed first, all other samples are delayed with 32 elements. In particular the 32 last samples are taken out.
- Multiply  $X$  componentwise with a vector  $C$  (this vector is defined through a table in the standard), to obtain a vector  $Z \in \mathbb{R}^{512}$ . The standard calls this *windowing*.
- Compute the vector  $Y \in \mathbb{R}^{64}$  where  $Y_i = \sum_{j=0}^7 Z_{i+64j}$ . The standard calls this a *partial calculation*.

- Calculate  $S = MY \in \mathbb{R}^{32}$ , where  $M$  is the  $32 \times 64$ - matrix where  $M_{ik} = \cos((2i + 1)(k - 16)\pi/64)$ .  $S$  is called the vector of output samples, or output subband samples. The standard calls this *matrixing*.

The standard does not motivate these steps, and does not put them into the filter bank transform framework which we have established. Also, the standard does not explain how the values in the vector  $C$  have been constructed.

Let us start by proving that the steps above really corresponds to applying a forward filter bank transform, and let us state the corresponding filters of this transform. The procedure computes 32 outputs in each iteration, and each of them is associated with a subband. Therefore, from the standard we would guess that we have  $M = 32$  channels, and we would like to find the corresponding 32 filters  $H_0, H_1, \dots, H_{31}$ .

It may seem strange to use the name *matrixing* here, for something which obviously is matrix multiplication. The reason for this name must be that the at the origin of the procedure come from outside a linear algebra framework. The name *windowing* is a bit strange, too. This really does not correspond to applying a window to the sound samples as we explained in Section 3.3.1. We will see that it rather corresponds to applying a filter coefficient to a sound sample. A third and final thing which seems a bit strange is that the order of the input samples is reversed, since we are used to having the first sound samples in time with lowest index. This is perhaps more usual to do in an engineering context, and not so usual in a mathematical context. FIFO.

Clearly, the procedure above defines a linear transformation, and we need to show that this linear transformation coincides with the procedure we defined for a forward filter bank transform, for a set of 32 filters. The input to the transformation are the audio samples, which we will denote by a vector  $\mathbf{x}$ . At iteration  $s$  of the procedure above the input audio samples are  $x_{32s-512}, x_{32s-510}, \dots, x_{32s-1}$ , and  $X_i = x_{32s-i-1}$  due to the reversal of the input samples. The output to the transformation at iteration  $s$  of the procedure are the  $S_0, \dots, S_{31}$ . We assemble these into a vector  $\mathbf{z}$ , so that the output at iteration  $s$  are  $z_{32(s-1)} = S_0, z_{32(s-1)+1} = S_1, \dots, z_{32(s-1)+31} = S_{31}$ .

We will have use for the following cosine-properties, which are easily verified:

$$\cos(2\pi(n + 1/2)(k + 2Nr)/(2N)) = (-1)^r \cos(2\pi(n + 1/2)k/(2N)) \quad (6.35)$$

$$\cos(2\pi(n + 1/2)(2N - k)/(2N)) = -\cos(2\pi(n + 1/2)k/(2N)). \quad (6.36)$$

With the terminology above and using Property (6.35) the transformation can be written as

$$\begin{aligned}
 z_{32(s-1)+n} &= \sum_{k=0}^{63} \cos((2n+1)(k-16)\pi/64) Y_k = \sum_{k=0}^{63} \cos((2n+1)(k-16)\pi/64) \sum_{j=0}^7 Z_{k+64j} \\
 &= \sum_{k=0}^{63} \sum_{j=0}^7 (-1)^j \cos((2n+1)(k+64j-16)\pi/64) Z_{k+64j} \\
 &= \sum_{k=0}^{63} \sum_{j=0}^7 \cos((2n+1)(k+64j-16)\pi/64) (-1)^j C_{k+64j} X_{k+64j} \\
 &= \sum_{k=0}^{63} \sum_{j=0}^7 \cos((2n+1)(k+64j-16)\pi/64) (-1)^j C_{k+64j} x_{32s-(k+64j)-1}.
 \end{aligned}$$

Now, if we define  $\{h_r\}_{r=0}^{511}$  by  $h_{k+64j} = (-1)^j C_{k+64j}$ ,  $0 \leq j < 8, 0 \leq k < 64$ , and  $h^{(n)}$  as the filter with coefficients  $\{\cos((2n+1)(k-16)\pi/64)h_k\}_{k=0}^{511}$ , the above can be simplified as

$$\begin{aligned}
 z_{32(s-1)+n} &= \sum_{k=0}^{511} \cos((2n+1)(k-16)\pi/64) h_k x_{32s-k-1} = \sum_{k=0}^{511} (h^{(n)})_k x_{32s-k-1} \\
 &= (h^{(n)} \mathbf{x})_{32s-1} = (E_{n-31} h^{(n)} \mathbf{x})_{32(s-1)+n}.
 \end{aligned}$$

This means that the output of the procedure stated in the MP3 standard can be computed as a forward filter bank transform, and that we can choose the analysis filters as  $H_n = E_{n-31} h^{(n)}$ .

**Theorem 6.24.** *Forward filter bank transform for the MP3 standard.*

Define  $\{h_r\}_{r=0}^{511}$  by  $h_{k+64j} = (-1)^j C_{k+64j}$ ,  $0 \leq j < 8, 0 \leq k < 64$ , and  $h^{(n)}$  as the filter with coefficients  $\{\cos((2n+1)(k-16)\pi/64)h_k\}_{k=0}^{511}$ . If we define  $H_n = E_{n-31} h^{(n)}$ , the procedure stated in the MP3 standard corresponds to applying the corresponding forward filter bank transform.

The filters  $H_n$  were shown in Example 3.34 as examples of filters which concentrate on specific frequency ranges. The  $h_k$  are the filter coefficients of what is called a prototype filter. This kind of filter bank is also called a *cosine-modulated filter*. The multiplication with  $\cos(2\pi(n+1/2)(k-16)/(2N))h_k$ , modulated the filter coefficients so that the new filter has a frequency response which is simply shifted in frequency in a symmetric manner: In Exercise 3.44, we saw that, by multiplying with a cosine, we could construct new filters with real filter coefficients, which also corresponded to shifting a prototype filter in frequency. Of course, multiplication with a complex exponential would also shift the frequency response (such filter banks are called *DFT-modulated filter banks*), but the problem with this is that the new filter has complex coefficients: It will turn out that cosine-modulated filter banks can also be constructed so that they are invertible, and that one can find such filter banks where the inverse is easily found.

The effect of the delay in the definition of  $H_n$  is that, for each  $n$ , the multiplications with the vector  $\mathbf{x}$  are “aligned”, so that we can save a lot of multiplications by performing this multiplication first, and summing these. We actually save even more multiplications in the sum where  $j$  goes from 0 to 7, since we here multiply with the same cosines. The steps defined in the MP3 standard are clearly motivated by the desire to reduce the number of multiplications due to these facts. A simple arithmetic count illustrates these savings: For every 32 output samples, we have the following number of multiplications:

- The first step computes 512 multiplications.
- The second step computes 64 sums of 8 elements each, i.e. a total of  $7 \times 64 = 448$  additions (note that  $q = 512/64 = 8$ ).

The standard says nothing about how the matrix multiplication in the third step can be implemented. A direct multiplication would yield  $32 \times 64 = 2048$  multiplications, leaving a total number of multiplications at 2560. In a direct implementation of the forward filter bank transform, the computation of 32 samples would need  $32 \times 512 = 16384$  multiplications, so that the procedure sketched in the standard gives a big reduction.

The standard does not mention all possibilities for saving multiplications, however: We can reduce the number of multiplications even further, since clearly a DCT-type implementation can be used for the matrixing operation. We already have an efficient implementation for multiplication with a  $32 \times 32$  type-III cosine matrix (this is simply the IDCT). We have seen that this implementation can be chosen to reduce the number of multiplications to  $N \log_2 N/2 = 80$ , so that the total number of multiplications is  $512 + 80 = 592$ . Clearly then, when we use the DCT, the first step is the computationally most intensive part.

### 6.3.2 Reverse filter bank transform in the MP3 standard

Let us now turn to how decoding is specified in the MP3 standard, and see that we can associate this with a reverse filter bank transform. The MP3 standard also states the following procedure for decoding:

- Input 32 new subband samples as the vector  $S$ .
- Change vector  $V \in \mathbb{R}^{512}$ , so that all elements are delayed with 64 elements. In particular the 64 last elements are taken out.
- Set the first 64 elements of  $V$  as  $NS \in \mathbb{R}^{64}$ , where  $N$  is the  $64 \times 32$ -matrix where  $N_{ik} = \cos((16+i)(2k+1)\pi/64)$ . The standard also calls this matrixing.
- Build the vector  $U \in \mathbb{R}^{512}$  from  $V$  from the formulas  $U_{64i+j} = V_{128i+j}$ ,  $U_{64i+32+j} = V_{128i+96+j}$  for  $0 \leq i \leq 7$  and  $0 \leq j \leq 31$ , i.e.  $U$  is the vector where  $V$  is first split into segments of length 132, and  $U$  is constructed by assembling the first and last 32 elements of each of these segments.



- Multiply  $U$  componentwise with a vector  $D$  (this vector is defined in the standard), to obtain a vector  $W \in \mathbb{R}^{512}$ . The standard also calls this windowing.
- Compute the 32 next sound samples as  $\sum_{i=0}^{15} W_{32i+j}$ .

To interpret this also in terms of filters, rewrite first steps 4 to 6 as

$$\begin{aligned}
 x_{32(s-1)+j} &= \sum_{i=0}^{15} W_{32i+j} = \sum_{i=0}^{15} D_{32i+j} U_{32i+j} \\
 &= \sum_{i=0}^7 D_{64i+j} U_{64i+j} + \sum_{i=0}^7 D_{64i+32+j} U_{64i+32+j} \\
 &= \sum_{i=0}^7 D_{64i+j} V_{128i+j} + \sum_{i=0}^7 D_{64i+32+j} V_{128i+96+j}. \tag{6.37}
 \end{aligned}$$

The elements in  $V$  are obtained by “matrixing” different segments of the vector  $z$ . More precisely, at iteration  $s$  we have that

$$\begin{pmatrix} V_{64r} \\ V_{64r+1} \\ \vdots \\ V_{64r+63} \end{pmatrix} = N \begin{pmatrix} z_{32(s-r-1)} \\ z_{32(s-r-1)+1} \\ \vdots \\ z_{32(s-r-1)+31} \end{pmatrix},$$

so that

$$V_{64r+j} = \sum_{k=0}^{31} \cos((16+j)(2k+1)\pi/64) z_{32(s-r-1)+k}$$

for  $0 \leq j \leq 63$ . Since also

$$V_{128i+j} = V_{64(2i)+j} \quad V_{128i+96+j} = V_{64(2i+1)+j+32},$$

we can rewrite Equation (6.37) as

$$\begin{aligned}
 &\sum_{i=0}^7 D_{64i+j} \sum_{k=0}^{31} \cos((16+j)(2k+1)\pi/64) z_{32(s-2i-1)+k} \\
 &+ \sum_{i=0}^7 D_{64i+32+j} \sum_{k=0}^{31} \cos((16+j+32)(2k+1)\pi/64) z_{32(s-2i-2)+k}.
 \end{aligned}$$

Again using Relation (6.35), this can be written as

$$\begin{aligned} & \sum_{k=0}^{31} \sum_{i=0}^7 (-1)^i D_{64i+j} \cos((16+64i+j)(2k+1)\pi/64) z_{32(s-2i-1)+k} \\ & + \sum_{k=0}^{31} \sum_{i=0}^7 (-1)^i D_{64i+32+j} \cos((16+64i+j+32)(2k+1)\pi/64) z_{32(s-2i-2)+k}. \end{aligned}$$

Now, if we define  $\{g_r\}_{r=0}^{511}$  by  $g_{64i+s} = (-1)^i C_{64i+s}$ ,  $0 \leq i < 8, 0 \leq s < 64$ , and  $g^{(k)}$  as the filter with coefficients  $\{\cos((r+16)(2k+1)\pi/64)g_r\}_{r=0}^{511}$ , the above can be simplified as

$$\begin{aligned} & \sum_{k=0}^{31} \sum_{i=0}^7 (g^{(k)})_{64i+j} z_{32(s-2i-1)+k} + \sum_{k=0}^{31} \sum_{i=0}^7 (g^{(k)})_{64i+j+32} z_{32(s-2i-2)+k} \\ & = \sum_{k=0}^{31} \left( \sum_{i=0}^7 (g^{(k)})_{32(2i)+j} z_{32(s-2i-1)+k} + \sum_{i=0}^7 (g^{(k)})_{32(2i+1)+j} z_{32(s-2i-2)+k} \right) \\ & = \sum_{k=0}^{31} \sum_{r=0}^{15} (g^{(k)})_{32r+j} z_{32(s-r-1)+k}, \end{aligned}$$

where we observed that  $2i$  and  $2i+1$  together run through the values from 0 to 15 when  $i$  runs from 0 to 7. Since  $z$  has the same values as  $z_k$  on the indices  $32(s-r-1)+k$ , this can be written as

$$\begin{aligned} & = \sum_{k=0}^{31} \sum_{r=0}^{15} (g^{(k)})_{32r+j} (z_k)_{32(s-r-1)+k} \\ & = \sum_{k=0}^{31} (g^{(k)} z_k)_{32(s-1)+j+k} = \sum_{k=0}^{31} ((E_{-k} g^{(k)}) z_k)_{32(s-1)+j}. \end{aligned}$$

By substituting a general  $s$  and  $j$  we see that  $\mathbf{x} = \sum_{k=0}^{31} (E_{-k} g^{(k)}) z_k$ . We have thus proved the following.

**Theorem 6.25.** *Reverse filter bank transform for the MP3 standard.*

Define  $\{g_r\}_{r=0}^{511}$  by  $g_{64i+s} = (-1)^i C_{64i+s}$ ,  $0 \leq i < 8, 0 \leq s < 64$ , and  $g^{(k)}$  as the filter with coefficients  $\{\cos((r+16)(2k+1)\pi/64)g_r\}_{r=0}^{511}$ . If we define  $G_k = E_{-k} g^{(k)}$ , the procedure stated in the MP3 standard corresponds to applying the corresponding reverse filter bank transform.

In other words, both procedures for encoding and decoding stated in the MP3 standard both correspond to filter banks: A forward filter bank transform for the encoding, and a reverse filter bank transform for the decoding. Moreover, both filter banks can be constructed by cosine-modulating prototype filters, and the coefficients of these prototype filters are stated in the MP3 standard (up to

multiplication with an alternating sign). Note, however, that the two prototype filters may be different. When we compare the two tables for these coefficients in the standard they do indeed seem to be different. At closer inspection, however, one sees a connection: If you multiply the values in the  $D$ -table with 32, and reverse them, you get the values in the  $C$ -table. This indicates that the analysis and synthesis prototype filters are the same, up to multiplication with a scalar. This connection will be explained in Section 8.3.

While the steps defined in the MP3 standard for decoding seem a bit more complex than the steps for encoding, they are clearly also motivated by the desire to reduce the number of multiplications. In both cases (encoding and decoding), the window tables ( $C$  and  $D$ ) are in direct connection with the filter coefficients of the prototype filter: one simply adds a sign which alternates for every 64 elements. The standard document does not mention this connection, and it is perhaps not so simple to find this connection in the literature (but see [35]).

The forward and reverse filter bank transforms are clearly very related. The following result clarifies this.

**Theorem 6.26.** *Connection between the forward and reverse filter bank transforms in the MP3 standard.*

Assume that a forward filter bank transform has filters on the form  $H_i = E_{i-31}h^{(i)}$  for a prototype filter  $h$ . Then  $G = E_{481}H^T$  is a reverse filter bank transform with filters on the form  $G_k = E_{-k}g^{(k)}$ , where  $g$  is a prototype filter where the elements equal the reverse of those in  $h$ . Vice versa,  $H = E_{481}G^T$ .

*Proof.* From Theorem 6.23 we know that  $H^T$  is a reverse filter bank transform with filters

$$(H_i)^T = (E_{i-31}h^{(i)})^T = E_{31-i}(h^{(i)})^T.$$

$(h^{(i)})^T$  has filter coefficients  $\cos((2i+1)(-k-16)\pi/64)h_{-k}$ . If we delay all  $(H_i)^T$  with  $481 = 512 - 31$  elements as in the theorem, we get a total delay of  $512 - 31 + 31 - i = 512 - i$  elements, so that we get the filter

$$\begin{aligned} & E_{512-i}\{\cos((2i+1)(-k-16)\pi/64)h_{-k}\}_k \\ &= E_{-i}\{\cos((2i+1)(-(k-512)-16)\pi/64)h_{-(k-512)}\}_k \\ &= E_{-i}\{\cos((2i+1)(k+16)\pi/64)h_{-(k-512)}\}_k. \end{aligned}$$

Now, we define the prototype filter  $g$  with elements  $g_k = h_{-(k-512)}$ . This has, just as  $h$ , its support on  $[1, 511]$ , and consists of the elements from  $h$  in reverse order. If we define  $g^{(i)}$  as the filter with coefficients  $\cos((2i+1)(k+16)\pi/64)g_k$ , we see that  $E_{481}H^T$  is a reverse filter bank transform with filters  $E_{-i}g^{(i)}$ . Since  $g^{(k)}$  now has been defined as for the MP3 standard, and its elements are the reverse of those in  $h$ , the result follows.  $\square$

We will have use for this result in Section 8.3, when we find conditions on the prototype filter in order for the reverse transform to invert the forward

transform. Preferably, the reverse filter bank transform inverts exactly the forward filter bank transform. In Exercise 6.26 we construct examples which show that this is not the case. In the same exercise we also find many examples where the reverse transform does what we would expect. These examples will also be explained in Section 8.3, where we also will see how one can get around this so that we obtain a system with perfect reconstruction. It may seem strange that the MP3 standard does not do this.

In the MP3 standard, the output from the forward filter bank transform is processed further, before the result is compressed using a lossless compression method.

### Exercise 6.25: Plotting frequency responses

The values  $C_q, D_q$  can be found by calling the functions `mp3ctable`, `mp3dtable` which can be found on the book's webpage.

- a) Use your computer to verify the connection we stated between the tables  $C$  and  $D$ , i.e. that  $D_i = 32C_i$  for all  $i$ .
- b) Plot the frequency responses of the corresponding prototype filters, and verify that they both are lowpass filters. Use the connection from Theorem (6.24) to find the prototype filter coefficients from the  $C_q$ .

### Exercise 6.26: Implementing forward and reverse filter bank transforms

It is not too difficult to make implementations of the forward and reverse steps as explained in the MP3 standard. In this exercise we will experiment with this. In your code you can for simplicity assume that the input and output vectors to your methods all have lengths which are multiples of 32. Also, use the functions `mp3ctable`, `mp3dtable` mentioned in the previous exercise.

- a) Write a function `mp3forwardfbt` which implements the steps in the forward direction of the MP3 standard.
- b) Write also a function `mp3reversefbt` which implements the steps in the reverse direction.

## 6.4 Summary

We started this chapter by noting that, by reordering the target base of the DWT, the change of coordinate matrix took a particular form. From this form we understood that the DWT could be realized in terms of two filters  $H_0$  and  $H_1$ , and that the IDWT could be realized in a similar way in terms of two filters  $G_0$  and  $G_1$ . This gave rise to what we called the filter representation of wavelets. The filter representation gives an entirely different view on wavelets: instead of constructing function spaces with certain properties and deducing corresponding

filters from these, we can instead construct filters with certain properties (such as alias cancellation and perfect reconstruction), and attempt to construct corresponding mother wavelets, scaling functions, and function spaces. This strategy, which replaces problems from function theory with discrete problems, will be the subject of the next chapter. In practice this is what is done.

We stated what is required for filter bank matrices to invert each other: The frequency responses of the lowpass filters needed to satisfy a certain equation, and once this is satisfied the highpass filters can easily be obtained in the same way we previously obtained highpass filters from lowpass filters. We will return to this equation in the next chapter.

A useful consequence of the filter representation was that we could reuse existing implementations of filters to implement the DWT and the IDWT, and reuse existing theory, such as symmetric extensions. For wavelets, symmetric extensions are applied in a slightly different way, when compared to the developments which lead to the DCT. We looked at the frequency responses of the filters for the wavelets we have encountered up to now. From these we saw that  $G_0, H_0$  were lowpass filters, and that  $G_1, H_1$  were highpass filters, and we argued why this is typically the case for other wavelets as well. The filter representation was also easily generalized from 2 to  $M > 2$  filters, and such transformations had a similar interpretation in terms of splitting the input into a uniform set of frequencies. Such transforms were generally called filter bank transforms, and we saw that the processing performed by the MP3 standard could be interpreted as a certain filter bank transform, called a cosine-modulated filter bank. This is just one of many possible filter banks. In fact, the filter bank of the MP3 standard is largely outdated, since it is too simple, and as we will see it does not even give perfect reconstruction (only alias cancellation and no phase distortion). It is merely chosen here since it is the simplest to present theoretically, and since it is perhaps the best known standard for compression of sound. Other filter banks with better properties have been constructed, and they are used in more recent standards. In many of these filter banks, the filters do not partition frequencies uniformly, and have been adapted to the way the human auditory system handles the different frequencies. Different construction methods are used to construct such filter banks. The motivation behind filter bank transforms is that their output is more suitable for further processing, such as compression, or playback in an audio system, and that they have efficient implementations.

We mentioned that the MP3 standard does not say how the prototype filters were chosen. We will have more to say on what dictates their choice in Section 8.3.

There are several differences between the use of wavelet transformations in wavelet theory, and the use of filter bank transforms in signal processing theory. One is that wavelet transforms are typically applied in stages, while filter bank transforms often are not. Nevertheless, such use of filter banks also has theoretical importance, and this gives rise to what is called *tree-structured filter banks* [47]. Another difference lies in the use of the term perfect reconstruction system. In wavelet theory this is a direct consequence of the wavelet construction, since the DWT and the IDWT correspond to change of coordinates to and from the same bases. The alternative QMF filter bank was used as an example

of a filter bank which stems from signal processing, and which also shows up in wavelet transformation. In signal processing theory, one has a wider perspective, since one can design many useful systems with fast implementations when one replaces the perfect reconstruction requirement with a near perfect reconstruction requirement. One instead requires that the reverse transform gives alias cancellation. The classical QMF filter banks were an example of this. The original definition of classical QMF filter banks are from [9], and differ only in a sign from how they are defined here.

All filters we encounter in wavelets and filter banks in this book are FIR. This is just done to limit the exposition. Much useful theory has been developed using IIR-filters.

**What you should have learned in this chapter.**

- How one can find the filters of a wavelet transformation by considering its matrix and its inverse.
- Forward and reverse filter bank transforms.
- How one can implement the DWT and the IDWT with the help of the filters.
- Plot of the frequency responses for the filters of the wavelets we have considered, and their interpretation as low-pass and high-pass filters.

## Chapter 7

# Constructing interesting wavelets

In the previous chapter, from an MRA with corresponding scaling function and mother wavelet, we defined what we called a forward filter bank transform. We also defined a reverse filter bank transform, but we did not state an MRA connected to this, or prove if any such association could be made. In this chapter we will address this. We will also see, if we start with a forward and reverse filter bank transform, how we can construct corresponding MRA's, and for which transforms we can make this construction. We will see that there is a great deal of flexibility in the filter bank transforms we can construct (as this is a discrete problem). Actually it is so flexible that we can construct scaling functions/mother wavelets with any degree of regularity, and well suited for approximation of functions. This will also explain our previous interest in vanishing moments, and explain how we can find the simplest filters which give rise to a given number of vanishing moments, or a given degree of differentiability. Answers to these questions certainly transfer much more theory between wavelets and filters. Several of these filters enjoy a widespread use in applications. We will look at two of these. These are used for lossless and lossy compression in JPEG2000, which is a much used standard. These wavelets all have symmetric filters. We end the chapter by looking at a family of orthonormal wavelets with different number of vanishing moments.

### 7.1 From filters to scaling functions and mother wavelets

From Theorem 6.9 it follows that the support sizes of these dual functions are 4 and 3, respectively, so that their supports should be  $[-2, 2]$  and  $[-1, 2]$ , respectively. This is the reason why we have plotted the functions over  $[-2, 2]$ . The plots seem to confirm the support sizes we have computed.

In our first examples of wavelets in Chapter 5, we started with some bases of functions  $\phi_m$ , and deduced filters  $G_0$  and  $G_1$  from these. If we instead start with the filters  $G_0$  and  $G_1$ , what properties must they fulfill in order for us to make an association the opposite way? We should thus demand that there exist functions  $\phi, \psi$  so that

$$\phi(t) = \sum_{n=0}^{2N-1} (G_0)_{n,0} \phi_{1,n}(t) \quad (7.1)$$

$$\psi(t) = \sum_{n=0}^{2N-1} (G_1)_{n,1} \phi_{1,n}(t) \quad (7.2)$$

Using Equation (7.1), the Fourier transform of  $\phi$  is

$$\begin{aligned} \hat{\phi}(\omega) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \phi(t) e^{-i\omega t} dt = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \left( \sum_n (G_0)_{n,0} \sqrt{2} \phi(2t-n) \right) e^{-i\omega t} dt \\ &= \frac{1}{\sqrt{2}\sqrt{2\pi}} \sum_n \int_{-\infty}^{\infty} (G_0)_{n,0} \phi(t) e^{-i\omega(t+n)/2} dt \\ &= \frac{1}{\sqrt{2}} \left( \sum_n (G_0)_{n,0} e^{-i\omega n/2} \right) \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \phi(t) e^{-i(\omega/2)t} dt = \frac{\lambda_{G_0}(\omega/2)}{\sqrt{2}} \hat{\phi}(\omega/2). \end{aligned} \quad (7.3)$$

Clearly this expression can be continued recursively. We can thus state the following result.

**Theorem 7.1.**  $g_N$ .

Define

$$g_N(\omega) = \prod_{s=1}^N \frac{\lambda_{G_0}(\omega/2^s)}{\sqrt{2}} \chi_{[0,2\pi]}(2^{-N}\omega). \quad (7.4)$$

Then on  $[0, 2\pi 2^N]$  we have that  $\hat{\phi}(\nu) = g_N(\nu) \hat{\phi}(\nu/2^N)$ .

We can now prove the following.

**Lemma 7.2.**  $g_N(\nu)$  converges.

Assume that  $\sum_n (G_0)_n = \sqrt{2}$  (i.e.  $\lambda_{G_0}(0) = \sqrt{2}$ ), and that  $G_0$  is a FIR-filter. Then  $g_N(\nu)$  converges pointwise as  $N \rightarrow \infty$  to an infinitely differentiable function.

*Proof.* We need to verify that the infinite product  $\prod_{s=1}^{\infty} \frac{\lambda_{G_0}(2\pi\nu/2^s)}{\sqrt{2}}$  converges. Taking logarithms we get  $\sum_s \ln \left( \frac{\lambda_{G_0}(2\pi\nu/2^s)}{\sqrt{2}} \right)$ . To see if this series converges, we consider the ratio between two successive terms:



$$\frac{\ln\left(\frac{\lambda_{G_0}(2\pi\nu/2^{s+1})}{\sqrt{2}}\right)}{\ln\left(\frac{\lambda_{G_0}(2\pi\nu/2^s)}{\sqrt{2}}\right)}.$$

Since  $\sum_n(G_0)_n = \sqrt{2}$ , we see that  $\lambda_{G_0}(0) = \sqrt{2}$ . Since  $\lim_{\nu \rightarrow 0} \lambda_{G_0}(\nu) = \sqrt{2}$ , both the numerator and the denominator above tends to 0 (to one inside the logarithms), so that we can use L'hospital's rule on  $\frac{\ln\left(\frac{\lambda_{G_0}(\nu/2)}{\sqrt{2}}\right)}{\ln\left(\frac{\lambda_{G_0}(\nu)}{\sqrt{2}}\right)}$  to obtain

$$\frac{\lambda_{G_0}(\nu)}{\lambda_{G_0}(\nu/2)} \frac{\sum_n(G_0)_n(-in)e^{-in\nu/2}/2}{\sum_n(G_0)_n(-in)e^{-in\nu}} \rightarrow \frac{1}{2} < 1$$

as  $\nu \rightarrow 0$ . It follows that the product converges for any  $\nu$ . Clearly the convergence is absolute and uniform on compact sets, so that the limit is infinitely differentiable.  $\square$

It follows that  $\hat{\phi}$ , when  $\phi$  exists, must be an infinitely differentiable function also. Similarly we get

$$\begin{aligned} \hat{\psi}(\omega) &= \frac{1}{\sqrt{2}} \left( \sum_n (G_1)_{n-1,0} e^{-i\omega n/2} \right) \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \phi(t) e^{-i(\omega/2)t} dt \\ &= \frac{1}{\sqrt{2}} \left( \sum_n (G_1)_{n,0} e^{-i\omega(n+1)/2} \right) \hat{\phi}(\omega/2) = e^{-i\omega/2} \frac{\lambda_{G_1}(\omega/2)}{\sqrt{2}} \hat{\phi}(\omega/2). \end{aligned}$$

It follows in the same way that  $\hat{\psi}$  must be an infinitely differentiable function also.

Now consider the dual filter bank transform, as defined in Chapter 6. Its synthesis filter are  $(H_0)^T$  and  $(H_1)^T$ . If there exist a scaling function  $\tilde{\phi}$  and a mother wavelet  $\tilde{\psi}$  for the dual transform, they must in the same way be infinitely differentiable. Moreover,  $\hat{\phi}, \hat{\psi}, \tilde{\phi}, \tilde{\psi}$  can be found as infinite products of the known frequency responses. If these functions are in  $L^2\mathbb{R}$ , then we can find unique functions  $\phi, \psi, \tilde{\phi}, \tilde{\psi}$  with these as Fourier transforms.

So, our goal is to find filters so that the derived infinite products of the frequency responses lie in  $L^2(\mathbb{R})$ , and so that the constructed functions  $\phi, \psi, \tilde{\phi}, \tilde{\psi}$  give rise to “nice” wavelet bases. Some more technical requirements will be needed in order for this. In order to state these we should be clear on what we mean by a “nice” basis in this context. First of all, the bases should together span all of  $L^2(\mathbb{R})$ . But our bases are not orthogonal, so we should have some substitute for this. We will need the following definitions.

**Definition 7.3.** *Frame.*

Let  $\mathcal{H}$  be a Hilbert space. A set of vectors  $\{u_n\}_n$  is called a frame of  $\mathcal{H}$  if there exist constants  $A > 0$  and  $B > 0$  so that, for any  $f \in \mathcal{H}$ ,

$$A\|f\|^2 \leq \sum_n |\langle f, u_n \rangle|^2 \leq B\|f\|^2.$$

If  $A = B$ , the frame is said to be tight.

Note that, for a frame of  $\mathcal{H}$ , any  $f \in \mathcal{H}$  is uniquely characterized by the inner products  $\langle f, u_n \rangle$ . Indeed, if both  $a, b \in \mathcal{H}$  have the same inner products, then  $a - b \in \mathcal{H}$  have inner products 0, which implies that  $a = b$  from the left inequality.

For every frame one can find a dual frame  $\{\tilde{u}_n\}_n$  which satisfies

$$\frac{1}{B}\|f\|^2 \leq \sum_n |\langle f, \tilde{u}_n \rangle|^2 \leq \frac{1}{A}\|f\|^2,$$

and

$$f = \sum_n \langle f, u_n \rangle \tilde{u}_n = \sum_n \langle f, \tilde{u}_n \rangle u_n. \quad (7.5)$$

Thus, if the frame is tight, the dual frame is also tight.

A frame is called a Riesz basis if all its vectors also are linearly independent. One can show that the vectors in the dual frame of a Riesz basis also are linearly independent, so that the dual frame of a Riesz basis also is a Riesz basis. It is also called the dual Riesz basis. We will also need the following definition.

**Definition 7.4.** *Biorthogonal bases.*

We say that two bases  $\{\mathbf{f}_n\}_n, \{\mathbf{g}_m\}_m$  are *biorthogonal* if  $\langle \mathbf{f}_n, \mathbf{g}_m \rangle = 0$  whenever  $n \neq m$ , and 1 if  $n = m$ .

From Equation (7.5) and linear independence, it is clear that the vectors in a Riesz basis and in its dual Riesz basis are biorthogonal. In the absence of orthonormal bases for  $L^2(\mathbb{R})$ , the best we can hope for is dual Riesz bases for  $L^2(\mathbb{R})$ . The following result explains how we can obtain this from the filters.

**Proposition 7.5.** *Biorthogonality.*

Assume that the frequency responses  $\lambda_{G_0}$  and  $\lambda_{H_0}$  can be written as.

$$\frac{\lambda_{G_0}(\omega)}{\sqrt{2}} = \left( \frac{1 + e^{-i\omega}}{2} \right)^L \mathcal{F}(\omega) \quad \frac{\lambda_{H_0}(\omega)}{\sqrt{2}} = \left( \frac{1 + e^{-i\omega}}{2} \right)^{\tilde{L}} \tilde{\mathcal{F}}(\omega), \quad (7.6)$$

where  $\mathcal{F}$  and  $\tilde{\mathcal{F}}$  are trigonometric polynomials of finite degree. Assume also that, for some  $k, \tilde{k} > 0$ ,

$$B_k = \max_{\omega} |\mathcal{F}(\omega) \cdots \mathcal{F}(2^{k-1}\omega)|^{1/k} < 2^{L-1/2} \quad (7.7)$$

$$\tilde{B}_k = \max_{\omega} |\tilde{\mathcal{F}}(\omega) \cdots \tilde{\mathcal{F}}(2^{\tilde{k}-1}\omega)|^{1/\tilde{k}} < 2^{\tilde{L}-1/2} \quad (7.8)$$

Then the following hold:

- $\phi, \tilde{\phi} \in L^2(\mathbb{R})$ , and the corresponding bases  $\phi_0$  and  $\tilde{\phi}_0$  are biorthogonal.
- $\psi_{m,n}$  is a Riesz basis of  $L^2(\mathbb{R})$ .
- $\tilde{\psi}_{m,n}$  is the dual Riesz basis of  $\psi_{m,n}$ . Thus,  $\psi_{m,n}$  and  $\tilde{\psi}_{m,n}$  are biorthogonal bases, and for any  $f \in L^2(\mathbb{R})$ ,

$$f = \sum_{m,n} \langle f, \tilde{\psi}_{m,n} \rangle \psi_{m,n} = \sum_{m,n} \langle f, \psi_{m,n} \rangle \tilde{\psi}_{m,n}. \quad (7.9)$$

If also

$$B_k < 2^{L-1-m} \quad \tilde{B}_k < 2^{\tilde{L}-1-\tilde{m}}, \quad (7.10)$$

then

- $\phi, \psi$  are  $m$  times differentiable and  $\tilde{\psi}$  has  $m+1$  vanishing moments,
- $\tilde{\phi}, \tilde{\psi}$  are  $\tilde{m}$  times differentiable and  $\psi$  has  $\tilde{m}+1$  vanishing moments.

The proof for Proposition 7.5 is long, technical, and split in many stages. The entire proof can be found in [7], and we will not go through all of it, only address some simple parts of it in the following subsections. After that we will see how we can find  $G_0, H_0$  so that equations (7.6), (7.7), (7.8) are fulfilled. Before we continue on this path, several comments are in order.

1. The paper [7] much more general conditions for when filters give rise to a Riesz basis as stated here. The conditions (7.7), (7.8) are simply chosen because they apply or the filters we consider.

2. From Equation (7.6) it follows that the flatness in the frequency responses close to  $\pi$  explains how good the bases are for approximations, since the number of vanishing moments is inferred from the multiplicity of the zero at  $\pi$  for the frequency response.

3. From the result we obtain an MRA (with scaling function  $\phi$ ), and a dual MRA (with scaling function  $\tilde{\phi}$ ), as well as mother wavelets ( $\psi$  and  $\tilde{\psi}$ ), and we can define the resolution spaces  $V_m$  and the detail spaces  $W_m$  as before, as well as the “dual resolution spaces”  $\tilde{V}_m$ , (the spaces spanned by  $\tilde{\phi}_m = \{\tilde{\phi}_{m,n}\}_n$ ) and “dual detail spaces”  $\tilde{W}_m$  (the spaces spanned by  $\tilde{\psi}_m = \{\tilde{\psi}_{m,n}\}_n$ ). In general  $V_m$  is different from  $\tilde{V}_m$  (except when  $\phi = \tilde{\phi}$ ), and  $W_m$  is in general different from the orthogonal complement of  $V_{m-1}$  in  $V_m$  (except when  $\phi = \tilde{\phi}$ , when all bases are orthonormal), although constructed so that  $V_m = V_{m-1} \oplus W_{m-1}$ . Our construction thus involves two MRA’s

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m \subset \cdots \quad \tilde{V}_0 \subset \tilde{V}_1 \subset \tilde{V}_2 \subset \cdots \subset \tilde{V}_m \subset \cdots$$

where there are different scaling functions, satisfying a biorthogonality relationship. This is also called a dual multiresolution analysis.

4. The DWT and IDWT are defined as before, so that the same change of coordinates can be applied, as dictated by the filter coefficients. As will be seen below, while proving Proposition 7.5 it also follows that the bases  $\phi_0 \oplus \psi_0 \oplus \psi_1 \cdots \psi_{m-1}$  and  $\tilde{\phi}_0 \oplus \tilde{\psi}_0 \oplus \tilde{\psi}_1 \cdots \tilde{\psi}_{m-1}$  are biorthogonal (in addition to that  $\phi_m$  and  $\tilde{\phi}_m$  are biorthogonal, as stated). For  $f \in V_m$  this means that

$$f(t) = \sum_n \langle f(t), \tilde{\phi}_{m,n} \rangle \phi_{m,n} = \sum_n \langle f(t), \tilde{\phi}_{0,n} \rangle \phi_{0,n} + \sum_{m' < m, n} \langle f(t), \tilde{\psi}_{m',n} \rangle \psi_{m',n},$$

since this relationship is fulfilled for any linear combination of the  $\{\phi_{m,n}\}_n$ , or for any of the  $\{\phi_{0,n}, \psi_{m',n}\}_{m' < m, n}$ , due to biorthogonality. Similarly, for  $\tilde{f} \in \tilde{V}_m$

$$\tilde{f}(t) = \sum_n \langle \tilde{f}(t), \phi_{m,n} \rangle \tilde{\phi}_{m,n} = \sum_n \langle \tilde{f}(t), \phi_{0,n} \rangle \tilde{\phi}_{0,n} + \sum_{m' < m, n} \langle \tilde{f}(t), \psi_{m',n} \rangle \tilde{\psi}_{m',n}.$$

It follows that for  $f \in V_m$  and for  $\tilde{f} \in \tilde{V}_m$  the DWT and the IDWT and their duals can be expressed in terms of inner products as follows.

- The input to the DWT is  $c_{m,n} = \langle f, \tilde{\phi}_{m,n} \rangle$ . The output of the DWT is  $c_{0,n} = \langle f, \tilde{\phi}_{0,n} \rangle$  and  $w_{m',n} = \langle f, \tilde{\psi}_{m',n} \rangle$
- The input to the dual DWT is  $\tilde{c}_{m,n} = \langle \tilde{f}, \phi_{m,n} \rangle$ . The output of the dual DWT is  $\tilde{c}_{0,n} = \langle \tilde{f}, \phi_{0,n} \rangle$  and  $\tilde{w}_{m',n} = \langle \tilde{f}, \psi_{m',n} \rangle$ .
- in the DWT matrix, column  $k$  has entries  $\langle \phi_{1,k}, \tilde{\phi}_{0,l} \rangle$ , and  $\langle \phi_{1,k}, \tilde{\psi}_{0,l} \rangle$  (with a similar expression for the dual DWT).
- in the IDWT matrix, column  $2k$  has entries  $\langle \phi_{0,k}, \tilde{\phi}_{1,l} \rangle$ , and column  $2k+1$  has entries  $\langle \psi_{0,k}, \tilde{\phi}_{1,l} \rangle$  (with a similar expression for the dual IDWT).

Equation (7.9) comes from eliminating the  $\phi_{m,n}$  by letting  $m \rightarrow \infty$ .

5. When  $\phi = \tilde{\phi}$  (orthonormal MRA's), the approximations (finite sums) above coincide with projections onto the spaces  $V_m, \tilde{V}_m, W_m, \tilde{W}_m$ . When  $\phi \neq \tilde{\phi}$ , however, there are no reasons to believe that these approximations equal the best approximations to  $f$  from  $V_m$ . In this case we have no procedure for computing best approximations. When  $f$  is not in  $V_m, \tilde{V}_m$  we can, however, consider the approximations

$$\sum_n \langle f(t), \tilde{\phi}_{m,n} \rangle \phi_{m,n}(t) \in V_m \text{ and } \sum_n \langle f(t), \phi_{m,n} \rangle \tilde{\phi}_{m,n}(t) \in \tilde{V}_m$$

(when the MRA is orthonormal, this coincides with the best approximation). Now, we can choose  $m$  so large that  $f(t) = \sum_n c_n \phi_{m,n}(t) + \epsilon(t)$ , with  $\epsilon(t)$  a small function. The first approximation can now be written

$$\begin{aligned} \sum_n \langle \sum_{n'} c_{n'} \phi_{m,n'}(t) + \epsilon(t), \tilde{\phi}_{m,n} \rangle \phi_{m,n}(t) &= \sum_n c_n \phi_{m,n}(t) + \sum_n \langle \epsilon(t), \tilde{\phi}_{m,n} \rangle \phi_{m,n}(t) \\ &= f(t) + \sum_n \langle \epsilon(t), \tilde{\phi}_{m,n} \rangle \phi_{m,n}(t) - \epsilon(t). \end{aligned}$$

Clearly, the difference  $\sum_n \langle \epsilon(t), \tilde{\phi}_{m,n} \rangle \phi_{m,n}(t) - \epsilon(t)$  from  $f$  is small. It may, however, be hard to compute the  $c_n$  above, so that instead, as in Theorem 5.33, one uses  $\frac{2^{-m}}{\int_0^N \phi_{m,0}(t) dt} f(n/2^m) \phi_{m,n}(t)$  as an approximation to  $f$  (i.e. use sample values as  $c_n$ ) also in this more general setting.

6. Previously we were taught to think in a periodic or folded way, so that we could restrict to an interval  $[0, N]$ , and to bases of finite dimensions ( $\{\phi_{0,n}\}_{n=0}^{N-1}$ ). But the results above are only stated for wavelet bases of infinite dimension. Let us therefore say something on how the results carry over to our finite dimensional setting. If  $f \in L^2(\mathbb{R})$  we can define the function

$$f^{per}(t) = \sum_k f(t + kN) \quad f^{fold}(t) = \sum_k f(t + 2kN) + \sum_k f(2kN - t).$$

$f^{per}$  and  $f^{fold}$  are seen to be periodic with periods  $N$  and  $2N$ . It is easy to see that the restriction of  $f^{per}$  to  $[0, N]$  is in  $L^2([0, N])$ , and that the restriction of  $f^{fold}$  to  $[0, 2N]$  is in  $L^2([0, 2N])$ . In [6] it is shown that the result above extends to a similar result for the periodized/folded basis (i.e.  $\psi_{m,n}^{fold}$ ), so that we obtain dual Riesz bases for  $L^2([0, N])$  and  $L^2([0, 2N])$  instead of  $L^2(\mathbb{R})$ . The result on the vanishing moments does not extend, however. One can, however, alter some of the basis functions so that one achieves this. This simply changes some of the columns in the DWT/IDWT matrices. Note that our extension strategy is not optimal. The extension is usually not differentiable at the boundary, so that the corresponding wavelet coefficients may be large, even though the wavelet has many vanishing moments. The only way to get around this would be to find an extension strategy which gave a more regular extension. However, natural images may not have high regularity, which would make such an extension strategy useless.

**Sketch of proof for the biorthogonality in Proposition 7.5 (1).** We first show that  $\phi_0$  and  $\tilde{\phi}_0$  are biorthogonal. Recall that definition (7.4) said that  $g_N(\omega) = \prod_{s=1}^N \frac{\lambda_{G_0}(\omega/2^s)}{\sqrt{2}} \chi_{[0,2\pi]}(2^{-N}\omega)$ . Let us similarly define  $h_N(\omega) = \prod_{s=1}^N \frac{\lambda_{H_0}(\omega/2^s)}{\sqrt{2}} \chi_{[0,2\pi]}(2^{-N}\omega)$ . Recall that  $g_N \rightarrow \hat{\phi}$  and  $h_N \rightarrow \hat{\tilde{\phi}}$  pointwise as  $N \rightarrow \infty$ . We have that

$$g_{N+1}(\omega) = \frac{\lambda_{G_0}(\omega/2)}{\sqrt{2}} g_N(\omega/2) \quad h_{N+1}(\omega) = \frac{\lambda_{H_0}(\omega/2)}{\sqrt{2}} h_N(\omega/2).$$

$g_N, h_N$  are compactly supported, and equal to trigonometric polynomials on their support, so that  $g_N, h_N \in L^2(\mathbb{R})$ . Since the Fourier transform also is an isomorphism of  $L^2(\mathbb{R})$  onto itself, there exist functions  $u_N, v_N \in L^2(\mathbb{R})$  so that  $g_N = \hat{u}_N, h_N = \hat{v}_N$ . Since the above relationship equals that of Equation (7.3), with  $\hat{\phi}$  replaced with  $g_N$ , we must have that

$$u_{N+1}(t) = \sum_n (G_0)_{n,0} \sqrt{2} u_N(2t - n) \quad v_{N+1}(t) = \sum_n (H_0)_{0,n} \sqrt{2} v_N(2t - n).$$

Now, note that  $g_0(\omega) = h_0(\omega) = \chi_{[0,1]}(\omega)$ . Since  $\langle u_0, v_0 \rangle = \langle g_0, h_0 \rangle$  we get that

$$\int_{-\infty}^{-\infty} u_0(t) \overline{v_0(t-k)} dt = \int_{-\infty}^{-\infty} g_0(\nu) \overline{h_0(\nu)} e^{2\pi i k \nu} d\nu = \int_0^{2\pi} e^{-2\pi i k \nu} d\nu = \delta_{k,0}.$$

Now assume that we have proved that  $\langle u_N(t), v_N(t-k) \rangle = \delta_{k,0}$ . We then get that

$$\begin{aligned} \langle u_{N+1}(t), v_{N+1}(t-k) \rangle &= 2 \sum_{n_1, n_2} (G_0)_{n_1,0} (H_0)_{0,n_2} \langle u_N(2t - n_1), v_N(2(t-k) - n_2) \rangle \\ &= 2 \sum_{n_1, n_2} (G_0)_{n_1,0} (H_0)_{0,n_2} \langle u_N(t), v_N(t + n_1 - n_2 - 2k) \rangle \\ &= \sum_{n_1, n_2 | n_1 - n_2 = 2k} (G_0)_{n_1,0} (H_0)_{0,n_2} = \sum_n (H_0)_{0,n-2k} (G_0)_{n,0} \\ &= \sum_n (H_0)_{2k,n} (G_0)_{n,0} = \sum_n H_{2k,n} G_{n,0} = (HG)_{2k,0} = I_{2k,0} = \delta_{k,0} \end{aligned}$$

where we did the change of variables  $u = 2t - n_1$ . There is an extra argument to show that  $g_N \rightarrow_{L^2} \hat{\phi}$  (stronger than pointwise convergence as was stated above), so that also  $u_N \rightarrow_{L^2} \phi \in L^2(\mathbb{R})$ , since the Fourier transform is an isomorphism of  $L^2(\mathbb{R})$  onto itself. It follows that

$$\langle \phi_{m,k}, \tilde{\phi}_{m,l} \rangle = \lim_{N \rightarrow \infty} \langle u_N(t-k), v_N(t-l) \rangle = \delta_{k,l}.$$

While proving this one also establishes that

$$|\hat{\phi}(\omega)| \leq C(1 + |\omega|)^{-1/2-\epsilon} \quad |\hat{\tilde{\phi}}(\omega)| \leq C(1 + |\omega|)^{-1/2-\epsilon}, \quad (7.11)$$

where  $\epsilon = L - 1/2 - \log B_k / \log 2 > 0$  due to Assumption (7.7). In the paper it is proved that this condition implies that the bases constitute dual frames. The biorthogonality is used to show that they also are dual Riesz bases (i.e. that they also are linearly independent).

**Sketch of proof for the biorthogonality of in Proposition 7.5 (2).** The biorthogonality of  $\psi_{m,n}$  and  $\tilde{\psi}_{m,n}$  can be deduced from the biorthogonality of  $\phi_0$  and  $\tilde{\phi}_0$  as follows. We have that

$$\begin{aligned}
\langle \psi_{0,k}, \tilde{\psi}_{0,l} \rangle &= \sum_{n_1, n_2} (G_1)_{n_1,1} (H_1)_{1,n_2} \langle \phi_{1,n_1+2k}(t) \tilde{\phi}_{1,n_2+2l}(t) \rangle \\
&= \sum_n (G_1)_{n,1} (H_1)_{1,n+2(k-l)} = \sum_n (H_1)_{1+2(l-k),n} (G_1)_{n,1} = \sum_n H_{1+2(l-k),n} G_{n,1} \\
&= (HG)_{1+2(l-k),1} = \delta_{k,0}.
\end{aligned}$$

Similarly,

$$\begin{aligned}
\langle \tilde{\psi}_{0,k}, \tilde{\phi}_{0,l} \rangle &= \sum_{n_1, n_2} (G_1)_{n_1,1} (H_0)_{0,n_2} \langle \phi_{1,n_1+2k}(t) \tilde{\phi}_{1,n_2+2l}(t) \rangle = \sum_n (G_1)_{n,1} (H_0)_{0,n+2(k-l)} \\
&= \sum_n (H_0)_{2(l-k),n} (G_1)_{n,1} = \sum_n H_{2(l-k),n} G_{n,1} = (HG)_{2(l-k),1} = 0 \\
\langle \phi_{0,k}, \tilde{\psi}_{0,l} \rangle &= \sum_{n_1, n_2} (G_0)_{n_1,0} (H_1)_{1,n_2} \langle \phi_{1,n_1+2k}(t) \tilde{\phi}_{1,n_2+2l}(t) \rangle = \sum_n (G_0)_{n,0} (H_1)_{1,n+2(k-l)} \\
&= \sum_n (H_1)_{1+2(l-k),n} (G_0)_{n,0} = \sum_n H_{1+2(l-k),n} G_{n,0} = (HG)_{1+2(l-k),0} = 0.
\end{aligned}$$

From this we also get with a simple change of coordinates that

$$\langle \psi_{m,k}, \tilde{\psi}_{m,l} \rangle = \langle \psi_{m,k}, \tilde{\phi}_{m,l} \rangle = \langle \phi_{m,k}, \tilde{\psi}_{m,l} \rangle = 0.$$

Finally, if  $m' < m$ ,  $\phi_{m',k}, \psi_{m',k}$  can be written as a linear combination of  $\phi_{m,l}, \psi_{m,l}$ , so that  $\langle \phi_{m',k}, \tilde{\psi}_{m,l} \rangle = \langle \psi_{m',k}, \psi_{m,l} \rangle = 0$  due to what we showed above. Similarly,  $\langle \tilde{\phi}_{m',k}, \psi_{m,l} \rangle = \langle \tilde{\psi}_{m',k}, \psi_{m,l} \rangle = 0$ .

**Regularity and vanishing moments.** Now assume also that  $B_k < 2^{L-1-m}$ , so that  $\log B_k < L-1-m$ . We have that  $\epsilon = L-1/2 - \log B_k / \log 2 > L-1/2 - L+1+m = m+1/2$ , so that  $|\hat{\phi}(\omega)| < C(1+|\omega|)^{-1/2-\epsilon} = C(1+|\omega|)^{-m-1-\delta}$  for some  $\delta > 0$ . This implies that  $\hat{\phi}(\omega)(1+|\omega|)^m < C(1+|\omega|)^{-1-\delta} \in L^1$ . An important property of the Fourier transform is that  $\hat{\phi}(\omega)(1+|\omega|)^m \in L^1$  if and only if  $\phi$  is  $m$  times differentiable. This property implies that  $\phi$ , and thus  $\psi$  is  $m$  times differentiable. Similarly,  $\tilde{\phi}, \tilde{\psi}$  are  $\tilde{m}$  times differentiable.

In [7] it is also proved that if

- $\psi_{m,n}$  and  $\tilde{\psi}_{m,n}$  are biorthogonal bases,
- $\psi$  is  $m$  times differentiable with all derivatives  $\psi^{(l)}(t)$  of order  $l \leq m$  bounded, and
- $\tilde{\psi}(t) < C(1+|t|)^{m+1}$ ,

then  $\tilde{\psi}$  has  $m+1$  vanishing moments. In our case we have that  $\psi$  and  $\tilde{\psi}$  have compact support, so that these conditions are satisfied. It follows that  $\tilde{\psi}$  has  $m+1$  vanishing moments.

In the next section we will construct a wide range of forward and reverse filter bank transforms which invert each other, and which give rise to wavelets.

In [7] one checks that many of these wavelets satisfy (7.7) and (7.8) (implying that they give rise to dual Riesz bases for  $L^2(\mathbb{R})$ ), or the more general (7.10) (implying a certain regularity and a certain number of vanishing moments). Requirements on the filters lengths in order to obtain a given number of vanishing moments are also stated.

### Exercise 7.1: Implementation of the cascade algorithm

a) In the code above, we turned off symmetric extensions (the `symm`-argument is 0). Attempt to use symmetric extensions instead, and observe the new plots you obtain. Can you explain why these new plots do not show the correct functions, while the previous plots are correct?

### Exercise 7.2: Using the cascade algorithm

In Exercise 6.20 we constructed a new mother wavelet  $\hat{\psi}$  for piecewise linear functions by finding constants  $\alpha, \beta, \gamma, \delta$  so that

$$\hat{\psi} = \psi - \alpha\phi_{0,0} - \beta\phi_{0,1} - \delta\phi_{0,2} - \gamma\phi_{0,N-1}.$$

Use the cascade algorithm to plot  $\hat{\psi}$ . Do this by using the wavelet kernel for the piecewise linear wavelet (do not use the code above, since we have not implemented kernels for this wavelet yet).

## 7.2 Vanishing moments

The scaling functions and mother wavelets we constructed in Chapter 5 were very simple. They were however, enough to provide scaling functions which were differentiable. This may clearly be important for signal approximation, at least in cases where we know certain things about the regularity of the functions we approximate. However, there seemed to be nothing which dictated how the mother wavelet should be chosen in order to be useful. To see that this may pose a problem, consider the mother wavelet we chose for piecewise linear functions. Set  $N = 1$  and consider the space  $V_{10}$ , which has dimension  $2^{10}$ . When we apply a DWT, we start with a function  $g_{10} \in V_{10}$ . This may be a very good representation of the underlying data. However, when we compute  $g_{m-1}$  we just pick every other coefficient from  $g_m$ . By the time we get to  $g_0$  we are just left with the first and last coefficient from  $g_{10}$ . In some situations this may be adequate, but usually not.

**Idea 7.6.** *Approximation.*

We would like a wavelet basis to be able to represent  $f$  efficiently. By this we mean that the approximation  $f^{(m)} = \sum_n c_{0,n}\phi_{0,n} + \sum_{m' < m, n} w_{m',n}\psi_{m',n}$  to  $f$  from Observation 7.9 should converge quickly for the  $f$  we work with, as



$m$  increases. This means that, with relatively few  $\psi_{m,n}$ , we can create good approximations of  $f$ .

In this section we will address a property which the mother wavelet must fulfill in order to be useful in this respect. To motivate this property, let us first use decompose  $f \in V_m$  as

$$f = \sum_{n=0}^{N-1} \langle f, \tilde{\phi}_{0,n} \rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle f, \tilde{\psi}_{r,n} \rangle \psi_{r,n}. \quad (7.12)$$

If  $f$  is  $s$  times differentiable, it can be represented as  $f = P_s(x) + Q_s(x)$ , where  $P_s$  is a polynomial of degree  $s$ , and  $Q_s$  is a function which is very small ( $P_s$  could for instance be a Taylor series expansion of  $f$ ). If in addition  $\langle t^k, \tilde{\psi} \rangle = 0$ , for  $k = 1, \dots, s$ , we have also that  $\langle t^k, \tilde{\psi}_{r,t} \rangle = 0$  for  $r \leq s$ , so that  $\langle P_s, \tilde{\psi}_{r,t} \rangle = 0$  also. This means that Equation (7.12) can be written

$$\begin{aligned} f &= \sum_{n=0}^{N-1} \langle P_s + Q_s, \tilde{\phi}_{0,n} \rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle P_s + Q_s, \tilde{\psi}_{r,n} \rangle \psi_{r,n} \\ &= \sum_{n=0}^{N-1} \langle P_s + Q_s, \tilde{\phi}_{0,n} \rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle P_s, \tilde{\psi}_{r,n} \rangle \psi_{r,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle Q_s, \tilde{\psi}_{r,n} \rangle \psi_{r,n} \\ &= \sum_{n=0}^{N-1} \langle f, \tilde{\phi}_{0,n} \rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle Q_s, \tilde{\psi}_{r,n} \rangle \psi_{r,n}. \end{aligned}$$

Here the first sum lies in  $V_0$ . We see that the wavelet coefficients from  $W_r$  are  $\langle Q_s, \tilde{\psi}_{r,n} \rangle$ , which are very small since  $Q_s$  is small. This means that the detail in the different spaces  $W_r$  is very small, which is exactly what we aimed for. Let us summarize this as follows:

**Theorem 7.7.** *Vanishing moments.*

If a function  $f \in V_m$  is  $r$  times differentiable, and  $\tilde{\psi}$  has  $r$  vanishing moments, then  $f$  can be approximated well from  $V_0$ . Moreover, the quality of this approximation improves when  $r$  increases.

Having many vanishing moments is thus very good for compression, since the corresponding wavelet basis is very efficient for compression. In particular, if  $f$  is a polynomial of degree less than or equal to  $k-1$  and  $\tilde{\psi}$  has  $k$  vanishing moments, then the detail coefficients  $w_{m,n}$  are exactly 0. Since  $(\phi, \psi)$  and  $(\tilde{\phi}, \tilde{\psi})$  both are wavelet bases, it is equally important for both to have vanishing moments. We will in the following concentrate on the number of vanishing moments of  $\psi$ .

The Haar wavelet has one vanishing moment, since  $\tilde{\psi} = \psi$  and  $\int_0^N \psi(t) dt = 0$  as we noted in Observation 5.14. It is an exercise to see that the Haar wavelet has only one vanishing moment, i.e.  $\int_0^N t\psi(t) dt \neq 0$ .

**Theorem 7.8.** *Vanishing moments.*

Assume that the filters are chosen so that the scaling functions exist. Then the following hold

- The number of vanishing moments of  $\tilde{\psi}$  equals the multiplicity of a zero at  $\omega = \pi$  for  $\lambda_{G_0}(\omega)$ .
- The number of vanishing moments of  $\psi$  equals the multiplicity of a zero at  $\omega = \pi$  for  $\lambda_{H_0}(\omega)$ .

number of vanishing moments of  $\psi$ ,  $\tilde{\psi}$  equal the multiplicities of the zeros of the frequency responses  $\lambda_{H_0}(\omega)$ ,  $\lambda_{G_0}(\omega)$ , respectively, at  $\omega = \pi$ .

In other words, the flatter the frequency responses  $\lambda_{H_0}(\omega)$  and  $\lambda_{G_0}(\omega)$  are near high frequencies ( $\omega = \pi$ ), the better the wavelet functions are for approximation of functions. This is analogous to the smoothing filters we constructed previously, where the use of values from Pascals triangle resulted in filters which behaved like the constant function one at low frequencies. The frequency response for the Haar wavelet had just a simple zero at  $\pi$ , so that it cannot represent functions efficiently. The result also proves why we should consider  $G_0, H_0$  as lowpass filters,  $G_1, H_1$  as highpass filters.

*Proof.* We have that

$$\lambda_{s_{-\tilde{\psi}(-t)}}(\nu) = - \int_{-\infty}^{\infty} \tilde{\psi}(-t) e^{-2\pi i \nu t} dt. \quad (7.13)$$

By differentiating this expression  $k$  times w.r.t.  $\nu$  (differentiate under the integral sign) we get

$$(\lambda_{s_{-\tilde{\psi}(-t)}})^{(k)}(\nu) = - \int (-2\pi i t)^k \tilde{\psi}(t) e^{-2\pi i \nu t} dt. \quad (7.14)$$

Evaluating this at  $\nu = 0$  gives

$$(\lambda_{s_{-\tilde{\psi}(-t)}})^{(k)}(0) = - \int (-2\pi i t)^k \tilde{\psi}(t) dt. \quad (7.15)$$

From this expression it is clear that the number of vanishing moments of  $\tilde{\psi}$  equals the multiplicity of a zero at  $\nu = 0$  for  $\lambda_{s_{-\tilde{\psi}(-t)}}(\nu)$ , which we have already shown equals the multiplicity of a zero at  $\omega = 0$  for  $\lambda_{H_1}(\omega)$ . Similarly it follows that the number of vanishing moments of  $\psi$  equals the multiplicity of a zero at  $\omega = 0$  for  $\lambda_{G_1}(\omega)$ . Since we know that  $\lambda_{G_0}(\omega)$  has the same number of zeros at  $\pi$  as  $\lambda_{H_1}(\omega)$  has at 0, and  $\lambda_{H_0}(\omega)$  has the same number of zeros at  $\pi$  as  $\lambda_{G_1}(\omega)$  has at 0, the result follows.  $\square$

These results explain how we can construct  $\phi$ ,  $\psi$ ,  $\tilde{\phi}$ ,  $\tilde{\psi}$  from FIR-filters  $H_0$ ,  $H_1$ ,  $G_0$ ,  $G_1$  satisfying the perfect reconstruction condition. Also, the results explain how we can obtain such functions with as much differentiability and as many vanishing moments as we want. We will use these results in the next

section to construct interesting wavelets. There we will also cover how we can construct the simplest possible such filters.

There are some details which have been left out in this section: We have not addressed why the wavelet bases we have constructed are linearly independent, and why they span  $L^2(\mathbb{R})$ . Dual Riesz bases. These details are quite technical, and we refer to [7] for them. Let us also express what we have found in terms of analog filters.

**Observation 7.9.** *Analog filters.*

Let

$$f(t) = \sum_n c_{m,n} \phi_{m,n} = \sum_n c_{0,n} \phi_{0,n} + \sum_{m' < m, n} w_{m',n} \psi_{m',n} \in V_m.$$

$c_{m,n}$  and  $w_{m,n}$  can be computed by sampling the output of an analog filter. To be more precise,

$$c_{m,n} = \langle f, \tilde{\phi}_{m,n} \rangle = \int_0^N f(t) \tilde{\phi}_{m,n}(t) dt = \int_0^N (-\tilde{\phi}_{m,0}(-t)) f(2^{-m}n - t) dt$$

$$w_{m,n} = \langle f, \tilde{\psi}_{m,n} \rangle = \int_0^N f(t) \tilde{\psi}_{m,n}(t) dt = \int_0^N (-\tilde{\psi}_{m,0}(-t)) f(2^{-m}n - t) dt.$$

In other words,  $c_{m,n}$  can be obtained by sampling  $s_{-\tilde{\phi}_{m,0}(-t)}(f(t))$  at the points  $2^{-m}n$ ,  $w_{m,n}$  by sampling  $s_{-\tilde{\psi}_{m,0}(-t)}(f(t))$  at  $2^{-m}n$ , where the analog filters  $s_{-\tilde{\phi}_{m,0}(-t)}$ ,  $s_{-\tilde{\psi}_{m,0}(-t)}$  were defined in Theorem 1.25, i.e.

$$s_{-\tilde{\phi}_{m,0}(-t)}(f(t)) = \int_0^N (-\tilde{\phi}_{m,0}(-s)) f(t-s) ds \quad (7.16)$$

$$s_{-\tilde{\psi}_{m,0}(-t)}(f(t)) = \int_0^N (-\tilde{\psi}_{m,0}(-s)) f(t-s) ds. \quad (7.17)$$

A similar statement can be made for  $\tilde{f} \in \tilde{V}_m$ . Here the convolution kernels of the filters were as before, with the exception that  $\phi, \psi$  were replaced by  $\tilde{\phi}, \tilde{\psi}$ . Note also that, if the functions  $\tilde{\phi}, \tilde{\psi}$  are symmetric, we can increase the precision in the DWT with the method of symmetric extension also in this more general setting.

### 7.3 Characterization of wavelets w.r.t. number of vanishing moments

We have seen that wavelets are particularly suitable for approximation of functions when the mother wavelet or the dual mother wavelet have vanishing moments. The more vanishing moments they have, the more attractive they are. In this section we will attempt to characterize wavelets which have a given

number of vanishing moments. In particular we will characterize the simplest such, those where the filters have few filters coefficients.

There are two particular cases we will look at. First we will consider the case when all filters are symmetric. Then we will look at the case of orthonormal wavelets. It turns out that these two cases are mutually disjoint (except for trivial examples), but that there is a common result which can be used to characterize the solutions to both problems. We will state the results in terms of the multiplicities of the zeros of  $\lambda_{H_0}$ ,  $\lambda_{G_0}$  at  $\pi$ , which we proved are the same as the number of vanishing moments.

### 7.3.1 Symmetric filters

The main result when the filters are symmetric looks as follows.

**Theorem 7.10.** *Wavelet criteria.*

Assume that  $H_0, H_1, G_0, G_1$  are the filters of a wavelet, and that

- the filters are symmetric,
- $\lambda_{H_0}$  has a zero of multiplicity  $N_1$  at  $\pi$ ,
- $\lambda_{G_0}$  has a zero of multiplicity  $N_2$  at  $\pi$ .

Then  $N_1$  and  $N_2$  are even, and there exist a polynomial  $Q$  which satisfies

$$u^{(N_1+N_2)/2}Q(1-u) + (1-u)^{(N_1+N_2)/2}Q(u) = 2. \quad (7.18)$$

so that  $\lambda_{H_0}(\omega), \lambda_{G_0}(\omega)$  can be written on the form

$$\lambda_{H_0}(\omega) = \left(\frac{1}{2}(1 + \cos \omega)\right)^{N_1/2} Q_1\left(\frac{1}{2}(1 - \cos \omega)\right) \quad (7.19)$$

$$\lambda_{G_0}(\omega) = \left(\frac{1}{2}(1 + \cos \omega)\right)^{N_2/2} Q_2\left(\frac{1}{2}(1 - \cos \omega)\right), \quad (7.20)$$

where  $Q = Q_1Q_2$ .

*Proof.* Since the filters are symmetric,  $\lambda_{H_0}(\omega) = \lambda_{H_0}(-\omega)$  and  $\lambda_{G_0}(\omega) = \lambda_{G_0}(-\omega)$ . Since  $e^{in\omega} + e^{-in\omega} = 2\cos(n\omega)$ , and since  $\cos(n\omega)$  is the real part of  $(\cos \omega + i \sin \omega)^n$ , which is a polynomial in  $\cos^k \omega \sin^l \omega$  with  $l$  even, and since  $\sin^2 \omega = 1 - \cos^2 \omega$ ,  $\lambda_{H_0}$  and  $\lambda_{G_0}$  can both be written on the form  $P(\cos \omega)$ , with  $P$  a real polynomial.

Note that a zero at  $\pi$  in  $\lambda_{H_0}, \lambda_{G_0}$  corresponds to a factor of the form  $1 + e^{-i\omega}$ , so that we can write

$$\lambda_{H_0}(\omega) = \left(\frac{1 + e^{-i\omega}}{2}\right)^{N_1} f(e^{i\omega}) = e^{-iN_1\omega/2} \cos^{N_1}(\omega/2) f(e^{i\omega}),$$

where  $f$  is a polynomial. In order for this to be real, we must have that  $f(e^{i\omega}) = e^{iN_1\omega/2}g(e^{i\omega})$  where  $g$  is real-valued, and then we can write  $g(e^{i\omega})$  as a real polynomial in  $\cos \omega$ . This means that  $\lambda_{H_0}(\omega) = \cos^{N_1}(\omega/2)P_1(\cos \omega)$ , and similarly for  $\lambda_{G_0}(\omega)$ . Clearly this can be a polynomial in  $e^{i\omega}$  only if  $N_1$  is even. Both  $N_1$  and  $N_2$  must then be even, and we can write

$$\begin{aligned}\lambda_{H_0}(\omega) &= \cos^{N_1}(\omega/2)P_1(\cos \omega) = (\cos^2(\omega/2))^{N_1/2}P_1(1 - 2\sin^2(\omega/2)) \\ &= (\cos^2(\omega/2))^{N_1/2}Q_1(\sin^2(\omega/2)),\end{aligned}$$

where we have used that  $\cos \omega = 1 - 2\sin^2(\omega/2)$ , and defined  $Q_1$  by the relation  $Q_1(x) = P_1(1-2x)$ . Similarly we can write  $\lambda_{G_0}(\omega) = (\cos^2(\omega/2))^{N_2/2}Q_2(\sin^2(\omega/2))$  for another polynomial  $Q_2$ . Using the identities

$$\cos^2 \frac{\omega}{2} = \frac{1}{2}(1 + \cos \omega) \qquad \sin^2 \frac{\omega}{2} = \frac{1}{2}(1 - \cos \omega),$$

we see that  $\lambda_{H_0}$  and  $\lambda_{G_0}$  satisfy equations (7.19) and (7.20). With  $Q = Q_1Q_2$ , Equation (6.25) can now be rewritten as

$$\begin{aligned}2 &= \lambda_{G_0}(\omega)\lambda_{H_0}(\omega) + \lambda_{G_0}(\omega + \pi)\lambda_{H_0}(\omega + \pi) \\ &= (\cos^2(\omega/2))^{(N_1+N_2)/2}Q(\sin^2(\omega/2)) + (\cos^2((\omega + \pi)/2))^{(N_1+N_2)/2}Q(\sin^2((\omega + \pi)/2)) \\ &= (\cos^2(\omega/2))^{(N_1+N_2)/2}Q(\sin^2(\omega/2)) + (\sin^2(\omega/2))^{(N_1+N_2)/2}Q(\cos^2(\omega/2)) \\ &= (\cos^2(\omega/2))^{(N_1+N_2)/2}Q(1 - \cos^2(\omega/2)) + (1 - \cos^2(\omega/2))^{(N_1+N_2)/2}Q(\cos^2(\omega/2))\end{aligned}$$

Setting  $u = \cos^2(\omega/2)$  we see that  $Q$  must fulfill the equation

$$u^{(N_1+N_2)/2}Q(1-u) + (1-u)^{(N_1+N_2)/2}Q(u) = 2,$$

which is Equation (7.18). This completes the proof.  $\square$

While this result characterizes all wavelets with a given number of vanishing moments, it does not say which of these have fewest filter coefficients. The polynomial  $Q$  decides the length of the filters  $H_0, G_0$ , however, so that what we need to do is to find the polynomial  $Q$  of smallest degree. In this direction, note first that the polynomials  $u^{N_1+N_2}$  and  $(1-u)^{N_1+N_2}$  have no zeros in common. Bezouts theorem, proved in Section 7.3.3, states that the equation

$$u^N q_1(u) + (1-u)^N q_2(u) = 1 \tag{7.21}$$

has unique solutions  $q_1, q_2$  with  $\deg(q_1), \deg(q_2) < (N_1 + N_2)/2$ . To find these solutions, substituting  $1-u$  for  $u$  gives the following equations:

$$\begin{aligned}u^N q_1(u) + (1-u)^N q_2(u) &= 1 \\ u^N q_2(1-u) + (1-u)^N q_1(1-u) &= 1,\end{aligned}$$

and uniqueness in Bezouts theorem gives that  $q_1(u) = q_2(1 - u)$ , and  $q_2(u) = q_1(1 - u)$ . Equation (7.21) can thus be stated as

$$u^N q_2(1 - u) + (1 - u)^N q_2(u) = 1,$$

and comparing with Equation (7.18) (set  $N = (N_1 + N_2)/2$ ) we see that  $Q(u) = 2q_2(u)$ .  $u^N q_1(u) + (1 - u)^N q_2(u) = 1$  now gives

$$\begin{aligned} q_2(u) &= (1 - u)^{-N} (1 - u^N q_1(u)) = (1 - u)^{-N} (1 - u^N q_2(1 - u)) \\ &= \left( \sum_{k=0}^{N-1} \binom{N+k-1}{k} u^k + O(u^N) \right) (1 - u^N q_2(1 - u)) \\ &= \sum_{k=0}^{N-1} \binom{N+k-1}{k} u^k + O(u^N), \end{aligned}$$

where we have used the first  $N$  terms in the Taylor series expansion of  $(1 - u)^{-N}$  around 0. Since  $q_2$  is a polynomial of degree  $N - 1$ , we must have that

$$Q(u) = 2q_2(u) = 2 \sum_{k=0}^{N-1} \binom{N+k-1}{k} u^k. \quad (7.22)$$

Define  $Q^{(N)}(u) = 2 \sum_{k=0}^{N-1} \binom{N+k-1}{k} u^k$ . The first  $Q^{(N)}$  are

$$\begin{aligned} Q^{(1)}(u) &= 2 & Q^{(2)}(u) &= 2 + 4u \\ Q^{(3)}(u) &= 2 + 6u + 12u^2 & Q^{(4)}(u) &= 2 + 8u + 20u^2 + 40u^3, \end{aligned}$$

for which we compute

$$\begin{aligned} Q^{(1)}\left(\frac{1}{2}(1 - \cos \omega)\right) &= 2 \\ Q^{(2)}\left(\frac{1}{2}(1 - \cos \omega)\right) &= -e^{-i\omega} + 4 - e^{i\omega} \\ Q^{(3)}\left(\frac{1}{2}(1 - \cos \omega)\right) &= \frac{3}{4}e^{-2i\omega} - \frac{9}{2}e^{-i\omega} + \frac{19}{2} - \frac{9}{2}e^{i\omega} + \frac{3}{4}e^{2i\omega} \\ Q^{(4)}\left(\frac{1}{2}(1 - \cos \omega)\right) &= -\frac{5}{8}e^{-3i\omega} + 5e^{-2i\omega} - \frac{131}{8}e^{-i\omega} + 26 - \frac{131}{8}e^{i\omega} + 5e^{2i\omega} - \frac{5}{8}e^{3i\omega}, \end{aligned}$$

Thus in order to construct wavelets where  $\lambda_{H_0}, \lambda_{G_0}$  have as many zeros at  $\pi$  as possible, and where there are as few filter coefficients as possible, we need to compute the polynomials above, factorize them into polynomials  $Q_1$  and  $Q_2$ , and distribute these among  $\lambda_{H_0}$  and  $\lambda_{G_0}$ . Since we need real factorizations, we must in any case pair complex roots. If we do this we obtain the factorizations

$$\begin{aligned}
Q^{(1)}\left(\frac{1}{2}(1 - \cos \omega)\right) &= 2 \\
Q^{(2)}\left(\frac{1}{2}(1 - \cos \omega)\right) &= \frac{1}{3.7321}(e^{i\omega} - 3.7321)(e^{-i\omega} - 3.7321) \\
Q^{(3)}\left(\frac{1}{2}(1 - \cos \omega)\right) &= \frac{3}{4} \frac{1}{9.4438}(e^{2i\omega} - 5.4255e^{i\omega} + 9.4438) \\
&\quad \times (e^{-2i\omega} - 5.4255e^{-i\omega} + 9.4438) \\
Q^{(4)}\left(\frac{1}{2}(1 - \cos \omega)\right) &= \frac{5}{8} \frac{1}{3.0407} \frac{1}{7.1495}(e^{i\omega} - 3.0407)(e^{2i\omega} - 4.0623e^{i\omega} + 7.1495) \\
&\quad \times (e^{-i\omega} - 3.0407)(e^{-2i\omega} - 4.0623e^{-i\omega} + 7.1495), \quad (7.23)
\end{aligned}$$

The factors in these factorizations can be distributed as factors in the frequency responses of  $\lambda_{H_0}(\omega)$ , and  $\lambda_{G_0}(\omega)$ . One possibility is to let one of these frequency responses absorb all the factors, another possibility is to split the factors as evenly as possible across the two. When a frequency response absorbs more factors, the corresponding filter gets more filter coefficients. In the following examples, both factor distribution strategies will be encountered. Note that it is straightforward to use your computer to factor  $Q$  into a product of polynomials  $Q_1$  and  $Q_2$ . First the `roots` function can be used to find the roots in the polynomials. Then the `conv` function can be used to multiply together factors corresponding to different roots, to obtain the coefficients in the polynomials  $Q_1$  and  $Q_2$ .

### 7.3.2 Orthonormal wavelets

Now we turn to the case of orthonormal wavelets, i.e. where  $G_0 = (H_0)^T$ ,  $G_1 = (H_1)^T$ . For simplicity we will assume  $d = 0, \alpha = -1$  in conditions (6.23) and (6.24) (this corresponded to requiring  $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega + \pi)}$  in the definition of alternative QMF filter banks). We will also assume for simplicity that  $G_0$  is *causal*, meaning that  $t_{-1}, t_{-2}, \dots$  all are zero (the other solutions can be derived from this). We saw that the Haar wavelet was such an orthonormal wavelet. We have the following result:

**Theorem 7.11.** *Criteria for perfect reconstruction.*

Assume that  $H_0, H_1, G_0, G_1$  are the filters of an orthonormal wavelet (i.e.  $H_0 = (G_0)^T$  and  $H_1 = (G_1)^T$ ) which also is an alternative QMF filter bank (i.e.  $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega + \pi)}$ ). Assume also that  $\lambda_{G_0}(\omega)$  has a zero of multiplicity  $N$  at  $\pi$  and that  $G_0$  is causal. Then there exists a polynomial  $Q$  which satisfies

$$u^N Q(1 - u) + (1 - u)^N Q(u) = 2, \quad (7.24)$$

so that if  $f$  is another polynomial which satisfies  $f(e^{i\omega})f(e^{-i\omega}) = Q\left(\frac{1}{2}(1 - \cos \omega)\right)$ ,  $\lambda_{G_0}(\omega)$  can be written on the form

$$\lambda_{G_0}(\omega) = \left( \frac{1 + e^{-i\omega}}{2} \right)^N f(e^{-i\omega}), \quad (7.25)$$

We avoided stating  $\lambda_{H_0}(\omega)$  in this result, since the relation  $H_0 = (G_0)^T$  gives that  $\lambda_{H_0}(\omega) = \overline{\lambda_{G_0}(\omega)}$ . In particular,  $\lambda_{H_0}(\omega)$  also has a zero of multiplicity  $N$  at  $\pi$ . That  $G_0$  is causal is included to simplify the expression further.

*Proof.* The proof is very similar to the proof of Theorem 7.10.  $N$  vanishing moments and that  $G_0$  is causal means that we can write

$$\lambda_{G_0}(\omega) = \left( \frac{1 + e^{-i\omega}}{2} \right)^N f(e^{-i\omega}) = (\cos(\omega/2))^N e^{-iN\omega/2} f(e^{-i\omega}),$$

where  $f$  is a real polynomial. Also

$$\lambda_{H_0}(\omega) = \overline{\lambda_{G_0}(\omega)} = (\cos(\omega/2))^N e^{iN\omega/2} f(e^{i\omega}).$$

Condition (6.25) now says that

$$\begin{aligned} 2 &= \lambda_{G_0}(\omega)\lambda_{H_0}(\omega) + \lambda_{G_0}(\omega + \pi)\lambda_{H_0}(\omega + \pi) \\ &= (\cos^2(\omega/2))^N f(e^{i\omega})f(e^{-i\omega}) + (\sin^2(\omega/2))^N f(e^{i(\omega+\pi)})f(e^{-i(\omega+\pi)}). \end{aligned}$$

Now, the function  $f(e^{i\omega})f(e^{-i\omega})$  is symmetric around 0, so that it can be written on the form  $P(\cos \omega)$  with  $P$  a polynomial, so that

$$\begin{aligned} 2 &= (\cos^2(\omega/2))^N P(\cos \omega) + (\sin^2(\omega/2))^N P(\cos(\omega + \pi)) \\ &= (\cos^2(\omega/2))^N P(1 - 2\sin^2(\omega/2)) + (\sin^2(\omega/2))^N P(1 - 2\cos^2(\omega/2)). \end{aligned}$$

If we as in the proof of Theorem 7.10 define  $Q$  by  $Q(x) = P(1 - 2x)$ , we can write this as

$$(\cos^2(\omega/2))^N Q(\sin^2(\omega/2)) + (\sin^2(\omega/2))^N Q(\cos^2(\omega/2)) = 2,$$

which again gives Equation (7.18) for finding  $Q$ . What we thus need to do is to compute the polynomial  $Q(\frac{1}{2}(1 - \cos \omega))$  as before, and consider the different factorizations of this on the form  $f(e^{i\omega})f(e^{-i\omega})$ . Since this polynomial is symmetric,  $a$  is a root if and only  $1/a$  is, and if and only if  $\bar{a}$  is. If the real roots are

$$b_1, \dots, b_m, 1/b_1, \dots, 1/b_m,$$

and the complex roots are



$$a_1, \dots, a_n, \overline{a_1}, \dots, \overline{a_n} \text{ and } 1/a_1, \dots, 1/a_n, \overline{1/a_1}, \dots, \overline{1/a_n},$$

we can write

$$\begin{aligned} & Q\left(\frac{1}{2}(1 - \cos \omega)\right) \\ &= K(e^{-i\omega} - b_1) \dots (e^{-i\omega} - b_m) \\ &\quad \times (e^{-i\omega} - a_1)(e^{-i\omega} - \overline{a_1})(e^{-i\omega} - a_2)(e^{-i\omega} - \overline{a_2}) \dots (e^{-i\omega} - a_n)(e^{-i\omega} - \overline{a_n}) \\ &\quad \times (e^{i\omega} - b_1) \dots (e^{i\omega} - b_m) \\ &\quad \times (e^{i\omega} - a_1)(e^{i\omega} - \overline{a_1})(e^{i\omega} - a_2)(e^{i\omega} - \overline{a_2}) \dots (e^{i\omega} - a_n)(e^{i\omega} - \overline{a_n}) \end{aligned}$$

where  $K$  is a constant. We now can define the polynomial  $f$  by

$$\begin{aligned} f(e^{i\omega}) &= \sqrt{K}(e^{i\omega} - b_1) \dots (e^{i\omega} - b_m) \\ &\quad \times (e^{i\omega} - a_1)(e^{i\omega} - \overline{a_1})(e^{i\omega} - a_2)(e^{i\omega} - \overline{a_2}) \dots (e^{i\omega} - a_n)(e^{i\omega} - \overline{a_n}) \end{aligned}$$

in order to obtain a factorization  $Q\left(\frac{1}{2}(1 - \cos \omega)\right) = f(e^{i\omega})f(e^{-i\omega})$ . This concludes the proof.  $\square$

In the previous proof we note that the polynomial  $f$  is not unique - we could pair the roots in many different ways. The new algorithm is thus as follows:

- As before, write  $Q\left(\frac{1}{2}(1 - \cos \omega)\right)$  as a polynomial in  $e^{i\omega}$ , and find the roots.
- Split the roots into the two classes

$$\{b_1, \dots, b_m, a_1, \dots, a_n, \overline{a_1}, \dots, \overline{a_n}\}$$

and

$$\{1/b_1, \dots, 1/b_m, 1/a_1, \dots, 1/a_n, \overline{1/a_1}, \dots, \overline{1/a_n}\},$$

and form the polynomial  $f$  as above.

- Compute  $\lambda_{G_0}(\omega) = \left(\frac{1+e^{-i\omega}}{2}\right)^N f(e^{-i\omega})$ .

Clearly the filters obtained with this strategy are not symmetric since  $f$  is not symmetric. In Section 7.6 we will take a closer look at wavelets constructed in this way.

### 7.3.3 The proof of Bezouts theorem

**Theorem 7.12.** *Existence of polynomials.*

If  $p_1$  and  $p_2$  are two polynomials, of degrees  $n_1$  and  $n_2$  respectively, with no common zeros, then there exist unique polynomials  $q_1, q_2$ , of degree less than  $n_2, n_1$ , respectively, so that

$$p_1(x)q_1(x) + p_2(x)q_2(x) = 1. \quad (7.26)$$

*Proof.* We first establish the existence of  $q_1, q_2$  satisfying Equation (7.26). Denote by  $\deg(P)$  the degree of the polynomial  $P$ . Renumber the polynomials if necessary, so that  $n_1 \geq n_2$ . By polynomial division, we can now write

$$p_1(x) = a_2(x)p_2(x) + b_2(x),$$

where  $\deg(a_2) = \deg(p_1) - \deg(p_2)$ ,  $\deg(b_2) < \deg(p_2)$ . Similarly, we can write

$$p_2(x) = a_3(x)b_2(x) + b_3(x),$$

where  $\deg(a_3) = \deg(p_2) - \deg(b_2)$ ,  $\deg(b_3) < \deg(b_2)$ . We can repeat this procedure, so that we obtain a sequence of polynomials  $a_n(x), b_n(x)$  so that

$$b_{n-1}(x) = a_{n+1}(x)b_n(x) + b_{n+1}(x), \quad (7.27)$$

where  $\deg a_{n+1} = \deg(b_{n-1}) - \deg(b_n)$ ,  $\deg(b_{n+1}) < \deg(b_n)$ . Since  $\deg(b_n)$  is strictly decreasing, we must have that  $b_{N+1} = 0$  and  $b_N \neq 0$  for some  $N$ , i.e.  $b_{N-1}(x) = a_{N+1}(x)b_N(x)$ . Since  $b_{N-2} = a_N b_{N-1} + b_N$ , it follows that  $b_{N-2}$  can be divided by  $b_N$ , and by induction that all  $b_n$  can be divided by  $b_N$ , in particular  $p_1$  and  $p_2$  can be divided by  $b_N$ . Since  $p_1$  and  $p_2$  have no common zeros,  $b_N$  must be a nonzero constant.

Using Equation (7.27), we can write recursively

$$\begin{aligned} b_N &= b_{N-2} - a_N b_{N-1} \\ &= b_{N-2} - a_N (b_{N-3} - a_{N-1} b_{N-2}) \\ &= (1 + a_N a_{N-1}) b_{N-2} - a_N b_{N-3}. \end{aligned}$$

By induction we can write

$$b_N = a_{N,k}^{(1)} b_{N-k} + a_{N,k}^{(2)} b_{N-k-1}.$$

We see that the leading order term for  $a_{N,k}^{(1)}$  is  $a_N \cdots a_{N-k+1}$ , which has degree

$$(\deg(b_{N-2}) - \deg(b_{N-1})) + \cdots + (\deg(b_{N-k-1}) - \deg(b_{N-k})) = \deg(b_{N-k-1}) - \deg(b_{N-1}),$$

while the leading order term for  $a_{N,k}^{(2)}$  is  $a_N \cdots a_{N-k+2}$ , which similarly has order  $\deg(b_{N-k}) - \deg(b_{N-1})$ . For  $k = N - 1$  we find

$$b_N = a_{N,N-1}^{(1)}b_1 + a_{N,N-1}^{(2)}b_0 = a_{N,N-1}^{(1)}p_2 + a_{N,N-1}^{(2)}p_1, \quad (7.28)$$

with  $\deg(a_{N,N-1}^{(1)}) = \deg(p_1) - \deg(b_{N-1}) < \deg(p_1)$  (since by construction  $\deg(b_{N-1}) > 0$ ), and  $\deg(a_{N,N-1}^{(2)}) = \deg(p_2) - \deg(b_{N-1}) < \deg(p_2)$ . From Equation (7.28) it follows that  $q_1 = a_{N,N-1}^{(2)}/b_N$  and  $q_2 a_{N,N-1}^{(1)}/b_N$  satisfies Equation (7.26), and that they satisfy the required degree constraints.

Now we turn to uniqueness of solutions  $q_1, q_2$ . Assume that  $r_1, r_2$  are two other solutions to Equation (7.26). Then

$$p_1(q_1 - r_1) + p_2(q_2 - r_2) = 0.$$

Since  $p_1$  and  $p_2$  have no zeros in common this means that every zero of  $p_2$  is a zero of  $q_1 - r_1$ , with at least the same multiplicity. If  $q_1 \neq r_1$ , this means that  $\deg(q_1 - r_1) \geq \deg(p_2)$ , which is impossible since  $\deg(q_1) < \deg(p_2)$ ,  $\deg(r_1) < \deg(p_2)$ . Hence  $q_1 = r_1$ . Similarly  $q_2 = r_2$ , establishing uniqueness.  $\square$

### Exercise 7.3: Compute filters

Compute the filters  $H_0, G_0$  in Theorem 7.10 when  $N = N_1 = N_2 = 4$ , and  $Q_1 = Q^{(4)}, Q_2 = 1$ . Compute also filters  $H_1, G_1$  so that we have perfect reconstruction (note that these are not unique).

## 7.4 A design strategy suitable for lossless compression

We choose  $Q_1 = Q, Q_2 = 1$ . In this case there is no need to find factors in  $Q$ . The frequency responses of the filters in the filter factorization are

$$\begin{aligned} \lambda_{H_0}(\omega) &= \left(\frac{1}{2}(1 + \cos \omega)\right)^{N_1/2} Q^{(N)} \left(\frac{1}{2}(1 - \cos \omega)\right) \\ \lambda_{G_0}(\omega) &= \left(\frac{1}{2}(1 + \cos \omega)\right)^{N_2/2}, \end{aligned} \quad (7.29)$$

where  $N = (N_1 + N_2)/2$ . Since  $Q^{(N)}$  has degree  $N - 1$ ,  $\lambda_{H_0}$  has degree  $N_1 + N_1 + N_2 - 2 = 2N_1 + N_2 - 2$ , and  $\lambda_{G_0}$  has degree  $N_2$ . These are both even numbers, so that the filters have odd length. The names of these filters are indexed by the filter lengths, and are called *Spline wavelets*, since, as we now now will show, the scaling function for this design strategy is the *B-spline* of order  $N_2$ : we have that

$$\lambda_{G_0}(\omega) = \frac{1}{2^{N_2/2}}(1 + \cos \omega)^{N_2/2} = \cos(\omega/2)^{N_2}.$$

Letting  $s$  be the analog filter with convolution kernel  $\phi$  we can as in Equation (7.3) write

$$\begin{aligned}\lambda_s(f) &= \lambda_s(f/2^k) \prod_{i=1}^k \frac{\lambda_{G_0}(2\pi f/2^i)}{2} = \lambda_s(f/2^k) \prod_{i=1}^k \frac{\cos^{N_2}(\pi f/2^i)}{2} \\ &= \lambda_s(f/2^k) \prod_{i=1}^k \left( \frac{\sin(2\pi f/2^i)}{2 \sin(\pi f/2^i)} \right)^{N_2} = \lambda_s(f/2^k) \left( \frac{\sin(\pi f)}{2^k \sin \pi f/2^k} \right)^{N_2},\end{aligned}$$

where we have used the identity  $\cos \omega = \frac{\sin(2\omega)}{2 \sin \omega}$ . If we here let  $k \rightarrow \infty$ , and use the identity  $\lim_{f \rightarrow 0} \frac{\sin f}{f} = 1$ , we get that

$$\lambda_s(f) = \lambda_s(0) \left( \frac{\sin(\pi f)}{\pi f} \right)^{N_2}.$$

On the other hand, the frequency response of  $\chi_{[-1/2, 1/2)}(t)$

$$\begin{aligned}&= \int_{-1/2}^{1/2} e^{-2\pi i f t} dt = \left[ \frac{1}{-2\pi i f} e^{-2\pi i f t} \right]_{-1/2}^{1/2} \\ &= \frac{1}{-2\pi i f} (e^{-\pi i f} - e^{\pi i f}) = \frac{1}{-2\pi i f} 2i \sin(-\pi f) = \frac{\sin(\pi f)}{\pi f}.\end{aligned}$$

Due to this  $\left( \frac{\sin(\pi f)}{\pi f} \right)^{N_2}$  is the frequency response of  $*_{k=1}^{N_2} \chi_{[-1/2, 1/2)}(t)$ . By the uniqueness of the frequency response we have that  $\phi(t) = \hat{\phi}(0) *_{k=1}^{N_2} \chi_{[-1/2, 1/2)}(t)$ . In Exercise 7.5 you will be asked to show that this scaling function gives rise to the multiresolution analysis of functions which are piecewise polynomials which are differentiable at the borders, also called *splines*. This explains why this type of wavelet is called a spline wavelet. To be more precise, the resolution spaces are as follows.

**Definition 7.13.** *Resolution spaces of piecewise polynomials.*

We define  $V_m$  as the subspace of functions which are  $r - 1$  times continuously differentiable and equal to a polynomial of degree  $r$  on any interval of the form  $[n2^{-m}, (n+1)2^{-m}]$ .

Note that the piecewise linear wavelet can be considered as the first Spline wavelet. This is further considered in the following example.

### 7.4.1 The Spline 5/3 wavelet

For the case of  $N_1 = N_2 = 2$  when the first design strategy is used, equations (7.19) and (7.20) take the form

$$\begin{aligned}\lambda_{G_0}(\omega) &= \frac{1}{2}(1 + \cos \omega) = \frac{1}{4}e^{i\omega} + \frac{1}{2} + \frac{1}{4}e^{-i\omega} \\ \lambda_{H_0}(\omega) &= \frac{1}{2}(1 + \cos \omega)Q^{(1)}\left(\frac{1}{2}(1 - \cos \omega)\right) = \frac{1}{4}(2 + e^{i\omega} + e^{-i\omega})(4 - e^{i\omega} - e^{-i\omega}) \\ &= -\frac{1}{4}e^{2i\omega} + \frac{1}{2}e^{i\omega} + \frac{3}{2} + \frac{1}{2}e^{-i\omega} - \frac{1}{4}e^{-2i\omega}.\end{aligned}$$

The filters  $G_0, H_0$  are thus

$$G_0 = \left\{ \frac{1}{4}, \frac{1}{2}, \frac{1}{4} \right\} \quad H_0 = \left\{ -\frac{1}{4}, \frac{1}{2}, \frac{3}{2}, \frac{1}{2}, -\frac{1}{4} \right\}$$

The length of the filters are 3 and 5 in this case, so that this wavelet is called the *Spline 5/3 wavelet*. Up to a constant, the filters are seen to be the same as those of the alternative piecewise linear wavelet, see Example 6.3. Now, how do we find the filters  $(G_1, H_1)$ ? Previously we saw how to find the constant  $\alpha$  in Theorem 6.16 when we knew one of the two pairs  $(G_0, G_1), (H_0, H_1)$ . This was the last part of information we needed in order to construct the other two filters. Here we know  $(G_0, H_0)$  instead. In this case it is even easier to find  $(G_1, H_1)$  since we can set  $\alpha = 1$ . This means that  $(G_1, H_1)$  can be obtained simply by adding alternating signs to  $(G_0, H_0)$ , i.e. they are the corresponding high-pass filters. We thus can set

$$G_1 = \left\{ -\frac{1}{4}, -\frac{1}{2}, \frac{3}{2}, -\frac{1}{2}, -\frac{1}{4} \right\} \quad H_1 = \left\{ -\frac{1}{4}, \frac{1}{2}, -\frac{1}{4} \right\}.$$

We have now found all the filters. It is clear that the forward and reverse filter bank transforms here differ only by multiplication with a constant from those of the the alternative piecewise linear wavelet, so that this gives the same scaling function and mother wavelet as that wavelet.

The coefficients for the Spline wavelets are always dyadic fractions, and are therefore suitable for lossless compression, as they can be computed using low precision arithmetic and bitshift operations. The particular Spline wavelet from Example 7.4.1 is used for lossless compression in the JPEG2000 standard.

### Exercise 7.4: Viewing the frequency response

In this exercise we will see how we can view the frequency responses, scaling functions and mother wavelets for any spline wavelet.

a) Plot the frequency responses of the filters of some of the spline wavelets in this section.

**Exercise 7.5: Wavelets based on higher degree polynomials**

Show that  $B_r(t) = \ast_{k=1}^r \chi_{[-1/2, 1/2)}(t)$  is  $r - 2$  times differentiable, and equals a polynomial of degree  $r - 1$  on subintervals of the form  $[n, n + 1]$ . Explain why these functions can be used as basis for the spaces  $V_j$  of functions which are piecewise polynomials of degree  $r - 1$  on intervals of the form  $[n2^{-m}, (n + 1)2^{-m}]$ , and  $r - 2$  times differentiable.  $B_r$  is also called the  $B$ -spline of order  $r$ .

**7.5 A design strategy suitable for lossy compression**

The factors of  $Q$  are split evenly among  $Q_1$  and  $Q_2$ . In this case we need to factorize  $Q$  into a product of real polynomials. This can be done by finding all roots, and pairing the complex conjugate roots into real second degree polynomials (if  $Q$  is real, its roots come in conjugate pairs), and then distribute these as evenly as possible among  $Q_1$  and  $Q_2$ . These filters are called the CDF-wavelets, after Cohen, Daubechies, and Feauveau, who discovered them.

**Example 7.6: The CDF 9/7 wavelet**

We choose  $N_1 = N_2 = 4$ . In Equation (7.23) we pair inverse terms to obtain

$$\begin{aligned} Q^{(3)} \left( \frac{1}{2}(1 - \cos \omega) \right) &= \frac{5}{8} \frac{1}{3.0407} \frac{1}{7.1495} (e^{i\omega} - 3.0407)(e^{-i\omega} - 3.0407) \\ &\quad \times (e^{2i\omega} - 4.0623e^{i\omega} + 7.1495)(e^{-2i\omega} - 4.0623e^{-i\omega} + 7.1495) \\ &= \frac{5}{8} \frac{1}{3.0407} \frac{1}{7.1495} (-3.0407e^{i\omega} + 10.2456 - 3.0407e^{-i\omega}) \\ &\quad \times (7.1495e^{2i\omega} - 33.1053e^{i\omega} + 68.6168 - 33.1053e^{-i\omega} + 7.1495e^{-2i\omega}). \end{aligned}$$

We can write this as  $Q_1 Q_2$  with  $Q_1(0) = Q_2(0)$  when

$$\begin{aligned} Q_1(\omega) &= -1.0326e^{i\omega} + 3.4795 - 1.0326e^{-i\omega} \\ Q_2(\omega) &= 0.6053e^{2i\omega} - 2.8026e^{i\omega} + 5.8089 - 2.8026e^{-i\omega} + 0.6053e^{-2i\omega}, \end{aligned}$$

from which we obtain

$$\begin{aligned}
\lambda_{G_0}(\omega) &= \left(\frac{1}{2}(1 + \cos \omega)\right)^2 Q_1(\omega) \\
&= -0.0645e^{3i\omega} - 0.0407e^{2i\omega} + 0.4181e^{i\omega} + 0.7885 \\
&\quad + 0.4181e^{-i\omega} - 0.0407e^{-2i\omega} - 0.0645e^{-3i\omega} \\
\lambda_{H_0}(\omega) &= \left(\frac{1}{2}(1 + \cos \omega)\right)^2 40Q_2(\omega) \\
&= 0.0378e^{4i\omega} - 0.0238e^{3i\omega} - 0.1106e^{2i\omega} + 0.3774e^{i\omega} + 0.8527 \\
&\quad + 0.3774e^{-i\omega} - 0.1106e^{-2i\omega} - 0.0238e^{-3i\omega} + 0.0378e^{-4i\omega}.
\end{aligned}$$

The filters  $G_0, H_0$  are thus

$$\begin{aligned}
G_0 &= \{0.0645, 0.0407, -0.4181, \underline{-0.7885}, -0.4181, 0.0407, 0.0645\} \\
H_0 &= \{-0.0378, 0.0238, 0.1106, -0.3774, \underline{-0.8527}, -0.3774, 0.1106, 0.0238, -0.0378\}.
\end{aligned}$$

The corresponding frequency responses are plotted in Figure 7.1.

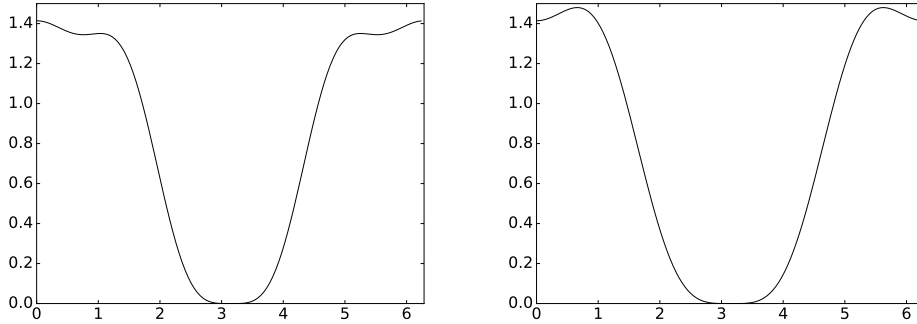


Figure 7.1: The frequency responses  $\lambda_{H_0}(\omega)$  (left) and  $\lambda_{G_0}(\omega)$  (right) for the CDF 9/7 wavelet.

It is seen that both filters are low-pass filters also here, and that they are closer to an ideal bandpass filter. Here, the frequency response acts even more like the constant zero function close to  $\pi$ , proving that our construction has worked. We also get

$$\begin{aligned}
G_1 &= \{-0.0378, -0.0238, 0.1106, 0.3774, \underline{-0.8527}, 0.3774, 0.1106, -0.0238, -0.0378\} \\
H_1 &= \{-0.0645, 0.0407, 0.4181, \underline{-0.7885}, 0.4181, 0.0407, -0.0645\}.
\end{aligned}$$

The lengths of the filters are 9 and 7 in this case, so that this wavelet is called the *CDF 9/7 wavelet*. This wavelet is for instance used for lossy compression with JPEG2000 since it gives a good tradeoff between complexity and compression.

In Example 6.3 we saw that we had analytical expressions for the scaling functions and the mother wavelet, but that we could not obtain this for the dual functions. For the CDF 9/7 wavelet it turns out that none of the four functions have analytical expressions. Let us therefore use the cascade algorithm to plot these functions. Note first that since  $G_0$  has 7 filter coefficients, and  $G_1$  has 9 filter coefficients, it follows from Theorem 6.9 that  $\text{supp}(\phi) = [-3, 3]$ ,  $\text{supp}(\psi) = [-3, 4]$ ,  $\text{supp}(\tilde{\phi}) = [-4, 4]$ , and  $\text{supp}(\tilde{\psi}) = [-3, 4]$ . The scaling functions and mother wavelets over these supports are shown in Figure 7.2. Again they have irregular shapes, but now at least the functions and dual functions more resemble each other.

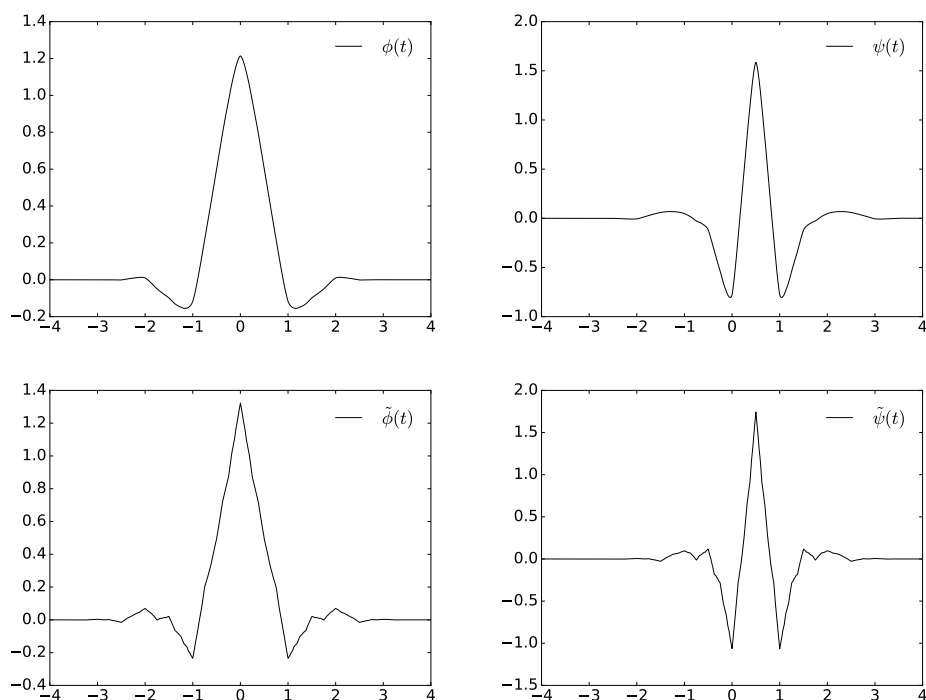


Figure 7.2: Scaling functions and mother wavelets for the CDF 9/7 wavelet.

In the above example there was a unique way of factoring  $Q$  into a product of real polynomials. For higher degree polynomials there is no unique way to form to distribute the factors, and we will not go into what strategy can be used for this. In general, the steps we must go through are as follows:

- Compute the polynomial  $Q$ , and find its roots.
- Pair complex conjugate roots into real second degree polynomials, and form polynomials  $Q_1, Q_2$ .
- Compute the coefficients in equations (7.19) and (7.20).



## 7.6 Orthonormal wavelets

Since the filters here are not symmetric, the method of symmetric extension does not work in the same simple way as before. This partially explains why symmetric filters are used more often: They may not be as efficient in representing functions, since the corresponding basis is not orthogonal, but their simple implementation still makes them attractive.

In Theorem 7.11 we characterized orthonormal wavelets where  $G_0$  was causal. All our filters have an even number, say  $2L$ , of filter coefficients. We can also find an orthonormal wavelet where  $H_0$  has a minimum possible overweight of filter coefficients with negative indices,  $H_1$  a minimum possible overweight of positive indices, i.e. that the filters can be written with the following compact notation:

$$H_0 = \{t_{-L}, \dots, t_{-1}, \underline{t_0}, t_1, \dots, t_{L-1}\} \quad H_1 = \{s_{-L+1}, \dots, s_{-1}, \underline{s_0}, s_1, \dots, s_L\}. \quad (7.30)$$

To see why, Theorem 6.16 says that we first can shift the filter coefficients of  $H_0$  so that it has this form (we then need to shift  $G_0$  in the opposite direction).  $H_1, G_1$  then can be defined by  $\alpha = 1$  and  $d = 0$ . We will follow this convention for the orthonormal wavelets we look at.

The polynomials  $Q^{(0)}$ ,  $Q^{(1)}$ , and  $Q^{(2)}$  require no further action to obtain the factorization  $f(e^{i\omega})f(e^{-i\omega}) = Q\left(\frac{1}{2}(1 - \cos\omega)\right)$ . The polynomial  $Q^{(3)}$  in Equation (7.23) can be factored further as

$$Q^{(3)}\left(\frac{1}{2}(1 - \cos\omega)\right) = \frac{5}{8} \frac{1}{3.0407} \frac{1}{7.1495} (e^{-3i\omega} - 7.1029e^{-2i\omega} + 19.5014e^{-i\omega} - 21.7391) \\ \times (e^{3i\omega} - 7.1029e^{2i\omega} + 19.5014e^{i\omega} - 21.7391),$$

which gives that  $f(e^{i\omega}) = \sqrt{\frac{5}{8} \frac{1}{3.0407} \frac{1}{7.1495}} (e^{3i\omega} - 7.1029e^{2i\omega} + 19.5014e^{i\omega} - 21.7391)$ .

This factorization is not unique, however. This gives the frequency response  $\lambda_{G_0}(\omega) = \left(\frac{1+e^{-i\omega}}{2}\right)^N f(e^{-i\omega})$  as

$$\frac{1}{2}(e^{-i\omega} + 1)\sqrt{2} \\ \frac{1}{4}(e^{-i\omega} + 1)^2 \sqrt{\frac{1}{3.7321}}(e^{-i\omega} - 3.7321) \\ \frac{1}{8}(e^{-i\omega} + 1)^3 \sqrt{\frac{3}{4} \frac{1}{9.4438}}(e^{-2i\omega} - 5.4255e^{-i\omega} + 9.4438) \\ \frac{1}{16}(e^{-i\omega} + 1)^4 \sqrt{\frac{5}{8} \frac{1}{3.0407} \frac{1}{7.1495}}(e^{-3i\omega} - 7.1029e^{-2i\omega} + 19.5014e^{-i\omega} - 21.7391),$$

which gives the filters

$$G_0 = (H_0)^T = (\sqrt{2}/2, \sqrt{2}/2)$$

$$G_0 = (H_0)^T = (-0.4830, -0.8365, -0.2241, 0.1294)$$

$$G_0 = (H_0)^T = (0.3327, 0.8069, 0.4599, -0.1350, -0.0854, 0.0352)$$

$$G_0 = (H_0)^T = (-0.2304, -0.7148, -0.6309, 0.0280, 0.1870, -0.0308, -0.0329, 0.0106)$$

so that we get 2, 4, 6 and 8 filter coefficients in  $G_0 = (H_0)^T$ . We see that the filter coefficients when  $N = 1$  are those of the Haar wavelet. The three next filters we have not seen before. The filter  $G_1 = (H_1)^T$  can be obtained from the relation  $\lambda_{G_1}(\omega) = -\lambda_{G_0}(\omega + \pi)$ , i.e. by reversing the elements and adding an alternating sign, plus an extra minus sign, so that

$$G_1 = (H_1)^T = (\sqrt{2}/2, -\sqrt{2}/2)$$

$$G_1 = (H_1)^T = (0.1294, 0.2241, -0.8365, 0.4830)$$

$$G_1 = (H_1)^T = (0.0352, 0.0854, -0.1350, -0.4599, 0.8069, -0.3327)$$

$$G_1 = (H_1)^T = (0.0106, 0.0329, -0.0308, -0.1870, 0.0280, 0.6309, -0.7148, 0.2304).$$

Frequency responses are shown in Figure 7.3 for  $N = 1$  to  $N = 6$ . It is seen that the frequency responses get increasingly flatter as  $N$  increases. The frequency responses are now complex, so their magnitudes are plotted.

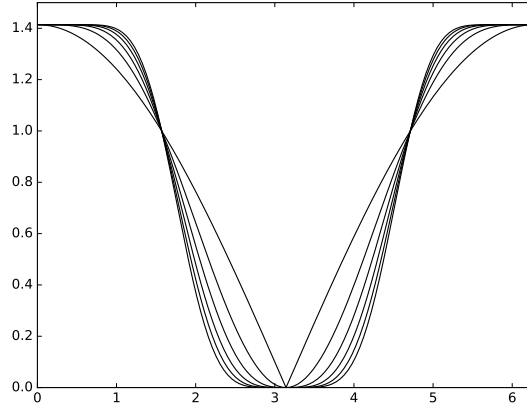


Figure 7.3: The magnitudes  $|\lambda_{G_0}(\omega)| = |\lambda_{H_0}(\omega)|$  for the first orthonormal wavelets.

Clearly these filters have low-pass characteristic. We also see that the high-pass characteristics resemble the low-pass characteristics. We also see that the

frequency response gets flatter near the high and low frequencies, as  $N$  increases. One can verify that this is the case also when  $N$  is increased further. The shapes for higher  $N$  are very similar to the frequency responses of those filters used in the MP3 standard (see Figure 3.10). One difference is that the support of the latter is concentrated on a smaller set of frequencies.

The way we have defined the filters, one can show in the same way as in the proof of Theorem 6.9 that, when all filters have  $2N$  coefficients,  $\phi = \tilde{\phi}$  has support  $[-N + 1, N]$ ,  $\psi = \tilde{\psi}$  has support  $[-N + 1/2, N - 1/2]$  (i.e. the support of  $\psi$  is symmetric about the origin). In particular we have that

- for  $N = 2$ :  $\text{supp}(\phi) = \text{supp}(\psi) = [-1, 2]$ ,
- for  $N = 3$ :  $\text{supp}(\phi) = \text{supp}(\psi) = [-2, 3]$ ,
- for  $N = 4$ :  $\text{supp}(\phi) = \text{supp}(\psi) = [-3, 4]$ .

The scaling functions and mother wavelets are shown in Figure 7.4. All functions have been plotted over  $[-4, 4]$ , so that all these support sizes can be verified. Also here we have used the cascade algorithm to approximate the functions.

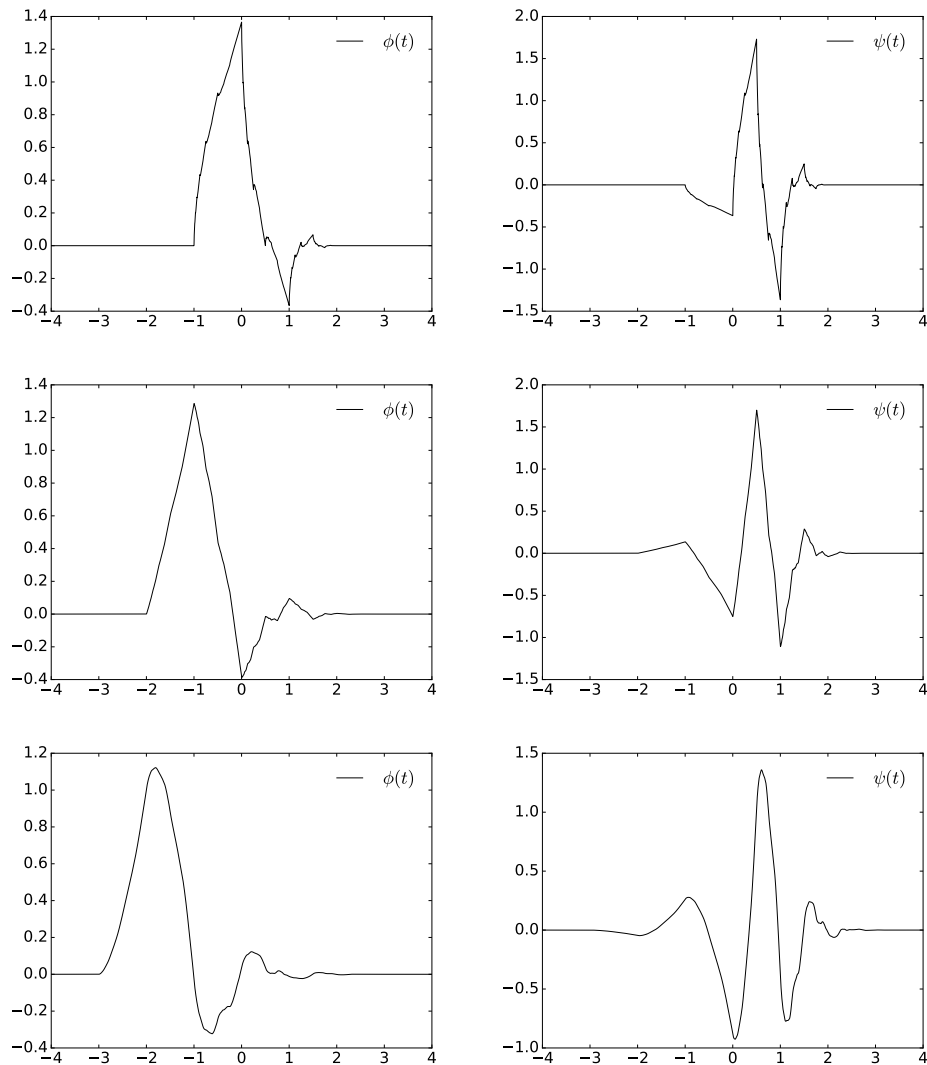


Figure 7.4: The scaling functions and mother wavelets for orthonormal wavelets with  $N$  vanishing moments, for different values of  $N$ .

## 7.7 Summary

We started the section by showing how filters from filter bank matrices can give rise to scaling functions and mother wavelets. We saw that we obtained dual function pairs in this way, which satisfied a mutual property called biorthogonality. We then saw how differentiable scaling functions or mother wavelets with vanishing moments could be constructed, and we saw how we could construct the simplest such. These could be found in terms of the frequency responses

of the involved filters. Finally we studied some examples with applications to image compression.

For the wavelets we constructed in this chapter, we also plotted the corresponding scaling functions and mother wavelets (see figures 7.2, 7.4). The importance of these functions are that they are particularly suited for approximation of regular functions, and providing a compact representation of these functions which is localized in time. It seems difficult to guess that these strange shapes are connected to such approximation. Moreover, it may seem strange that, although these functions are useful, we can't write down exact expressions for them, and they are only approximated in terms of the Cascade Algorithm.

In the literature, the orthonormal wavelets with compact support we have constructed were first constructed in [12]. Biorthogonal wavelets were first constructed in [7].

## Chapter 8

# The polyphase representation and wavelets

In Chapter 6 we saw that we could express wavelet transformations and more general transformations in terms of filters. Through this we obtained intuition for what information the different parts of a wavelet transformation represent, in terms of lowpass and highpass filters. We also obtained some insight into the filters used in the transformation used in the MP3 standard. We expressed the DWT and IDWT implementations in terms of what we called kernel transformations, and these were directly obtained from the filters of the wavelet.

We have looked at many wavelets, however, but have only stated the kernel transformation for the Haar wavelet. In order to use these wavelets in sound and image processing, or in order to use the cascade algorithm to plot the corresponding scaling functions and mother wavelets, we need to make these kernel transformations. This will be one of the goals in this chapter. This will be connected to what we will call the *polyphase representation* of the wavelet. This representation will turn out to be useful for different reasons than the filter representation as well. First of all, with the polyphase representation, transformations can be viewed as block matrices where the blocks are filters. This allows us to prove results in a different way than for filter bank transforms, since we can prove results through block matrix manipulation. There will be two major results we will prove in this way.

First, in Section 8.1 we obtain a factorization of a wavelet transformation into sparse matrices, called elementary lifting matrices. We will show that this factorization reduces the number of arithmetic operations, and also enables us to compute the DWT in-place, in a similar way to how the FFT could be computed in-place after a bit-reversal. This is important: recall that we previously factored a filter into a product of smaller filters which is useful for efficient hardware implementations. But this did not address the fact that only every second component of the filters needs to be evaluated in the DWT, something any efficient implementation of the DWT should take into account.

The factorization into sparse matrices will be called the *lifting factorization*, and it will be clear from this factorization how the wavelet kernels and their duals can be implemented. We will also see how we can use the polyphase representation to prove the remaining parts of Theorem 6.16.

Secondly, in Section 8.3 we will use the polyphase representation to analyze how the forward and reverse filter bank transforms from the MP3 standard can be chosen in order for us to have perfect or near perfect reconstruction. Actually, we will obtain a factorization of the polyphase representation into block matrices also here, and the conditions we need to put on the prototype filters will be clear from this.

## 8.1 The polyphase representation and the lifting factorization

Let us start by defining the basic concepts in the polyphase representation.

**Definition 8.1.** *Polyphase components and representation.*

Assume that  $S$  is a matrix, and that  $M$  is a number. By the *polyphase components* of  $S$  we mean the matrices  $S^{(i,j)}$  defined by  $S_{r_1, r_2}^{(i,j)} = S_{i+r_1M, j+r_2M}$ , i.e. the matrices obtained by taking every  $M$ 'th component of  $S$ . By the *polyphase representation* of  $S$  we mean the block matrix with entries  $S^{(i,j)}$ .

The polyphase representation applies in particular for vectors. Since a vector  $\mathbf{x}$  only has one column, we write  $\mathbf{x}^{(p)}$  for its polyphase components. As an examples consider the  $6 \times 6$  MRA-matrix

$$S = \begin{pmatrix} 2 & 3 & 0 & 0 & 0 & 1 \\ 4 & 5 & 6 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 \\ 6 & 0 & 0 & 0 & 4 & 5 \end{pmatrix}. \quad (8.1)$$

The polyphase components of  $S$  are

$$\begin{aligned} S^{(0,0)} &= \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} & S^{(0,1)} &= \begin{pmatrix} 3 & 0 & 1 \\ 1 & 3 & 0 \\ 0 & 1 & 3 \end{pmatrix} \\ S^{(1,0)} &= \begin{pmatrix} 4 & 6 & 0 \\ 0 & 4 & 6 \\ 6 & 0 & 4 \end{pmatrix} & S^{(1,1)} &= \begin{pmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{pmatrix} \end{aligned}$$

We will mainly be concerned with polyphase representations of MRA matrices. For such matrices we have the following result (this result can be stated more generally for any filter bank transform).

**Theorem 8.2.** *Similarity.*

When  $S$  is an MRA-matrix, the polyphase components  $S^{(i,j)}$  are filters (in general different from the filters considered in Chapter 6), i.e. the polyphase representation is a  $2 \times 2$ -block matrix where all blocks are filters. Also,  $S$  is similar to its polyphase representation, through a permutation matrix  $P$  which places the even-indexed elements first.

To see why, note that when  $P$  is the permutation matrix defined above, then  $PS$  consists of  $S$  with the even-indexed rows grouped first, and since also  $SP^T = (PS^T)^T$ ,  $SP^T$  groups the even-indexed columns first. From these observations it is clear that  $PSP^T$  is the polyphase representation of  $S$ , so that  $S$  is similar to its polyphase representation.

We also have the following result on the polyphase representation. This result is easily proved from manipulation with block matrices, and is therefore left to the reader.

**Theorem 8.3.** *Products and transpose.*

Let  $A$  and  $B$  be (forward or reverse) filter bank transforms, and denote the corresponding polyphase components by  $A^{(i,j)}$ ,  $B^{(i,j)}$ . The following hold

- $C = AB$  is also a filter bank transform, with polyphase components  $C^{(i,j)} = \sum_k A^{(i,k)} B^{(k,j)}$ .
- $A^T$  is also a filter bank transform, with polyphase components  $((A^T)^{(i,j)})_{k,l} = (A^{(j,i)})_{l,k}$ .

Also, the polyphase components of the identity matrix is the  $M \times M$ -block matrix with the identity matrix on the diagonal, and  $\mathbf{0}$  elsewhere.

To see an application of the polyphase representation, let us prove the final ingredient of Theorem 6.16. We need to prove the following:

**Theorem 8.4.** *Criteria for perfect reconstruction.*

For any set of FIR filters  $H_0, H_1, G_0, G_1$  which give perfect reconstruction, there exist  $\alpha \in \mathbb{R}$  and  $d \in \mathbb{Z}$  so that

$$\lambda_{H_1}(\omega) = \alpha^{-1} e^{-2id\omega} \lambda_{G_0}(\omega + \pi) \quad (8.2)$$

$$\lambda_{G_1}(\omega) = \alpha e^{2id\omega} \lambda_{H_0}(\omega + \pi). \quad (8.3)$$

*Proof.* Let  $H^{(i,j)}$  be the polyphase components of  $H$ ,  $G^{(i,j)}$  the polyphase components of  $G$ .  $GH = I$  means that

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix} \begin{pmatrix} H^{(0,0)} & H^{(0,1)} \\ H^{(1,0)} & H^{(1,1)} \end{pmatrix} = \begin{pmatrix} I & \mathbf{0} \\ \mathbf{0} & I \end{pmatrix}.$$

If we here multiply with  $\begin{pmatrix} G^{(1,1)} & -G^{(0,1)} \\ -G^{(1,0)} & G^{(0,0)} \end{pmatrix}$  on both sides to the left, or with  $\begin{pmatrix} H^{(1,1)} & -H^{(0,1)} \\ -H^{(1,0)} & H^{(0,0)} \end{pmatrix}$  on both sides to the right, we get



$$\begin{pmatrix} G^{(1,1)} & -G^{(0,1)} \\ -G^{(1,0)} & G^{(0,0)} \end{pmatrix} = \begin{pmatrix} (G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)})H^{(0,0)} & (G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)})H^{(0,1)} \\ (G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)})H^{(1,0)} & (G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)})H^{(1,1)} \end{pmatrix}$$

$$\begin{pmatrix} H^{(1,1)} & -H^{(0,1)} \\ -H^{(1,0)} & H^{(0,0)} \end{pmatrix} = \begin{pmatrix} (H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)})G^{(0,0)} & (H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)})G^{(0,1)} \\ (H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)})G^{(1,0)} & (H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)})G^{(1,1)} \end{pmatrix}$$

Now since  $G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)}$  and  $H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)}$  also are circulant Toeplitz matrices, the expressions above give that

$$\begin{aligned} l(H^{(0,0)}) &\leq l(G^{(1,1)}) \leq l(H^{(0,0)}) \\ l(H^{(0,1)}) &\leq l(G^{(0,1)}) \leq l(H^{(0,1)}) \\ l(H^{(1,0)}) &\leq l(G^{(1,0)}) \leq l(H^{(1,0)}) \end{aligned}$$

so that we must have equality here, and with both

$$G^{(0,0)}G^{(1,1)} - G^{(1,0)}G^{(0,1)} \quad \text{and} \quad H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)}$$

having only one nonzero diagonal. In particular we can define the diagonal matrix  $D = H^{(0,0)}H^{(1,1)} - H^{(1,0)}H^{(0,1)} = \alpha^{-1}E_d$  (for some  $\alpha, d$ ), and we have that

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix} = \begin{pmatrix} \alpha E_{-d} H^{(1,1)} & -\alpha E_{-d} H^{(0,1)} \\ -\alpha E_{-d} H^{(1,0)} & \alpha E_{-d} H^{(0,0)} \end{pmatrix}.$$

The first columns here state a relation between  $G_0$  and  $H_1$ , while the second columns state a relation between  $G_1$  and  $H_0$ . It is straightforward to show that these relations imply equation (8.2)-(8.3). The details for this can be found in Exercise 8.1.  $\square$

In the following we will find factorizations of  $2 \times 2$ -block matrices where the blocks are filters, into simpler such matrices. The importance of Theorem 8.2 is then that MRA-matrices can be written as a product of simpler MRA matrices. These simpler MRA matrices will be called elementary lifting matrices, and will be of the following type.

**Definition 8.5.** *Elementary lifting matrices.*

A matrix on the form

$$\begin{pmatrix} I & S \\ 0 & I \end{pmatrix}$$

where  $S$  is a filter is called an *elementary lifting matrix of even type*. A matrix on the form

$$\begin{pmatrix} I & 0 \\ S & I \end{pmatrix}$$

is called an *elementary lifting matrix of odd type*.

The following are the most useful properties of elementary lifting matrices:

**Lemma 8.6.** *Lifting lemma.*

The following hold:

$$\begin{pmatrix} I & S \\ 0 & I \end{pmatrix}^T = \begin{pmatrix} I & 0 \\ S^T & I \end{pmatrix}, \text{ and } \begin{pmatrix} I & 0 \\ S & I \end{pmatrix}^T = \begin{pmatrix} I & S^T \\ 0 & I \end{pmatrix},$$

$$\begin{pmatrix} I & S_1 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & S_2 \\ 0 & I \end{pmatrix} = \begin{pmatrix} I & S_1 + S_2 \\ 0 & I \end{pmatrix}, \text{ and } \begin{pmatrix} I & 0 \\ S_1 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ S_2 & I \end{pmatrix} = \begin{pmatrix} I & 0 \\ S_1 + S_2 & I \end{pmatrix},$$

$$\begin{pmatrix} I & S \\ 0 & I \end{pmatrix}^{-1} = \begin{pmatrix} I & -S \\ 0 & I \end{pmatrix}, \text{ and } \begin{pmatrix} I & 0 \\ S & I \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ -S & I \end{pmatrix}$$

These statements follow directly from Theorem 8.3. Due to Property 2, one can assume that odd and even types of lifting matrices appear in alternating order, since matrices of the same type can be grouped together. The following result states why elementary lifting matrices can be used to factorize general MRA-matrices:

**Theorem 8.7.** *Multiplying.*

Any invertible matrix on the form  $S = \begin{pmatrix} S^{(0,0)} & S^{(0,1)} \\ S^{(1,0)} & S^{(1,1)} \end{pmatrix}$ , where the  $S^{(i,j)}$  are filters with a finite number of filter coefficients, can be written on the form

$$\Lambda_1 \cdots \Lambda_n \begin{pmatrix} \alpha_0 E_p & 0 \\ 0 & \alpha_1 E_q \end{pmatrix}, \quad (8.4)$$

where  $\Lambda_i$  are elementary lifting matrices,  $p, q$  are integers,  $\alpha_0, \alpha_1$  are nonzero scalars, and  $E_p, E_q$  are time delay filters. The inverse is given by

$$\begin{pmatrix} \alpha_0^{-1} E_{-p} & 0 \\ 0 & \alpha_1^{-1} E_{-q} \end{pmatrix} (\Lambda_n)^{-1} \cdots (\Lambda_1)^{-1}. \quad (8.5)$$

Note that  $(\Lambda_i)^{-1}$  can be computed with the help of Property 3 of Lemma 8.6.

*Proof.* The proof will use the concept of the length of a filter, as defined in Definition 3.3. Let  $S = \begin{pmatrix} S^{(0,0)} & S^{(0,1)} \\ S^{(1,0)} & S^{(1,1)} \end{pmatrix}$  be an arbitrary invertible matrix. We will incrementally find an elementary lifting matrix  $\Lambda_i$  with filter  $S_i$  in the lower left or upper right corner so that  $\Lambda_i S$  has filters of lower length in the first column. Assume first that  $l(S^{(0,0)}) \geq l(S^{(1,0)})$ , where  $l(S)$  is the length of a filter as given by Definition 3.3. If  $\Lambda_i$  is of even type, then the first column in  $\Lambda_i S$  is

$$\begin{pmatrix} I & S_i \\ 0 & I \end{pmatrix} \begin{pmatrix} S^{(0,0)} \\ S^{(1,0)} \end{pmatrix} = \begin{pmatrix} S^{(0,0)} + S_i S^{(1,0)} \\ S^{(1,0)} \end{pmatrix}. \quad (8.6)$$

$S_i$  can now be chosen so that  $l(S^{(0,0)} + S_i S^{(1,0)}) < l(S^{(1,0)})$ . To see how, recall that we in Section 3.1 stated that multiplying filters corresponds to multiplying polynomials.  $S_i$  can thus be found from polynomial division with remainder: when we divide  $S^{(0,0)}$  by  $S^{(1,0)}$ , we actually find polynomials  $S_i$  and  $P$  with  $l(P) < l(S^{(1,0)})$  so that  $S^{(0,0)} = S_i S^{(1,0)} + P$ , so that the length of  $P = S^{(0,0)} - S_i S^{(1,0)}$  is less than  $l(S^{(1,0)})$ . The same can be said if  $\Lambda_i$  is of odd type, in which case the first and second components are simply swapped. This procedure can be continued until we arrive at a product

$$\Lambda_n \cdots \Lambda_1 S$$

where either the first or the second component in the first column is 0. If the first component in the first column is 0, the identity

$$\begin{pmatrix} I & 0 \\ -I & I \end{pmatrix} \begin{pmatrix} I & I \\ 0 & I \end{pmatrix} \begin{pmatrix} 0 & X \\ Y & Z \end{pmatrix} = \begin{pmatrix} Y & X+Z \\ 0 & -X \end{pmatrix}$$

explains that we can bring the matrix to a form where the second element in the first column is zero instead, with the help of the additional lifting matrices

$$\Lambda_{n+1} = \begin{pmatrix} I & I \\ 0 & I \end{pmatrix} \text{ and } \Lambda_{n+2} = \begin{pmatrix} I & 0 \\ -I & I \end{pmatrix},$$

so that we always can assume that the second element in the first column is 0, i.e.

$$\Lambda_n \cdots \Lambda_1 S = \begin{pmatrix} P & Q \\ 0 & R \end{pmatrix},$$

for some matrices  $P, Q, R$ . From the proof of Theorem 6.16 we will see that in order for  $S$  to be invertible, we must have that  $S^{(0,0)} S^{(1,1)} - S^{(0,1)} S^{(1,0)} = -\alpha^{-1} E_d$  for some nonzero scalar  $\alpha$  and integer  $d$ . Since

$$\begin{pmatrix} P & Q \\ 0 & R \end{pmatrix}$$

is also invertible, we must thus have that  $PR$  must be on the form  $\alpha E_n$ . When the filters have a finite number of filter coefficients, the only possibility for this to happen is when  $P = \alpha_0 E_p$  and  $R = \alpha_1 E_q$  for some  $p, q, \alpha_0, \alpha_1$ . Using this, and also isolating  $S$  on one side, we obtain that

$$S = (\Lambda_1)^{-1} \cdots (\Lambda_n)^{-1} \begin{pmatrix} \alpha_0 E_p & Q \\ 0 & \alpha_1 E_q \end{pmatrix}, \quad (8.7)$$

Noting that

$$\begin{pmatrix} \alpha_0 E_p & Q \\ 0 & \alpha_1 E_q \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{\alpha_1} E_{-q} Q \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_0 E_p & 0 \\ 0 & \alpha_1 E_q \end{pmatrix},$$

we can rewrite Equation (8.7) as

$$S = (\Lambda_1)^{-1} \cdots (\Lambda_n)^{-1} \begin{pmatrix} 1 & \frac{1}{\alpha_1} E_{-q} Q \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_0 E_p & 0 \\ 0 & \alpha_1 E_q \end{pmatrix},$$

which is a lifting factorization of the form we wanted to arrive at. The last matrix in the lifting factorization is not really a lifting matrix, but it too can easily be inverted, so that we arrive at Equation (8.5). This completes the proof.  $\square$

Factorizations on the form given by Equation (8.4) will be called *lifting factorizations*. Assume that we have applied Theorem 8.7 in order to get a factorization of the polyphase representation of the DWT kernel of the form

$$\Lambda_n \cdots \Lambda_2 \Lambda_1 H = \begin{pmatrix} \alpha & \mathbf{0} \\ \mathbf{0} & \beta \end{pmatrix}. \quad (8.8)$$

Theorem 8.6 then immediately gives us the following factorizations.

$$H = (\Lambda_1)^{-1} (\Lambda_2)^{-1} \cdots (\Lambda_n)^{-1} \begin{pmatrix} \alpha & \mathbf{0} \\ \mathbf{0} & \beta \end{pmatrix} \quad (8.9)$$

$$G = \begin{pmatrix} 1/\alpha & \mathbf{0} \\ \mathbf{0} & 1/\beta \end{pmatrix} \Lambda_n \cdots \Lambda_2 \Lambda_1 \quad (8.10)$$

$$H^T = \begin{pmatrix} \alpha & \mathbf{0} \\ \mathbf{0} & \beta \end{pmatrix} ((\Lambda_n)^{-1})^T ((\Lambda_{n-1})^{-1})^T \cdots ((\Lambda_1)^{-1})^T \quad (8.11)$$

$$G^T = (\Lambda_1)^T (\Lambda_2)^T \cdots (\Lambda_n)^T \begin{pmatrix} 1/\alpha & \mathbf{0} \\ \mathbf{0} & 1/\beta \end{pmatrix}. \quad (8.12)$$

Since  $H^T$  and  $G^T$  are the kernel transformations of the dual IDWT and the dual DWT, respectively, these formulas give us recipes for computing the DWT, IDWT, dual IDWT, and the dual DWT, respectively. All in all, everything can be computed by combining elementary lifting steps.

In practice, one starts with a given wavelet with certain proved properties such as the ones from Chapter 7, and applies an algorithm to obtain a lifting factorization of the polyphase representation of the kernels. The algorithm can easily be written down from the proof of Theorem 8.7. The lifting factorization is far from unique, and the algorithm only gives one of them.

It is desirable for an implementation to obtain a lifting factorization where the lifting steps are as simple as possible. Let us restrict to the case of wavelets with symmetric filters, since the wavelets used in most applications are symmetric. In particular this means that  $S^{(0,0)}$  is a symmetric matrix, and that  $S^{(1,0)}$  is symmetric about  $-1/2$  (see Exercise 8.8).

Assume that we in the proof of Theorem 8.7 add an elementary lifting of even type. At this step we then compute  $S^{(0,0)} + S_i S^{(1,0)}$  in the first entry of the first column. Since  $S^{(0,0)}$  is now assumed symmetric,  $S_i S^{(1,0)}$  must also be symmetric in order for the length to be reduced. And since the filter coefficients of  $S^{(1,0)}$  are assumed symmetric about  $-1/2$ ,  $S_i$  must be chosen with symmetry around  $1/2$ .

For most of our wavelets we will consider in the following examples it will turn out the filters in the first column differ in the number of filter coefficients by 1 at all steps. When this is the case, we can choose a filter of length 2 to reduce the length by 2, so that the  $S_i$  in an even lifting step can be chosen on the form  $S_i = \lambda_i\{\underline{1}, 1\}$ . Similarly, for an odd lifting step,  $S_i$  can be chosen on the form  $S_i = \lambda_i\{1, \underline{1}\}$ . Let us summarize this as follows:

**Theorem 8.8.** *Differing by 1.*

When the filters in a wavelet are symmetric and the lengths of the filters in the first column differ by 1 at all steps in the lifting factorization, the lifting steps of even and odd type take the simplified form

$$\begin{pmatrix} I & \lambda_i\{\underline{1}, 1\} \\ 0 & I \end{pmatrix} \text{ and } \begin{pmatrix} I & 0 \\ \lambda_i\{1, \underline{1}\} & I \end{pmatrix},$$

respectively.

The lifting steps mentioned in this theorem are quickly computed due to their simple structure.

Each lifting step leaves every second element unchanged, while for the remaining elements, we simply add the two neighbours. Clearly these computations can be computed in-place, without the need for extra memory allocations. From this it is also clear how we can compute the entire DWT/IDWT in-place. We simply avoid the reorganizing into the  $(\phi_{m-1}, \psi_{m-1})$ -basis until after all the lifting steps. After the application of the matrices above, we have coordinates in the  $\mathcal{C}_m$ -basis. Here only the coordinates with indices  $(0, 2, 4, \dots)$  need to be further transformed, so the next step in the algorithm should work directly on these. After the next step only the coordinates with indices  $(0, 4, 8, \dots)$  need to be further transformed, and so on. From this it is clear that

- the  $\psi_{m-k}$  coordinates are found at indices  $2^{k-1} + r2^k$ , i.e. the last  $k$  bits are 1 followed by  $k - 1$  zeros.
- the  $\phi_0$  coordinates are found at indices  $r2^m$ , i.e. the last  $m$  bits are 0.

If we place the last  $k$  bits of the  $\psi_{m-k}$ -coordinates in front in reverse order, and the the last  $m$  bits of the  $\phi_0$ -coordinates in front, the coordinates have the same order as in the  $(\phi_{m-1}, \psi_{m-1})$ -basis. This is also called a partial bit-reverse, and is related to the bit-reversal performed in the FFT algorithm.

Clearly, these lifting steps are also MRA-matrices with symmetric filters, so that our procedure factorizes an MRA-matrix with symmetric filters into simpler MRA-matrices which also have symmetric filters.

### 8.1.1 Reduction in the number of arithmetic operations

The number of arithmetic operations needed to apply matrices on the form stated in Equation (6.10) is easily computed. The number of multiplications is  $N/2$  if symmetry is exploited as in Observation 4.20 ( $N$  if symmetry is not

exploited). Similarly, the number of additions is  $N$ . Let  $K$  be the total number of filter coefficients in  $H_0, H_1$ . In the following we will see that each lifting step can be chosen to reduce the number of filter coefficients in the MRA matrix by 4, so that a total number of  $K/4$  lifting steps are required. Thus, a total number of  $KN/8$  ( $KN/4$ ) multiplications, and  $KN/4$  additions are required when a lifting factorization is used. In comparison, a direct implementation would require  $KN/4$  ( $KN/2$ ) multiplications, and  $KN/2$  additions. For the examples we will consider, we therefore have the following result.

**Theorem 8.9.** *Reducing arithmetic operations.*

The lifting factorization approximately halves the number of additions and multiplications needed, when compared with a direct implementation (regardless of whether symmetry is exploited or not).

### Exercise 8.1: The frequency responses of the polyphase components

Let  $H$  and  $G$  be MRA-matrices for a DWT/IDWT, with corresponding filters  $H_0, H_1, G_0, G_1$ , and polyphase components  $H^{(i,j)}, G^{(i,j)}$ .

a) Show that

$$\begin{aligned}\lambda_{H_0}(\omega) &= \lambda_{H^{(0,0)}}(2\omega) + e^{i\omega} \lambda_{H^{(0,1)}}(2\omega) \\ \lambda_{H_1}(\omega) &= \lambda_{H^{(1,1)}}(2\omega) + e^{-i\omega} \lambda_{H^{(1,0)}}(2\omega) \\ \lambda_{G_0}(\omega) &= \lambda_{G^{(0,0)}}(2\omega) + e^{-i\omega} \lambda_{G^{(1,0)}}(2\omega) \\ \lambda_{G_1}(\omega) &= \lambda_{G^{(1,1)}}(2\omega) + e^{i\omega} \lambda_{G^{(0,1)}}(2\omega).\end{aligned}$$

b) In the proof of the last part of Theorem 6.16, we deferred the last part, namely that equations (8.2)-(8.3) follow from

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix} = \begin{pmatrix} \alpha E_{-d} H^{(1,1)} & -\alpha E_{-d} H^{(0,1)} \\ -\alpha E_{-d} H^{(1,0)} & \alpha E_{-d} H^{(0,0)} \end{pmatrix}.$$

Prove this based on the result from a).

### Exercise 8.2: Finding new filters

Let  $S$  be a filter. Show that

a)

$$G \begin{pmatrix} I & \mathbf{0} \\ S & I \end{pmatrix}$$

is an MRA matrix with filters  $\tilde{G}_0, G_1$ , where

$$\lambda_{\tilde{G}_0}(\omega) = \lambda_{G_0}(\omega) + \lambda_S(2\omega) e^{-i\omega} \lambda_{G_1}(\omega),$$

b)

$$G \begin{pmatrix} I & S \\ \mathbf{0} & I \end{pmatrix}$$

is an MRA matrix with filters  $G_0, \tilde{G}_1$ , where

$$\lambda_{\tilde{G}_1}(\omega) = \lambda_{G_1}(\omega) + \lambda_S(2\omega)e^{i\omega} \lambda_{G_0}(\omega),$$

c)

$$\begin{pmatrix} I & \mathbf{0} \\ S & I \end{pmatrix} H$$

is an MRA-matrix with filters  $H_0, \tilde{H}_1$ , where

$$\lambda_{\tilde{H}_1}(\omega) = \lambda_{H_1}(\omega) + \lambda_S(2\omega)e^{-i\omega} \lambda_{H_0}(\omega).$$

d)

$$\begin{pmatrix} I & S \\ \mathbf{0} & I \end{pmatrix} H$$

is an MRA-matrix with filters  $\tilde{H}_0, H_1$ , where

$$\lambda_{\tilde{H}_0}(\omega) = \lambda_{H_0}(\omega) + \lambda_S(2\omega)e^{i\omega} \lambda_{H_1}(\omega).$$

In summary, this exercise shows that one can think of the steps in the lifting factorization as altering one of the filters of an MRA-matrix in alternating order.

### Exercise 8.3: Relating to the polyphase components

Show that  $S$  is a filter of length  $kM$  if and only if the entries  $\{S^{i,j}\}_{i,j=0}^{M-1}$  in the polyphase representation of  $S$  satisfy  $S^{(i+r) \bmod M, (j+r) \bmod M} = S_{i,j}$ . In other words,  $S$  is a filter if and only if the polyphase representation of  $S$  is a “block-circulant Toeplitz matrix”. This implies a fact that we will use:  $GH$  is a filter (and thus provides alias cancellation) if blocks in the polyphase representations repeat cyclically as in a Toeplitz matrix (in particular when the matrix is block-diagonal with the same block repeating on the diagonal).

### Exercise 8.4: QMF filter banks

Recall from Definition 6.18 that we defined a classical QMF filter bank as one where  $M = 2$ ,  $G_0 = H_0$ ,  $G_1 = H_1$ , and  $\lambda_{H_1}(\omega) = \lambda_{H_0}(\omega + \pi)$ . Show that the forward and reverse filter bank transforms of a classical QMF filter bank take the form

$$H = G = \begin{pmatrix} A & -B \\ B & A \end{pmatrix}$$

**Exercise 8.5: Alternative QMF filter banks**

Recall from Definition 6.19 that we defined an alternative QMF filter bank as one where  $M = 2$ ,  $G_0 = (H_0)^T$ ,  $G_1 = (H_1)^T$ , and  $\lambda_{H_1}(\omega) = \overline{\lambda_{H_0}(\omega + \pi)}$ . Show that the forward and reverse filter bank transforms of an alternative QMF filter bank take the form.

$$H = \begin{pmatrix} A^T & B^T \\ -B & A \end{pmatrix} \quad G = \begin{pmatrix} A & -B^T \\ B & A^T \end{pmatrix} = \begin{pmatrix} A^T & B^T \\ -B & A \end{pmatrix}^T.$$

**Exercise 8.6: Alternative QMF filter banks with additional sign**

Consider alternative QMF filter banks where we take in an additional sign, so that  $\lambda_{H_1}(\omega) = -\overline{\lambda_{H_0}(\omega + \pi)}$  (the Haar wavelet was an example of such a filter bank). Show that the forward and reverse filter bank transforms now take the form

$$H = \begin{pmatrix} A^T & B^T \\ B & -A \end{pmatrix} \quad G = \begin{pmatrix} A & B^T \\ B & -A^T \end{pmatrix} = \begin{pmatrix} A^T & B^T \\ B & -A \end{pmatrix}^T.$$

It is straightforward to check that also these satisfy the alias cancellation condition, and that the perfect reconstruction condition also here takes the form  $|\lambda_{H_0}(\omega)|^2 + |\lambda_{H_0}(\omega + \pi)|^2 = 2$ .

**8.2 Examples of lifting factorizations**

We have seen that the polyphase representations of wavelet kernels can be factored into a product of elementary lifting matrices. In this section we will compute the exact factorizations for the wavelets we have considered. In the exercises we will then complete the implementations, so that we can make actual experiments, such as listening to the low-resolution approximations in sound, or using the cascade algorithm to plot scaling functions and mother wavelets. We will omit the Haar wavelet. One can easily write down a lifting factorization for this as well, but there is little to save in this factorization when compared to the direct form of this we already have considered.

First we will consider the two piecewise linear wavelets we have looked at. It turns out that their lifting factorizations can be obtained in a direct way by considering the polyphase representations as a change of coordinates. To see how, we first define

$$\mathcal{D}_m = \{\phi_{m,0}, \phi_{m,2}, \phi_{m,4}, \dots, \phi_{m,1}, \phi_{m,3}, \phi_{m,5}, \dots\}, \quad (8.13)$$

$P_{\mathcal{D}_m \leftarrow \phi_m}$  is clearly the permutation matrix  $P$  used in the similarity between a matrix and its polyphase representation. Let now  $H$  and  $G$  be the kernel transformations of a wavelet. The polyphase representation of  $H$  is



$$PHP^T = P_{\mathcal{D}_m \leftarrow \phi_m} P_{\mathcal{C}_m \leftarrow \phi_m} P_{\phi_m \leftarrow \mathcal{D}_m} = P_{(\phi_1, \psi_1) \leftarrow \phi_m} P_{\phi_m \leftarrow \mathcal{D}_m} = P_{(\phi_1, \psi_1) \leftarrow \mathcal{D}_m}.$$

Taking inverses here we obtain that  $PGP^T = P_{\mathcal{D}_m \leftarrow (\phi_1, \psi_1)}$ . We therefore have the following result:

**Theorem 8.10.** *The polyphase representation.*

The polyphase representation of  $H$  equals the change of coordinates matrix  $P_{(\phi_1, \psi_1) \leftarrow \mathcal{D}_m}$ , and the polyphase representation of  $G$  equals the change of coordinates matrix  $P_{\mathcal{D}_m \leftarrow (\phi_1, \psi_1)}$ .

### 8.2.1 The piecewise linear wavelet

The polyphase representation of  $G$  is  $\frac{1}{\sqrt{2}} \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix}$ . Due to Theorem 8.6, the polyphase representation of  $H$  is  $\sqrt{2} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix}$ . We can summarize that the polyphase representations of the kernels  $H$  and  $G$  for the piecewise linear wavelet are

$$\sqrt{2} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix} \text{ and } \frac{1}{\sqrt{2}} \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix}, \quad (8.14)$$

respectively.

### Example 8.7: Lifting factorization of the alternative piecewise linear wavelet

The polyphase representation of  $H$  is

$$\sqrt{2} \begin{pmatrix} I & \frac{1}{4}\{1, 1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix}.$$

In this case we required one additional lifting step. We can thus conclude that the polyphase representations of the kernels  $H$  and  $G$  for the alternative piecewise linear wavelet are

$$\sqrt{2} \begin{pmatrix} I & \frac{1}{4}\{1, 1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix} \text{ and } \frac{1}{\sqrt{2}} \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{2}\{1, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & -\frac{1}{4}\{1, 1\} \\ \mathbf{0} & I \end{pmatrix}, \quad (8.15)$$

respectively.

### 8.2.2 The Spline 5/3 wavelet

Let us consider the Spline 5/3 wavelet, which we defined in Example 7.4.1. Let us start by looking at, and we recall that

$$H_0 = \left\{ -\frac{1}{4}, \frac{1}{2}, \frac{3}{2}, \frac{1}{2}, -\frac{1}{4} \right\} \quad H_1 = \left\{ -\frac{1}{4}, \frac{1}{2}, -\frac{1}{4} \right\}.$$

from which we see that the polyphase components of  $H$  are

$$\begin{pmatrix} H^{(0,0)} & H^{(0,1)} \\ H^{(1,0)} & H^{(1,1)} \end{pmatrix} = \begin{pmatrix} \{-\frac{1}{4}, \frac{3}{2}, -\frac{1}{4}\} & \frac{1}{2}\{1, 1\} \\ -\frac{1}{4}\{1, 1\} & \frac{1}{2}I \end{pmatrix}$$

We see here that the upper filter has most filter coefficients in the first column, so that we must start with an elementary lifting of even type. We need to find a filter  $S_1$  so that  $S_1\{-1/4, -1/4\} + \{-1/4, 3/2, -1/4\}$  has fewer filter coefficients than  $\{-1/4, 3/2, -1/4\}$ . It is clear that we can choose  $S_1 = \{-1, -1\}$ , and that

$$\Lambda_1 H = \begin{pmatrix} I & \{-1, -1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} \{-\frac{1}{4}, \frac{3}{2}, -\frac{1}{4}\} & \frac{1}{2}\{1, 1\} \\ -\frac{1}{4}\{1, 1\} & \frac{1}{2}I \end{pmatrix} = \begin{pmatrix} 2I & \mathbf{0} \\ -\frac{1}{4}\{1, 1\} & \frac{1}{2}I \end{pmatrix}$$

Now we need to apply an elementary lifting of odd type, and we need to find a filter  $S_2$  so that  $S_2 I - \frac{1}{4}\{1, 1\} = \mathbf{0}$ . Clearly we can choose  $S_2 = \{1/8, 1/8\}$ , and we get

$$\Lambda_2 \Lambda_1 H = \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{8}\{1, 1\} & I \end{pmatrix} \begin{pmatrix} 2I & \mathbf{0} \\ -\frac{1}{4}\{1, 1\} & \frac{1}{2}I \end{pmatrix} = \begin{pmatrix} 2I & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}I \end{pmatrix}.$$

Multiplying with inverses of elementary lifting steps, we now obtain that the polyphase representations of the kernels for the Spline 5/3 wavelet are

$$H = \begin{pmatrix} I & \{1, 1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{8}\{1, 1\} & I \end{pmatrix} \begin{pmatrix} 2I & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}I \end{pmatrix}$$

and

$$G = \begin{pmatrix} \frac{1}{2}I & \mathbf{0} \\ \mathbf{0} & 2I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{8}\{1, 1\} & I \end{pmatrix} \begin{pmatrix} I & \{-1, -1\} \\ \mathbf{0} & I \end{pmatrix},$$

respectively. Two lifting steps are thus required. We also see that the lifting steps involve only dyadic fractions, just as the filter coefficients did. This means that the lifting factorization also can be used for lossless operations.

### 8.2.3 The CDF 9/7 wavelet

For the wavelet we considered in Example 7.6, it is more cumbersome to compute the lifting factorization by hand. It is however, straightforward to write an algorithm which computes the lifting steps, as these are performed in the proof of Theorem 8.7. You will be spared the details of this algorithm. Also, when we

use these wavelets in implementations later they will use precomputed values of these lifting steps, and you can take these implementations for granted too. If we run the algorithm for computing the lifting factorization we obtain that the polyphase representations of the kernels  $H$  and  $G$  for the CDF 9/7 wavelet are

$$\begin{aligned} & \begin{pmatrix} I & 0.5861\{\underline{1}, 1\} \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0.6681\{\underline{1}, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & -0.0700\{\underline{1}, 1\} \\ 0 & I \end{pmatrix} \\ & \times \begin{pmatrix} I & 0 \\ -1.2002\{\underline{1}, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} -1.1496 & 0 \\ 0 & -0.8699 \end{pmatrix} \text{ and} \\ & \begin{pmatrix} -0.8699 & 0 \\ 0 & -1.1496 \end{pmatrix} \begin{pmatrix} I & 0 \\ 1.2002\{\underline{1}, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & 0.0700\{\underline{1}, 1\} \\ 0 & I \end{pmatrix} \\ & \times \begin{pmatrix} I & 0 \\ -0.6681\{\underline{1}, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & -0.5861\{\underline{1}, 1\} \\ 0 & I \end{pmatrix}, \end{aligned}$$

respectively. In this case four lifting steps were required.

Perhaps more important than the reduction in the number of arithmetic operations is the fact that the lifting factorization splits the DWT and IDWT into simpler components, each very attractive for hardware implementations since a lifting step only requires the additional value  $\lambda_i$  from Theorem 8.8. Lifting actually provides us with a complete implementation strategy for the DWT and IDWT, in which the  $\lambda_i$  are used as precomputed values.

Finally we will find a lifting factorization for orthonormal wavelets. Note that here the filters  $H_0$  and  $H_1$  are not symmetric, and each of them has an even number of filter coefficients. There are thus a different number of filter coefficients with positive and negative indices, and in Section 7.6 we defined the filters so that the filter coefficients were as symmetric as possible when it came to the number of nonzero filter coefficients with positive and negative indices.

### 8.2.4 Orthonormal wavelets

We will attempt to construct a lifting factorization where the following property is preserved after each lifting step:

P1:  $H^{(0,0)}$ ,  $H^{(1,0)}$  have a minimum possible overweight of filter coefficients with negative indices.

This property stems from the assumption in Section 7.6 that  $H_0$  is assumed to have a minimum possible overweight of filter coefficients with negative indices. To see that this holds at the start, assume as before that all the filters have  $2L$  nonzero filter coefficients, so that  $H_0$  and  $H_1$  are on the form given by Equation (7.30). Assume first that  $L$  is even. It is clear that

$$\begin{aligned}
 H^{(0,0)} &= \{t_{-L}, \dots, t_{-2}, \underline{t_0}, t_2, \dots, t_{L-2}\} \\
 H^{(0,1)} &= \{t_{-L+1}, \dots, t_{-3}, \underline{t_{-1}}, t_1, \dots, t_{L-1}\} \\
 H^{(1,0)} &= \{s_{-L+1}, \dots, s_{-1}, \underline{s_1}, s_3, \dots, s_{L-1}\} \\
 H^{(1,1)} &= \{s_{-L+2}, \dots, s_{-2}, \underline{s_0}, s_2, \dots, s_L\}.
 \end{aligned}$$

Clearly P1 holds. Assume now that  $L$  is odd. It is now clear that

$$\begin{aligned}
 H^{(0,0)} &= \{t_{-L+1}, \dots, t_{-2}, \underline{t_0}, t_2, \dots, t_{L-1}\} \\
 H^{(0,1)} &= \{t_{-L}, \dots, t_{-3}, \underline{t_{-1}}, t_1, \dots, t_{L-2}\} \\
 H^{(1,0)} &= \{s_{-L+2}, \dots, s_{-1}, \underline{s_1}, s_3, \dots, s_L\} \\
 H^{(1,1)} &= \{s_{-L+1}, \dots, s_{-2}, \underline{s_0}, s_2, \dots, s_{L-1}\}.
 \end{aligned}$$

In this case it is seen that all filters have equally many filter coefficients with positive and negative indices, so that P1 holds also here.

Now let us turn to the first lifting step. We will choose it so that the number of filter coefficients in the first column is reduced with 1, and so that  $H^{(0,0)}$  has an odd number of coefficients. If  $L$  is even, we saw that  $H^{(0,0)}$  and  $H^{(1,0)}$  had an even number of coefficients, so that the first lifting step must be even. To preserve P1, we must cancel  $t_{-L}$ , so that the first lifting step is

$$\Lambda_1 = \begin{pmatrix} I & -t_{-L}/s_{-L+1} \\ \mathbf{0} & I \end{pmatrix}.$$

If  $L$  is odd, we saw that  $H^{(0,0)}$  and  $H^{(1,0)}$  had an odd number of coefficients, so that the first lifting step must be odd. To preserve P1, we must cancel  $s_L$ , so that the first lifting step is

$$\Lambda_1 = \begin{pmatrix} I & \mathbf{0} \\ -s_L/t_{L-1} & I \end{pmatrix}.$$

Now that we have a difference of one filter coefficient in the first column, we will reduce the entry with the most filter coefficients with two with a lifting step, until we have  $H^{(0,0)} = \{\underline{K}\}$ ,  $H^{(1,0)} = 0$  in the first column.

Assume first that  $H^{(0,0)}$  has the most filter coefficients. We then need to apply an even lifting step. Before an even step, the first column has the form

$$\begin{pmatrix} \{t_{-k}, \dots, t_{-1}, \underline{t_0}, t_1, \dots, t_k\} \\ \{s_{-k}, \dots, s_{-1}, s_0, s_1, \dots, s_{k-1}\} \end{pmatrix}.$$

We can then choose  $\Lambda_i = \begin{pmatrix} I & \{-t_{-k}/s_{-k}, -t_k/s_{k-1}\} \\ \mathbf{0} & I \end{pmatrix}$  as a lifting step.

Assume then that  $H^{(1,0)}$  has the most filter coefficients. We then need to apply an odd lifting step. Before an odd step, the first column has the form

$$\begin{pmatrix} \{t_{-k}, \dots, t_{-1}, t_0, t_1, \dots, t_k\} \\ \{s_{-k-1}, \dots, s_{-1}, s_0, s_1, \dots, s_k\} \end{pmatrix}.$$

We can then choose  $\Lambda_i = \begin{pmatrix} I & \mathbf{0} \\ \{-s_{-k-1}/t_{-k}, -s_k/t_k\} & I \end{pmatrix}$  as a lifting step.

If  $L$  is even we end up with a matrix on the form  $\begin{pmatrix} \alpha & \{0, K\} \\ \mathbf{0} & \beta \end{pmatrix}$ , and we can choose the final lifting step as  $\Lambda_n = \begin{pmatrix} I & \{0, -K/\beta\} \\ \mathbf{0} & I \end{pmatrix}$ .

If  $L$  is odd we end up with a matrix on the form

$$\begin{pmatrix} \alpha & K \\ \mathbf{0} & \beta \end{pmatrix},$$

and we can choose the final lifting step as  $\Lambda_n = \begin{pmatrix} I & -K/\beta \\ \mathbf{0} & I \end{pmatrix}$ . Again using equations (8.9)-(8.10), this gives us the lifting factorizations.

In summary we see that all even and odd lifting steps take the form  $\begin{pmatrix} I & \{\lambda_1, \lambda_2\} \\ \mathbf{0} & I \end{pmatrix}$  and  $\begin{pmatrix} I & \mathbf{0} \\ \{\lambda_1, \lambda_2\} & I \end{pmatrix}$ . We see that symmetric lifting steps correspond to the special case when  $\lambda_1 = \lambda_2$ . The even and odd lifting matrices now used are

$$\begin{pmatrix} 1 & \lambda_1 & 0 & 0 & \cdots & 0 & 0 & \lambda_2 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda_2 & 1 & \lambda_1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda_2 & 1 & \lambda_1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda_2 & 1 & \lambda_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \lambda_1 & 0 & 0 & 0 & \cdots & 0 & \lambda_2 & 1 \end{pmatrix}, \tag{8.16}$$

respectively. We note that when we reduce elements to the left and right in the upper and lower part of the first column, the same type of reductions must occur in the second column, since the determinant  $H^{(0,0)}H^{(1,1)} - H(0,1)H^{(1,0)}$  is a constant after any number of lifting steps.

This example explains the procedure for finding the lifting factorization into steps of the form given in Equation (8.16). You will be spared the details of writing an implementation which applies this procedure. In order to use orthonormal wavelets in implementations, we have implemented a function `liftingfactortho`, which takes  $N$  as input, and computes the steps in a lifting factorization so that (8.8) holds. These are written to file, and read from file when needed (you need not call `liftingfactortho` yourself, this is handled behind the curtains). In the exercises, you will be asked to implement both these non-symmetric elementary lifting steps, as well as the full kernel transformations for orthonormal wavelets.

**Exercise 8.8: Polyphase components for symmetric filters**

Assume that the filters  $H_0, H_1$  of a wavelet are symmetric, and denote by  $S^{(i,j)}$  the polyphase components of the corresponding MRA-matrix  $H$ . Show that  $S^{(0,0)}$  and  $S^{(1,1)}$  are symmetric filters, that the filter coefficients of  $S^{(1,0)}$  has symmetry about  $-1/2$ , and that  $S^{(0,1)}$  has symmetry about  $1/2$ . Also show a similar statement for the MRA-matrix  $G$  of the inverse DWT.

**Exercise 8.9: Implementing kernels transformations using lifting**

Up to now in this chapter we have obtained lifting factorizations for four different wavelets where the filters are symmetric. Let us now implement the kernel transformations for these wavelets. Your functions should call the functions from Exercise 5.33 and Exercise 5.37. in order to compute the individual lifting steps. Recall that the kernel transformations should take the input vector  $\mathbf{x}$ , `symm` (i.e. whether symmetric extension should be applied), and `dual` (i.e. whether the dual wavelet transform should be applied) as input. You will need equations (8.9)-(8.12) here, in order to complete the kernels for bot the transformations and the dual transformations.

a) Write functions

```
dwt_kernel_53(x, bd_mode)
idwt_kernel_53(x, bd_mode)
```

which implement the DWT and IDWT kernel transformations for the Spline 5/3 wavelet. Use the lifting factorization obtained in Example 8.2.2.

b) Write functions

```
dwt_kernel_97(x, bd_mode)
idwt_kernel_97(x, bd_mode)
```

which implement the DWT and IDWT kernel transformations for the CDF 9/7 wavelet. Use the lifting factorization obtained in Example 8.2.3.

c) In Chapter 5, we listened to the low-resolution approximations and detail components in sound for three different wavelets. Repeat these experiments with the Spline 5/3 and the CDF 9/7 wavelet, using the new kernels we have implemented in this exercise.

d) Plot all scaling functions and mother wavelets for the Spline 5/3 and the CDF 9/7 wavelets, using the cascade algorithm and the kernels you have implemented.

**Exercise 8.10: Lifting orthonormal wavelets**

In this exercise we will implement the kernel transformations for orthonormal wavelets.

a) Write functions

```
lifting_even(lambda1, lambda2, x, bd_mode)
lifting_odd(lambda1, lambda2, x, bd_mode)
```

which apply the elementary lifting matrices (8.16) to  $\mathbf{x}$ . Assume that  $N$  is even.

b) Write functions

```
dwt_kernel_ortho(x, filters, bd_mode)
idwt_kernel_ortho(x, filters, bd_mode)
```

which apply the DWT and IDWT kernel transformations for orthonormal wavelets to  $\mathbf{x}$ . You should call the functions `lifting_even` and `lifting_odd`. You can assume that you can access the lifting steps so that the lifting factorization (8.8) holds, through the object `filters` by writing `filters.lambdas`, `filters.alpha`, and `filters.beta`. `filters.lambdas` is an  $n \times 2$ -matrix so that the filter coefficients  $\{\lambda_1, \lambda_2\}$  or  $\{\lambda_1, \lambda_2\}$  in the  $i$ 'th lifting step is found in row  $i$ . Recall that the last lifting step was even.

Due to the `filters` object, the functions `dwt_kernel_ortho` and `idwt_kernel_ortho` do not abide to the signature we have required for kernel functions up to now. The code base creates such functions based on the functions above in the following way:

```
filters = ...
dwt_kernel = lambda x, bd_mode: dwt_kernel_ortho(x, filters, bd_mode)
```

c) Listen to the low-resolution approximations and detail components in sound for orthonormal wavelets for  $N = 1, 2, 3, 4$ .

d) Plot all scaling functions and mother wavelets for the orthonormal wavelets for  $N = 1, 2, 3, 4$ , using the cascade algorithm. Since the wavelets are orthonormal, we should have that  $\phi = \tilde{\phi}$ , and  $\psi = \tilde{\psi}$ . In other words, you should see that the bottom plots equal the upper plots.

### Exercise 8.11: 4 vanishing moments

In Exercise 5.39 we found constants  $\alpha, \beta, \gamma, \delta$  which give the coordinates of  $\hat{\psi}$  in  $(\phi_1, \hat{\psi}_1)$ , where  $\hat{\psi}$  had four vanishing moments, and where we worked with the multiresolution analysis of piecewise constant functions.

a) Show that the polyphase representation of  $G$  when  $\hat{\psi}$  is used as mother wavelet can be factored as

$$\frac{1}{\sqrt{2}} \begin{pmatrix} I & \mathbf{0} \\ \{1/2, 1/2\} & I \end{pmatrix} \begin{pmatrix} I & \{-\gamma, -\alpha, -\beta, -\delta\} \\ \mathbf{0} & I \end{pmatrix}. \quad (8.17)$$

You here need to reconstruct what you did in the lifting factorization for the alternative piecewise linear wavelet, i.e. write

$$P_{\mathcal{D}_1 \leftarrow (\phi_1, \hat{\psi}_1)} = P_{\mathcal{D}_1 \leftarrow (\phi_1, \psi_1)} P_{(\phi_1, \psi_1) \leftarrow (\phi_1, \hat{\psi}_1)}.$$

By inversion, find also a lifting factorization of  $H$ .

### Exercise 8.12: Wavelet based on piecewise quadratic scaling function

In Exercise 7.3 you should have found the filters

$$\begin{aligned} H_0 &= \frac{1}{128} \{-5, 20, -1, -96, 70, \underline{280}, 70, -96, -1, 20, -5\} \\ H_1 &= \frac{1}{16} \{1, -4, \underline{6}, -4, 1\} \\ G_0 &= \frac{1}{16} \{1, 4, \underline{6}, 4, 1\} \\ G_1 &= \frac{1}{128} \{5, 20, 1, -96, -70, \underline{280}, -70, -96, 1, 20, 5\}. \end{aligned}$$

a) Show that

$$\begin{pmatrix} I & -\frac{1}{128} \{5, -29, -29, 5\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -\{1, 1\} & I \end{pmatrix} \begin{pmatrix} I & -\frac{1}{4} \{1, 1\} \\ \mathbf{0} & I \end{pmatrix} G = \begin{pmatrix} \frac{1}{4} & \mathbf{0} \\ \mathbf{0} & 4 \end{pmatrix}.$$

From this we can easily derive the lifting factorization of  $G$ .

- b) Listen to the low-resolution approximations and detail components in sound for this wavelet.
- c) Plot all scaling functions and mother wavelets for this wavelet, using the cascade algorithm.

## 8.3 Cosine-modulated filter banks and the MP3 standard

Previously we saw that the MP3 standard used a certain filter bank, called a cosine-modulated filter bank. We also illustrated that, surprisingly for a much used international standard, the synthesis system did not exactly invert the analysis system, i.e. we do not have perfect reconstruction, only “near-perfect reconstruction”. In this section we will first explain how this filter bank can be constructed, and why it can not give perfect reconstruction. In particular it will be clear how the prototype filter can be constructed. We will then construct a very similar filter bank, which actually can give perfect reconstruction. It may seem very surprising that the MP3 standard does not use this filter bank instead due to this. The explanation may lie in that the MP3 standard was established at about the same time as these filter banks were developed, so that the standard did not capture this very similar filter bank with perfect reconstruction.



### 8.3.1 Polyphase representations of the filter bank transforms

The main idea is to find the polyphase representations of the forward and reverse filter bank transforms of the MP3 standard. We start with the expression

$$z_{32(s-1)+n} = \sum_{k=0}^{511} \cos((n+1/2)(k-16)\pi/32) h_k x_{32s-k-1}, \quad (8.18)$$

which lead to the expression of the forward filter bank transform (Theorem 6.24). Using that any  $k < 512$  can be written uniquely on the form  $k = m + 64r$ , where  $0 \leq m < 64$ , and  $0 \leq r < 8$ , we can rewrite this as

$$\begin{aligned} &= \sum_{m=0}^{63} \sum_{r=0}^7 (-1)^r \cos(2\pi(n+1/2)(m-16)/64) h_{m+64r} x_{32s-(m+64r)-1} \\ &= \sum_{m=0}^{63} \cos(2\pi(n+1/2)(m-16)/64) \sum_{r=0}^7 (-1)^r h_{m+32 \cdot 2r} x_{32(s-2r)-m-1}. \end{aligned}$$

Here we also used Property (6.35). If we write

$$V^{(m)} = \{(-1)^0 h_m, 0, (-1)^1 h_{m+64}, 0, (-1)^2 h_{m+128}, \dots, (-1)^7 h_{m+7 \cdot 64}, 0\}, \quad (8.19)$$

for  $0 \leq m \leq 63$ , and we can write the expression above as

$$\begin{aligned} &\sum_{m=0}^{63} \cos(2\pi(n+1/2)(m-16)/64) \sum_{r=0}^{15} V_r^{(m)} x_{32(s-r)-m-1} \\ &= \sum_{m=0}^{63} \cos(2\pi(n+1/2)(m-16)/64) \sum_{r=0}^{15} V_r^{(m)} \mathbf{x}_{s-1-r}^{(32-m-1)} \\ &= \sum_{m=0}^{63} \cos(2\pi(n+1/2)(m-16)/64) (V^{(m)} \mathbf{x}^{(32-m-1)})_{s-1}, \end{aligned}$$

where we recognized  $x_{32(s-r)-m-1}$  in terms of the polyphase components of  $\mathbf{x}$ , and the inner sum as a convolution. We remark that the inner terms  $\{(V^{(m)} \mathbf{x}^{(32-m-1)})_{s-1}\}_{m=0}^{63}$  here are what the standard calls partial calculations (windowing refers to multiplication with the combined set of filter coefficients of the  $V^{(m)}$ ), and that matrixing here represents the multiplication with the cosine entries. Since  $\mathbf{z}^{(n)} = \{z_{32(s-1)+n}\}_{s=0}^{\infty}$  is the  $n$ 'th polyphase component of  $\mathbf{z}$ , this can be written as

$$\mathbf{z}^{(n)} = \sum_{m=0}^{63} \cos(2\pi(n+1/2)(m-16)/64) IV^{(m)} \mathbf{x}^{(32-m-1)}.$$

In terms of matrices this can be written as

$$\mathbf{z} = \begin{pmatrix} \cos(2\pi(0+1/2) \cdot (-16)/64) I & \cdots & \cos(2\pi(0+1/2) \cdot (47)/64) I \\ \vdots & \ddots & \vdots \\ \cos(2\pi(31+1/2) \cdot (-16)/64) I & \cdots & \cos(2\pi(31+1/2) \cdot (47)/64) I \end{pmatrix} \\ \times \begin{pmatrix} V^{(0)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(62)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(63)} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(31)} \\ \mathbf{x}^{(30)} \\ \vdots \\ \mathbf{x}^{(-32)} \end{pmatrix}.$$

If we place the 15 first columns in the cosine matrix last using Property (6.35) (we must then also place the 15 first rows last in the second matrix), we obtain

$$\mathbf{z} = \begin{pmatrix} \cos(2\pi(0+1/2) \cdot (0)/64) I & \cdots & \cos(2\pi(0+1/2) \cdot (63)/64) I \\ \vdots & \ddots & \vdots \\ \cos(2\pi(31+1/2) \cdot (0)/64) I & \cdots & \cos(2\pi(31+1/2) \cdot (63)/64) I \end{pmatrix} \\ \times \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & V^{(16)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & V^{(63)} \\ -V^{(0)} & \cdots & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \mathbf{0} \\ \mathbf{0} & \cdots & -V^{(15)} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(31)} \\ \mathbf{x}^{(30)} \\ \vdots \\ \mathbf{x}^{(-32)} \end{pmatrix}.$$

Using Equation (6.36) to combine column  $k$  and  $64 - k$  in the cosine matrix (as well as row  $k$  and  $64 - k$  in the second matrix), we can write this as

$$\begin{pmatrix} \cos(2\pi(0+1/2) \cdot (0)/64) I & \cdots & \cos(2\pi(0+1/2) \cdot (31)/64) I \\ \vdots & \ddots & \vdots \\ \cos(2\pi(31+1/2) \cdot (0)/64) I & \cdots & \cos(2\pi(31+1/2) \cdot (31)/64) I \end{pmatrix} (A' \quad B') \begin{pmatrix} \mathbf{x}^{(31)} \\ \mathbf{x}^{(30)} \\ \vdots \\ \mathbf{x}^{(-32)} \end{pmatrix}.$$

where

$$A' = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(15)} & \mathbf{0} & V^{(17)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(31)} \\ V^{(0)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix}$$

$$B' = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ V^{(32)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & V^{(33)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -V^{(63)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(47)} & \mathbf{0} & -V^{(49)} & \cdots & \mathbf{0} \end{pmatrix}.$$

Using Equation (4.3), the cosine matrix here can be written as

$$\sqrt{\frac{M}{2}} (D_M)^T \begin{pmatrix} \sqrt{2} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}.$$

The above can thus be written as

$$4(D_{32})^T (A \ B) \begin{pmatrix} \mathbf{x}^{(31)} \\ \mathbf{x}^{(30)} \\ \vdots \\ \mathbf{x}^{(-32)} \end{pmatrix},$$

where  $A$  and  $B$  are the matrices  $A', B'$  with the first row multiplied by  $\sqrt{2}$  (i.e. replace  $V^{(16)}$  with  $\sqrt{2}V^{(16)}$  in the matrix  $A'$ ). Using that  $\mathbf{x}^{(-i)} = E_1 \mathbf{x}_i$  for  $1 \leq i \leq 32$ , we can write this as

$$4(D_{32})^T (A \ B) \begin{pmatrix} \mathbf{x}^{(31)} \\ \vdots \\ \mathbf{x}^{(0)} \\ E_1 \mathbf{x}^{(31)} \\ \vdots \\ E_1 \mathbf{x}^{(0)} \end{pmatrix} = 4(D_{32})^T \left( A \begin{pmatrix} \mathbf{x}^{(31)} \\ \vdots \\ \mathbf{x}^{(0)} \end{pmatrix} + B \begin{pmatrix} E_1 \mathbf{x}^{(31)} \\ \vdots \\ E_1 \mathbf{x}^{(0)} \end{pmatrix} \right),$$

which can be written as

$$4(D_{32})^T \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \sqrt{2}V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(15)} & \mathbf{0} & V^{(17)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(31)} \\ V^{(0)} + E_1V^{(32)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & E_1V^{(33)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -E_1V^{(63)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & E_1V^{(47)} & \mathbf{0} & -E_1V^{(49)} & \cdots & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(31)} \\ \vdots \\ \mathbf{x}^{(0)} \end{pmatrix},$$

which also can be written as

$$4(D_{32})^T \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & \sqrt{2}V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & V^{(17)} & \mathbf{0} & V^{(15)} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ V^{(31)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(1)} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(0)} + E_1V^{(32)} \\ -E_1V^{(63)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_1V^{(33)} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & -E_1V^{(49)} & \mathbf{0} & E_1V^{(47)} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(0)} \\ \vdots \\ \mathbf{x}^{(31)} \end{pmatrix}.$$

We have therefore proved the following result.

**Theorem 8.11.** *Polyphase factorization of a forward filter bank transform based on a prototype filter.*

The polyphase form of a forward filter bank transform based on a prototype filter can be factored as

$$4(D_{32})^T \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & \sqrt{2}V^{(16)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & V^{(17)} & \mathbf{0} & V^{(15)} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ V^{(31)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & V^{(1)} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & V^{(0)} + E_1V^{(32)} \\ -E_1V^{(63)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_1V^{(33)} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & -E_1V^{(49)} & \mathbf{0} & E_1V^{(47)} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix} \quad (8.20)$$

Due to Theorem 6.26, it is also very simple to write down the polyphase factorization of the reverse filter bank transform as well. Since  $E_{481}G^T$  is a

forward filter bank transform where the prototype filter has been reversed,  $E_{481}G^T$  can be factored as above, with  $V^{(m)}$  replaced by  $W^{(m)}$ , with  $W^{(m)}$  being the filters derived from the synthesis prototype filter in reverse order. This means that the polyphase form of  $G$  can be factored as

$$\begin{aligned}
 & 4 \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & (W^{(31)})^T & \mathbf{0} & -E_{-1}(W^{(63)})^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & (W^{(17)})^T & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -E_{-1}(W^{(49)})^T \\ \sqrt{2}(W^{(16)})^T & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & (W^{(15)})^T & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_{-1}(W^{(47)})^T \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & (W^{(1)})^T & \mathbf{0} & E_{-1}(W^{(33)})^T & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & (W^{(0)})^T + E_{-1}(W^{(32)})^T & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \\
 & \times D_{32}E_{481}. \tag{8.21}
 \end{aligned}$$

Now, if we define  $U^{(m)}$  as the filters derived from the synthesis prototype filter itself, we have that

$$(W^{(k)})^T = -E_{-14}V^{(64-k)}, \quad 1 \leq k \leq 15 \quad (W^{(0)})^T = E_{-16}V^{(0)}.$$

Inserting this in Equation (8.21) we get the following result:

**Theorem 8.12.** *Polyphase factorization of a reverse filter bank transform based on a prototype filter.*

Assume that  $G$  is a reverse filter filter bank transform based on a prototype filter, and that  $U^{(m)}$  are the filters derived from this prototype filter. Then the polyphase form of  $G$  can be factored as

$$\begin{aligned}
 & 4 \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & -U^{(33)} & \mathbf{0} & E_{-1}U^{(1)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & -U^{(47)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & E_{-1}U^{(15)} \\ -\sqrt{2}U^{(48)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & -U^{(49)} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -E_{-1}U^{(17)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -U^{(63)} & \mathbf{0} & -E_{-1}U^{(31)} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & E_{-2}U^{(0)} - E_{-1}U^{(32)} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \\
 & \times D_{32}E_{33}. \tag{8.22}
 \end{aligned}$$

Now, consider the matrices

$$\begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -E_1V^{(64-i)} & E_1V^{(32+i)} \end{pmatrix} \text{ and } \begin{pmatrix} -U^{(32+i)} & E_{-1}U^{(i)} \\ -U^{(64-i)} & -E_{-1}U^{(32-i)} \end{pmatrix}. \tag{8.23}$$

for  $1 \leq i \leq 15$ . These make out submatrices in the matrices in equations (8.20) and (8.22). Clearly, only the product of these matrices influence the result. Since

$$\begin{aligned} & \begin{pmatrix} -U^{(32+i)} & E_{-1}U^{(i)} \\ -U^{(64-i)} & -E_{-1}U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -E_1V^{(64-i)} & E_1V^{(32+i)} \end{pmatrix} \\ &= \begin{pmatrix} -U^{(32+i)} & U^{(i)} \\ -U^{(64-i)} & -U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix} \end{aligned} \quad (8.24)$$

we have the following result.

**Theorem 8.13.** *Filter bank transforms.*

Let  $H, G$  be forward and reverse filter bank transforms defined from analysis and synthesis prototype filters. Let also  $V^{(k)}$  be the prototype filter of  $H$ , and  $U^{(k)}$  the reverse of the prototype filter of  $G$ . If

$$\begin{aligned} & \begin{pmatrix} -U^{(32+i)} & U^{(i)} \\ -U^{(64-i)} & -U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix} = c \begin{pmatrix} E_d & \mathbf{0} \\ \mathbf{0} & E_d \end{pmatrix} \\ & \quad (\sqrt{2}V^{(16)})(-\sqrt{2}U^{(48)}) = cE_d \\ & (V^{(0)} + E_1V^{(32)})(E_{-2}U^{(0)} - E_{-1}U^{(32)}) = cE_d \end{aligned} \quad (8.25)$$

for  $1 \leq i \leq 15$ , then  $GH = 16cE_{33+32d}$ .

This result is the key ingredient we need in order to construct forward and reverse systems which together give perfect reconstruction. In Exercise 8.15 we go through how we can use lifting in order to express a wide range of possible  $(U, V)$  matrix pairs which satisfy Equation (8.25). This turns the problem of constructing cosine-modulated filter banks which are useful for audio coding into an optimization problem: the optimization variables are values  $\lambda_i$  which characterize lifting steps, and the objective function is the deviation of the corresponding prototype filter from an ideal bandpass filter. This optimization problem has been subject to a lot of research, and we will not go into details on this.

### 8.3.2 The prototype filters

Now, let us return to the MP3 standard. We previously observed that in this standard the coefficients in the synthesis prototype filter seemed to equal 32 times the analysis prototype filter. This indicates that  $U^{(k)} = 32V^{(k)}$ . A closer inspection also yields that there is a symmetry in the values of the prototype filter: We see that  $C_i = -C_{512-i}$  (i.e. antisymmetry) for most values of  $i$ . The only exception is for  $i = 64, 128, \dots, 448$ , for which  $C_i = C_{512-i}$  (i.e. symmetry). The antisymmetry can be translated to that the filter coefficients of  $V^{(k)}$  equal those of  $V^{(64-k)}$  in reverse order, with a minus sign. The symmetry can be translated to that  $V^{(0)}$  is symmetric. These observations can be rewritten as

$$V^{(64-k)} = -E_{14}(V^{(k)})^T, 1 \leq k \leq 15. \quad (8.26)$$

$$V^{(0)} = E_{16}(V^{(0)})^T. \quad (8.27)$$

Inserting first that  $U^{(k)} = 32V^{(k)}$  in Equation (8.24) gives

$$\begin{aligned} & \begin{pmatrix} -U^{(32+i)} & U^{(i)} \\ -U^{(64-i)} & -U^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix} \\ &= 32 \begin{pmatrix} -V^{(32+i)} & V^{(i)} \\ -V^{(64-i)} & -V^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix}. \end{aligned}$$

Substituting for  $V^{(32+i)}$  and  $V^{(64-i)}$  after what we found by inspection now gives

$$\begin{aligned} & 32 \begin{pmatrix} E_{14}(V^{(32-i)})^T & V^{(i)} \\ E_{14}(V^{(i)})^T & -V^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ E_{14}(V^{(i)})^T & -E_{14}(V^{(32-i)})^T \end{pmatrix} \\ &= 32 \begin{pmatrix} E_{14} & \mathbf{0} \\ \mathbf{0} & E_{14} \end{pmatrix} \begin{pmatrix} (V^{(32-i)})^T & V^{(i)} \\ (V^{(i)})^T & -V^{(32-i)} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix} \\ &= 32 \begin{pmatrix} E_{14} & \mathbf{0} \\ \mathbf{0} & E_{14} \end{pmatrix} \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix}^T \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix} \\ &= 32 \begin{pmatrix} E_{14} & \mathbf{0} \\ \mathbf{0} & E_{14} \end{pmatrix} \begin{pmatrix} V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T & \mathbf{0} \\ \mathbf{0} & V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T \end{pmatrix}. \end{aligned} \quad (8.28)$$

Due to Exercise 8.6 (set  $A = (V^{(32-i)})^T, B = (V^{(i)})^T$ ), with

$$H = \begin{pmatrix} V^{(32-i)} & V^{(i)} \\ (V^{(i)})^T & -(V^{(32-i)})^T \end{pmatrix} \quad G = \begin{pmatrix} (V^{(32-i)})^T & V^{(i)} \\ (V^{(i)})^T & -V^{(32-i)} \end{pmatrix}$$

we recognize an alternative QMF filter bank. We thus have alias cancellation, with perfect reconstruction only if  $|\lambda_{H_0}(\omega)|^2 + |\lambda_{H_0}(\omega + \pi)|^2 = 1$ . For the two remaining filters we compute

$$\begin{aligned} & (\sqrt{2}V^{(16)})(-\sqrt{2}U^{(48)}) \\ &= -64V^{(16)}V^{(48)} = 64E_{14}V^{(16)}(V^{(16)})^T = 32E_{14}(V^{(16)}(V^{(16)})^T + V^{(16)}(V^{(16)})^T) \end{aligned} \quad (8.29)$$

and

$$\begin{aligned} & (V^{(0)} + E_1V^{(32)})(E_{-2}U^{(0)} - E_{-1}U^{(32)}) \\ &= 32(V^{(0)} + E_1V^{(32)})(E_{-2}V^{(0)} - E_{-1}V^{(32)}) = 32E_{-2}(V^{(0)} + E_1V^{(32)})(V^{(0)} - E_1V^{(32)}) \\ &= 32E_{-2}(V^{(0)}(V^{(0)})^T - (V^{(32)}(V^{(32)})^T) = 32E_{14}((V^{(0)}(V^{(0)})^T + V^{(32)}(V^{(32)})^T)). \end{aligned} \quad (8.30)$$

We see that the filters from equations (8.28)-(8.30) are similar, and that we thus can combine them into

$$\{V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T\}_{i=0}^{16}. \quad (8.31)$$

All of these can be the identity, except for  $1024V^{(16)}(V^{(16)})^T$ , since we know that the product of two FIR filters is never the identity, except when both are delays (And all  $V^{(m)}$  are FIR, since the prototype filters defined by the MP3 standard are FIR). This single filter is thus what spoils for perfect reconstruction, so that we can only hope for alias cancellation, and this happens when the filters from Equation (8.31) all are equal. Ideally this is close to  $cI$  for some scalar  $c$ , and we then have that

$$GH = 16 \cdot 32cE_{33+448} = 512cE_{481}I.$$

This explains the observation from the MP3 standard that  $GH$  seems to be close to  $E_{481}$ . Since all the filters  $V^{(i)}(V^{(i)})^T + V^{(32-i)}(V^{(32-i)})^T$  are symmetric,  $GH$  is also a symmetric filter due to Theorem 8.3, so that its frequency response is real, so that we have no phase distortion. We can thus summarize our findings as follows.

**Observation 8.14.** *MP3 standard.*

The prototype filters from the MP3 standard do not give perfect reconstruction. They are found by choosing 17 filters  $\{V^{(k)}\}_{k=0}^{16}$  so that the filters from Equation (8.31) are equal, and so that their combination into a prototype filter using equations (8.19) and (8.26) is as close to an ideal bandpass filter as possible. When we have equality the alias cancellation condition is satisfied, and we also have no phase distortion. When the common value is close to  $\frac{1}{512}I$ ,  $GH$  is close to  $E_{481}$ , so that we have near-perfect reconstruction.

This states clearly the optimization problem which the values stated in the MP3 standard solves.

### 8.3.3 Perfect reconstruction

How can we overcome the problem that  $1024V^{(16)}(V^{(16)})^T \neq I$ , which spoiled for perfect reconstruction in the MP3 standard? It turns out that we can address this a simple change in our procedure. In Equation (8.18) we replace with

$$z_{32(s-1)+n} = \sum_{k=0}^{511} \cos((n+1/2)(k+1/2-16)\pi/32)h_k x_{32s-k-1}, \quad (8.32)$$

i.e.  $1/2$  is added inside the cosine. We now have the properties

$$\cos(2\pi(n+1/2)(k+64r+1/2)/(2N)) = (-1)^r \cos(2\pi(n+1/2)(k+1/2)/(2N)) \quad (8.33)$$

$$\cos(2\pi(n+1/2)(2N-k-1+1/2)/(2N)) = -\cos(2\pi(n+1/2)(k+1/2)/(2N)). \quad (8.34)$$



Due to the first property, we can deduce as before that

$$\mathbf{z}^{(n)} = \sum_{m=0}^{63} \cos(2\pi(n+1/2)(m+1/2-16)/64) IV^{(m)} \mathbf{x}^{(32-m-1)},$$

where the filters  $V^{(m)}$  are defined as before. As before placing the 15 first columns of the cosine-matrix last, but instead using Property (8.34) to combine columns  $k$  and  $64-k-1$  of the cosine-matrix, we can write this as

$$\begin{pmatrix} \cos(2\pi(0+1/2) \cdot (0+1/2)/64) I & \cdots & \cos(2\pi(0+1/2) \cdot (31+1/2)/64) I \\ \vdots & \ddots & \vdots \\ \cos(2\pi(31+1/2) \cdot (0+1/2)/64) I & \cdots & \cos(2\pi(31+1/2) \cdot (31+1/2)/64) I \end{pmatrix} (A \ B) \begin{pmatrix} \mathbf{x}^{(31)} \\ \vdots \\ \mathbf{x}^{(-32)} \end{pmatrix}$$

where

$$A = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & V^{(15)} & V^{(16)} & \cdots & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & V^{(1)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & V^{(30)} & \mathbf{0} \\ V^{(0)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & V^{(31)} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix}$$

$$B = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ V^{(32)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -V^{(63)} \\ \mathbf{0} & V^{(33)} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & -V^{(62)} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & V^{(47)} & -V^{(48)} & \cdots & \cdots & \mathbf{0} \end{pmatrix}.$$

Since the cosine matrix can be written as  $\sqrt{\frac{M}{2}} D_M^{(iv)}$ , the above can be written as

$$4D_M^{(iv)} (A \ B) \begin{pmatrix} \mathbf{x}^{(31)} \\ \vdots \\ \mathbf{x}^{(-32)} \end{pmatrix}.$$

As before we can rewrite this as

$$4D_M^{(iv)} \begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(31)} \\ \vdots \\ \mathbf{x}^{(0)} \\ E_1 \mathbf{x}^{(31)} \\ \vdots \\ E_1 \mathbf{x}^{(0)} \end{pmatrix} = 4D_M^{(iv)} \left( A \begin{pmatrix} \mathbf{x}^{(31)} \\ \vdots \\ \mathbf{x}^{(0)} \end{pmatrix} + B \begin{pmatrix} E_1 \mathbf{x}^{(31)} \\ \vdots \\ E_1 \mathbf{x}^{(0)} \end{pmatrix} \right),$$

which can be written as

$$4D_M^{(iv)} \begin{pmatrix} \mathbf{0} & \mathbf{0} & \dots & V^{(15)} & V^{(16)} & \dots & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & V^{(1)} & \dots & \mathbf{0} & \mathbf{0} & \dots & V^{(30)} & \mathbf{0} \\ V^{(0)} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \dots & V^{(31)} \\ E_1 V^{(32)} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \dots & -E_1 V^{(63)} \\ \mathbf{0} & E_1 V^{(33)} & \dots & \mathbf{0} & \mathbf{0} & \dots & -E_1 V^{(62)} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & E_1 V^{(47)} & -E_1 V^{(48)} & \dots & \dots & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(31)} \\ \vdots \\ \mathbf{x}^{(0)} \end{pmatrix},$$

which also can be written as

$$4D_M^{(iv)} \begin{pmatrix} \mathbf{0} & \mathbf{0} & \dots & V^{(16)} & V^{(15)} & \dots & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & V^{(30)} & \dots & \mathbf{0} & \mathbf{0} & \dots & V^{(1)} & \mathbf{0} \\ V^{(31)} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \dots & V^{(0)} \\ -E_1 V^{(63)} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \dots & E_1 V^{(32)} \\ \mathbf{0} & -E_1 V^{(62)} & \dots & \mathbf{0} & \mathbf{0} & \dots & E_1 V^{(33)} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & -E_1 V^{(48)} & E_1 V^{(47)} & \dots & \dots & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(0)} \\ \vdots \\ \mathbf{x}^{(31)} \end{pmatrix}.$$

We therefore have the following result

**Theorem 8.15.** *Polyphase factorization of a forward filter bank transform based on a prototype filter, modified version.*

The modified version of the polyphase form of a forward filter bank transform based on a prototype filter can be factored as

$$4D_M^{(iv)} \begin{pmatrix} \mathbf{0} & \mathbf{0} & \dots & V^{(16)} & V^{(15)} & \dots & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & V^{(30)} & \dots & \mathbf{0} & \mathbf{0} & \dots & V^{(1)} & \mathbf{0} \\ V^{(31)} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \dots & V^{(0)} \\ -E_1 V^{(63)} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \dots & E_1 V^{(32)} \\ \mathbf{0} & -E_1 V^{(62)} & \dots & \mathbf{0} & \mathbf{0} & \dots & E_1 V^{(33)} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & -E_1 V^{(48)} & E_1 V^{(47)} & \dots & \dots & \mathbf{0} \end{pmatrix} \quad (8.35)$$

Clearly this factorization avoids having two blocks of filters: There are now  $16 \times 2 \times 2$ -polyphase matrices, and as we know, each of them can be invertible, so that the full matrix can be inverted in a similar fashion as before. It is therefore now possible to obtain perfect reconstruction. Although we do not state recipes for implementing this, one has just as efficient implementations as in the MP3 standard.

Since we ended up with the  $2 \times 2$  polyphase matrices  $M_k$ , we can apply the lifting factorization in order to halve the number of multiplications/additions. This is not done in practice, since a lifting factorization requires that we compute all outputs at once. In audio coding it is required that we compute the output progressively, due to the large size of the input vector. The procedure above is therefore mostly useful for providing the requirements for the filters, while the preceding comments can be used for the implementation.

**Exercise 8.13: Run forward and reverse transform**

Run the forward and then the reverse transform from Exercise 6.26 on the vector  $(1, 2, 3, \dots, 8192)$ . Verify that there seems to be a delay on 481 elements, as promised by Theorem 8.14. Do you get the exact same result back?

**Exercise 8.14: Verify statement of filters**

Use your computer to verify the symmetries we have stated for the symmetries in the prototype filters, i.e. that

$$C_i = \begin{cases} -C_{512-i} & i \neq 64, 128, \dots, 448 \\ C_{512-i} & i = 64, 128, \dots, 448. \end{cases}$$

Explain also that this implies that  $h_i = h_{512-i}$  for  $i = 1, \dots, 511$ . In other words, the prototype filter has symmetry around  $(511 + 1)/2 = 256$ , so that it has linear phase.

**Exercise 8.15: Lifting**

We mentioned that we could use the lifting factorization to construct filters on the form stated in Equation (8.19), so that the matrices on the form given by Equation (8.23), i.e.

$$\begin{pmatrix} V^{(32-i)} & V^{(i)} \\ -V^{(64-i)} & V^{(32+i)} \end{pmatrix},$$

are invertible. Let us see what kind of lifting steps produce such matrices.

a) Show that the lifting steps

$$\begin{pmatrix} I & \lambda E_2 \\ \mathbf{0} & I \end{pmatrix} \text{ and } \begin{pmatrix} I & \mathbf{0} \\ \lambda I & I \end{pmatrix}$$

applied in alternating order to a matrix on the form given by Equation (8.23), where the filters are on the form given by Equation (8.19), again produces matrices and filters on these forms. This explains how we can parametrize a larger number of such matrices with the help of lifting steps. It also explains why the inverse matrix is on the form stated in Equation (8.23) with filters on the same form, since the inverse lifting steps are of the same type.

b) Explain that 16 numbers  $\{\lambda_i\}_{i=1}^{16}$  are needed (together with what we start with on the diagonal in the lifting construction), in order to construct filters so that the prototype filter has 512 coefficients. Since there are 15 submatrices, this gives 240 optimization variables.

Lifting gives the following strategy for finding a corresponding synthesis prototype filter which gives perfect reconstruction: First compute matrices  $V, W$  which are inverses of one another using lifting (using the lifting steps of this exercise ensures that all filters will be on the form stated in Equation (8.19)), and write

$$\begin{aligned} VW &= \begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} W^{(1)} & -W^{(3)} \\ W^{(2)} & W^{(4)} \end{pmatrix} = \begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} (W^{(1)})^T & (W^{(2)})^T \\ -(W^{(3)})^T & (W^{(4)})^T \end{pmatrix}^T \\ &= \begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} E_{15}(W^{(1)})^T & E_{15}(W^{(2)})^T \\ -E_{15}(W^{(3)})^T & E_{15}(W^{(4)})^T \end{pmatrix}^T \begin{pmatrix} E_{15} & \mathbf{0} \\ \mathbf{0} & E_{15} \end{pmatrix} = I. \end{aligned}$$

Now, the matrices  $U^{(i)} = E_{15}(W^{(i)})^T$  are on the form stated in Equation (8.19), and we have that

$$\begin{pmatrix} V^{(1)} & V^{(2)} \\ -V^{(3)} & V^{(4)} \end{pmatrix} \begin{pmatrix} U^{(1)} & U^{(2)} \\ -U^{(3)} & U^{(4)} \end{pmatrix} = \begin{pmatrix} E_{-15} & \mathbf{0} \\ \mathbf{0} & E_{-15} \end{pmatrix}$$

We can now conclude from Theorem 8.13 that if we define the synthesis prototype filter as therein, and set  $c = 1, d = -15$ , we have that  $GH = 16E_{481-32 \cdot 15} = 16E_1$ .

## 8.4 Summary

We defined the polyphase representation of a matrix, and proved some useful properties. For filter bank transforms, the polyphase representation was a block matrix where the blocks are filters, and these blocks/filters were called polyphase components. In particular, the filter bank transforms of wavelets were  $2 \times 2$ -block matrices of filters. We saw that, for wavelets, the polyphase representation could be realized through a rearrangement of the wavelet bases, and thus paralleled the development in Chapter 6 for expressing the DWT in terms of filters, where we instead rearranged the target base of the DWT.

We showed with two examples that factoring the polyphase representation into simpler matrices (also referred to as a polyphase factorization) could be a useful technique. First, for wavelets ( $M = 2$ ), we established the lifting factorization. This is useful not only since it factorizes the DWT and the IDWT into simpler operations, but also since it reduces the number of arithmetic operations in these. The lifting factorization is therefore also used in practical implementations, and we applied it to some of the wavelets we constructed in Chapter 7. The JPEG2000 standard document [21] explains a procedure for implementing some of these wavelet transforms using lifting, and the values of the lifting steps used in the standard thus also appear here.

The polyphase representation was also useful for proving the characterization of wavelets we encountered in Chapter 7, which we used to find expressions for many useful wavelets.

The polyphase representation was also useful to explain how the prototype filters of the MP3 standard should be chosen, in order for the reverse filter bank transform to invert the forward filter bank transform. Again this was attacked by factoring the polyphase representation of the forward and reverse filter bank transforms. The parts of the factorization which represented the prototype filters were represented by a sparse matrix, and it was clear from this matrix what properties we needed to put on the prototype filter, in order to have alias cancellation, and no phase distortion. In fact, we proved that the MP3 standard could not possibly give perfect reconstruction, but it was very clear from our construction how the filter bank could be modified in order for the overall system to provide perfect reconstruction.

The lifting scheme as introduced here was first proposed by Sweldens [45]. How to use lifting for in-place calculation for the DWT was also suggested by Sweldens [44].

This development concludes the one-dimensional aspect of wavelets in this book. In the following we will extend our theory to also apply for images. Images will be presented in Chapter 9. After that we will define the tensor product concept, which will be the key ingredient to apply wavelets to two-dimensional objects such as images.

## Chapter 9

# Digital images

Upto now we have presented wavelets in a one-dimensional setting. Images, however, are two-dimensional by nature. This poses another challenge, which we did not encounter in the case of sound signals. In this chapter we will establish the mathematics to handle this, but first we will present some basics on images, as well as how they can be represented and manipulated with simple mathematics. Images are a very important type of digital media, and this material is thus useful, general knowledge for anyone with a digital camera and a computer. For many scientists this material is also an essential tool. As an example, in astrophysics data from both satellites and distant stars and galaxies is collected in the form of images, and information is extracted from the images with advanced image processing techniques. As another example, medical imaging makes it possible to gather different kinds of information in the form of images, even from the inside of the body. By analysing these images it is possible to discover tumours and other disorders.

We will see how filter-based operations extend naturally to the two-dimensional setting of images. Smoothing and edge detections are the two main examples of filter-based operations we will consider for images. The key mathematical concept in this extension is the *tensor product*, which can be thought of as a general tool for constructing two-dimensional objects from one-dimensional counterparts. We will also see that the tensor product allows us to establish an efficient implementation of filtering for images, efficient meaning a complexity substantially less than what is required by general linear transformations.

We will finally consider useful coordinate changes for images. Recall that the DFT, the DCT, and the wavelet transform were all defined as changes of coordinates for vectors or functions of one variable, and therefore cannot be directly applied to two-dimensional data like images. It turns out that the tensor product can also be used to extend changes of coordinates to a two-dimensional setting.

Functionality for accessing images are collected in a module called `images`.

## 9.1 What is an image?

Before we do computations with images, it is helpful to be clear about what an image really is. Images cannot be perceived unless there is some light present, so we first review superficially what light is.

### Light.

#### Fact 9.1. *Light.*

Light is electromagnetic radiation with wavelengths in the range 400–700 nm (1 nm is  $10^{-9}$  m): Violet has wavelength 400 nm and red has wavelength 700 nm. White light contains roughly equal amounts of all wave lengths.

Other examples of electromagnetic radiation are gamma radiation, ultraviolet and infrared radiation and radio waves, and all electromagnetic radiation travel at the speed of light ( $\approx 3 \times 10^8$  m/s). Electromagnetic radiation consists of waves and may be reflected and refracted, just like sound waves (but sound waves are not electromagnetic waves).

We can only see objects that emit light, and there are two ways that this can happen. The object can emit light itself, like a lamp or a computer monitor, or it reflects light that falls on it. An object that reflects light usually absorbs light as well. If we perceive the object as red it means that the object absorbs all light except red, which is reflected. An object that emits light is different; if it is to be perceived as being red it must emit only red light.

**Digital output media.** Our focus will be on objects that emit light, for example a computer display. A computer monitor consists of a matrix of small dots which emit light. In most technologies, each dot is really three smaller dots, and each of these smaller dots emit red, green and blue light. If the amounts of red, green and blue is varied, our brain merges the light from the three small light sources and perceives light of different colors. In this way the color at each set of three dots can be controlled, and a color image can be built from the total number of dots.

It is important to realise that it is possible to generate most, but not all, colors by mixing red, green and blue. In addition, different computer monitors use slightly different red, green and blue colors, and unless this is taken into consideration, colors will look different on the two monitors. This also means that some colors that can be displayed on one monitor may not be displayable on a different monitor.

Printers use the same principle of building an image from small dots. On most printers however, the small dots do not consist of smaller dots of different colors. Instead as many as 7–8 different inks (or similar substances) are mixed to the right color. This makes it possible to produce a wide range of colors, but not all, and the problem of matching a color from another device like a monitor is at least as difficult as matching different colors across different monitors.

Video projectors build an image that is projected onto a wall. The final image is therefore a reflected image and it is important that the surface is white so that it reflects all colors equally.

The quality of a device is closely linked to the density of the dots.

**Fact 9.2.** *Resolution.*

The resolution of a medium is the number of dots per inch (dpi). The number of dots per inch for monitors is usually in the range 70–120, while for printers it is in the range 150–4800 dpi. The horizontal and vertical densities may be different. On a monitor the dots are usually referred to as *pixels* (picture elements).

**Digital input media.** The two most common ways to acquire digital images is with a digital camera or a scanner. A scanner essentially takes a photo of a document in the form of a matrix of (possibly colored) dots. As for printers, an important measure of quality is the number of dots per inch.

**Fact 9.3.** *Printers.*

The resolution of a scanner usually varies in the range 75 dpi to 9600 dpi, and the color is represented with up to 48 bits per dot.

For digital cameras it does not make sense to measure the resolution in dots per inch, as this depends on how the image is printed (its size). Instead the resolution is measured in the number of dots recorded.

**Fact 9.4.** *Pixels.*

The number of pixels recorded by a digital camera usually varies in the range  $320 \times 240$  to  $6000 \times 4000$  with 24 bits of color information per pixel. The total number of pixels varies in the range 76 800 to 24 000 000 (0.077 megapixels to 24 megapixels).

For scanners and cameras it is easy to think that the more dots (pixels), the better the quality. Although there is some truth to this, there are many other factors that influence the quality. The main problem is that the measured color information is very easily polluted by noise. And of course high resolution also means that the resulting files become very big; an uncompressed  $6000 \times 4000$  image produces a 72 MB file. The advantage of high resolution is that you can magnify the image considerably and still maintain reasonable quality.

**Definition of digital image.** We have already talked about digital images, but we have not yet been precise about what they are. From a mathematical point of view, an image is quite simple.

**Fact 9.5.** *Digital image.*

A digital image  $P$  is a matrix of *intensity values*  $\{p_{i,j}\}_{i,j=1}^{M,N}$ . For grey-level images, the value  $p_{i,j}$  is a single number, while for color images each  $p_{i,j}$  is a vector of three or more values. If the image is recorded in the rgb-model, each  $p_{i,j}$  is a vector of three values,

$$p_{i,j} = (r_{i,j}, g_{i,j}, b_{i,j}),$$



that denote the amount of red, green and blue at the point  $(i, j)$ .

Note that, when referring to the coordinates  $(i, j)$  in an image,  $i$  will refer to row index,  $j$  to column index, in the same way as for matrices. In particular, the top row in the image has coordinates  $\{(0, j)\}_{j=0}^{N-1}$ , while the left column in the image has coordinates  $\{(i, 0)\}_{i=0}^{M-1}$ . With this notation, the dimension of the image is  $M \times N$ . The value  $p_{i,j}$  gives the color information at the point  $(i, j)$ . It is important to remember that there are many formats for this. The simplest case is plain black and white images in which case  $p_{i,j}$  is either 0 or 1. For grey-level images the intensities are usually integers in the range 0–255. However, we will assume that the intensities vary in the interval  $[0, 1]$ , as this sometimes simplifies the form of some mathematical functions. For color images there are many different formats, but we will just consider the rgb-format mentioned in the fact box. Usually the three components are given as integers in the range 0–255, but as for grey-level images, we will assume that they are real numbers in the interval  $[0, 1]$  (the conversion between the two ranges is straightforward, see Example 9.3 below).



Figure 9.1: Our test image.

In Figure 9.1 we have shown the test image we will work with, called the *Lena image*. It is named after the girl in the image. This image is also used as a test image in many textbooks on image processing.

In Figure 9.2 we have shown the corresponding black and white, and grey-level versions of the test image.

**Fact 9.6.** *Intensity.*

In these notes the intensity values  $p_{i,j}$  are assumed to be real numbers in the interval  $[0, 1]$ . For color images, each of the red, green, and blue intensity values are assumed to be real numbers in  $[0, 1]$ .



Figure 9.2: Black and white (left), and grey-level (right) versions of the image in Figure 9.1.



Figure 9.3:  $18 \times 18$  pixels excerpt of the color image in Figure 9.1. The grid indicates the borders between the pixels.

If we magnify the part of the color image in Figure 9.1 around one of the eyes, we obtain the images in figures 9.3-9.4. As we can see, the pixels have been magnified to big squares. This is a standard representation used by many programs — the actual shape of the pixels will depend on the output medium. Nevertheless, we will consider the pixels to be square, with integer coordinates at their centers, as indicated by the grids in figures 9.3-9.4.

**Fact 9.7.** *Shape of pixel.*

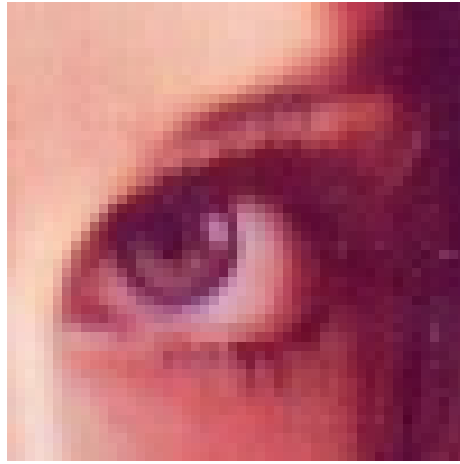


Figure 9.4:  $50 \times 50$  pixels excerpt of the color image in Figure 9.1.

The pixels of an image are assumed to be square with sides of length one, with the pixel with value  $p_{i,j}$  centered at the point  $(i, j)$ .

## 9.2 Some simple operations on images with Python

Images are two-dimensional matrices of numbers, contrary to the sound signals we considered in the previous section. In this respect it is quite obvious that we can manipulate an image by performing mathematical operations on the numbers. In this section we will consider some of the simpler operations. In later sections we will go through more advanced operations, and explain how the theory for these can be generalized from the corresponding theory for one-dimensional (sound) signals (which we will go through first).

In order to perform these operations, we need to be able to use images with a programming environment.

An image can also be thought of as a matrix, by associating each pixel with an element in a matrix. The matrix indices thus correspond to positions in the pixel grid. Black and white images correspond to matrices where the elements are natural numbers between 0 and 255. To store a color image, we need 3 matrices, one for each color component. We will also view this as a 3-dimensional matrix. In the following, operations on images will be implemented in such a way that they are applied to each color component simultaneously. This is similar to the FFT and the DWT, where the operations were applied to each sound channel simultaneously.

Since images are viewed as 2-dimensional or 3-dimensional matrices, we can use any linear algebra software in order to work with images. After we now have made the connection with matrices, we can create images from mathematical formulas, just as we could with sound in the previous sections. But what we also

need before we go through operations on images, is, as in the sections on sound, means of reading an image from a file so that its contents are accessible as a matrix, and write images represented by a matrix which we have constructed ourself to file. Reading a function from file can be done with help of the function `imread`. If we write

```
X = double(imread('filename.fmt', 'fmt'))
```

the image with the given path and format is read, and stored in the matrix which we call `X`. 'fmt' can be 'jpg', 'tif', 'gif', 'png', and so on. This parameter is optional: If it is not present, the program will attempt to determine the format from the first bytes in the file, and from the filename. After the call to `imread`, we have a matrix where the entries represent the pixel values, and of integer data type (more precisely, the data type `uint8`). To perform operations on the image, we must first convert the entries to the data type `double`, as shown above. Similarly, the function `imwrite`

can be used to write the image represented by a matrix to file. If we write

```
imwrite(uint8(X), 'filename.fmt', 'fmt')
```

the image represented by the matrix `X` is written to the given path, in the given format. Before the image is written to file, you see that we have converted the matrix values back to the integer data type. In other words: `imread` and `imwrite` both assume integer matrix entries, while operations on matrices assume double matrix entries. If you want to print images you have created yourself, you can use this function first to write the image to a file, and then send that file to the printer using another program. Finally, we need an alternative to playing a sound, namely displaying an image. The function `imshow(uint8(X))` displays the matrix `X` as an image in a separate window. Also here we needed to convert the samples using the function `uint8`.

The following examples go through some much used operations on images.

### Example 9.1: Normalising the intensities

We have assumed that the intensities all lie in the interval  $[0, 1]$ , but as we noted, many formats in fact use integer values in the range  $[0, 255]$ . And as we perform computations with the intensities, we quickly end up with intensities outside  $[0, 1]$  even if we start out with intensities within this interval. We therefore need to be able to *normalise* the intensities. This we can do with the simple linear function

$$g(x) = \frac{x - a}{b - a}, \quad a < b,$$

which maps the interval  $[a, b]$  to  $[0, 1]$ . A simple case is mapping  $[0, 255]$  to  $[0, 1]$  which we accomplish with the scaling  $g(x) = x/255$ . More generally, we typically perform computations that result in intensities outside the interval  $[0, 1]$ . We can then compute the minimum and maximum intensities  $p_{\min}$  and  $p_{\max}$  and

map the interval  $[p_{\min}, p_{\max}]$  back to  $[0, 1]$ . Below we have shown a function `mapto01` which achieves this task.

```
def mapto01(X):
    minval, maxval = X.min(), X.max()
    X -= minval
    X /= (maxval-minval)

def contrastadjust(X,epsilon):
    """
    Assumes that the values are in [0,255]
    """
    X /= 255.
    X += epsilon
    log(X, X)
    X -= log(epsilon)
    X /= (log(1+epsilon)-log(epsilon))
    X *= 255

def contrastadjust0(X,n):
    """
    Assumes that the values are in [0,255]
    """
    X /= 255.
    X -= 1/2.
    X *= n
    arctan(X, X)
    X /= (2*arctan(n/2.))
    X += 1/2.0
    X *= 255 # Maps the values back to [0,255]
```

Several examples of using this function will be shown below. A good question here is why the functions `min` and `max` are called three times in succession. The reason is that there is a third “dimension” in play, besides the spatial  $x$ - and  $y$ -directions. This dimension describes the color components in each pixel, which are usually the red-, green-, and blue color components.

### Example 9.2: Extracting the different colors

If we have a color image

$$P = (r_{i,j}, g_{i,j}, b_{i,j})_{i,j=1}^{m,n},$$

it is often useful to manipulate the three color components separately as the three images

$$P_r = (r_{i,j})_{i,j=1}^{m,n}, \quad P_g = (g_{i,j})_{i,j=1}^{m,n}, \quad P_b = (b_{i,j})_{i,j=1}^{m,n}.$$

As an example, let us first see how we can produce three separate images, showing the R,G, and B color components, respectively. Let us take the image `lena.png` used in Figure 9.1. When the image is read (first line below), the returned object has three dimensions. The first two dimensions represent the spatial directions (the row-index and column-index). The third dimension represents the color component. One can therefore view images representing the different color components with the help of the following code:

```

X1 = zeros_like(img)
X1[:, :, 0] = img[:, :, 0]

X2 = zeros_like(img)
X2[:, :, 1] = img[:, :, 1]

X3 = zeros_like(img)
X3[:, :, 2] = img[:, :, 2]

```

The resulting images are shown in Figure 9.5.

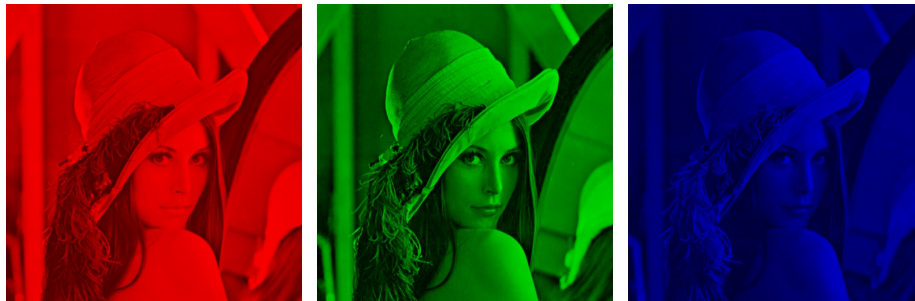


Figure 9.5: The red, green, and blue components of the color image in Figure 9.1.

### Example 9.3: Converting from color to grey-level

If we have a color image we can convert it to a grey-level image. This means that at each point in the image we have to replace the three color values  $(r, g, b)$  by a single value  $p$  that will represent the grey level. If we want the grey-level image to be a reasonable representation of the color image, the value  $p$  should somehow reflect the intensity of the image at the point. There are several ways to do this.

It is not unreasonable to use the largest of the three color components as a measure of the intensity, i.e. to set  $p = \max(r, g, b)$ . An alternative is to use the sum of the three values as a measure of the total intensity at the point. This corresponds to setting  $p = r + g + b$ . Here we have to be a bit careful with a subtle point. We have required each of the  $r$ ,  $g$  and  $b$  values to lie in the range  $[0, 1]$ , but their sum may of course become as large as 3. We also require our grey-level values to lie in the range  $[0, 1]$  so after having computed all the sums we must normalise as explained above. A third possibility is to think of the intensity of  $(r, g, b)$  as the length of the color vector, in analogy with points in space, and set  $p = \sqrt{r^2 + g^2 + b^2}$ . Again, we may end up with values in the range  $[0, \sqrt{3}]$  so we have to normalise like we did in the second case.

Let us sum this up as follows: A color image  $P = (r_{i,j}, g_{i,j}, b_{i,j})_{i,j=1}^{m,n}$  can be converted to a grey level image  $Q = (q_{i,j})_{i,j=1}^{m,n}$  by one of the following three operations:

- Set  $q_{i,j} = \max(r_{i,j}, g_{i,j}, b_{i,j})$  for all  $i$  and  $j$ .
- Compute  $\hat{q}_{i,j} = r_{i,j} + g_{i,j} + b_{i,j}$  for all  $i$  and  $j$ .
- Transform all the values to the interval  $[0, 1]$  by setting

$$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$

- Compute  $\hat{q}_{i,j} = \sqrt{r_{i,j}^2 + g_{i,j}^2 + b_{i,j}^2}$  for all  $i$  and  $j$ .
- Transform all the values to the interval  $[0, 1]$  by setting

$$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$

In practice one of the last two methods are preferred, perhaps with a preference for the last method, but the actual choice depends on the application. These can be implemented as follows.

```
mx = maximum(img[:, :, 0], img[:, :, 1])
X1 = maximum(img[:, :, 2], mx)

X2 = img[:, :, 0] + img[:, :, 1] + img[:, :, 2]
mapto01(X2)
X2 *= 255

X3 = sqrt(img[:, :, 0]**2 + img[:, :, 1]**2 + img[:, :, 2]**2)
mapto01(X3)
X3 *= 255
```

The results of applying these three operations can be seen in Figure 9.6.



Figure 9.6: Alternative ways to convert the color image in Figure 9.1 to a grey level image.

**Example 9.4: Computing the negative image**

In film-based photography a negative image was obtained when the film was developed, and then a positive image was created from the negative. We can easily simulate this and compute a negative digital image.

Suppose we have a grey-level image  $P = (p_{i,j})_{i,j=1}^{m,n}$  with intensity values in the interval  $[0, 1]$ . Here intensity value 0 corresponds to black and 1 corresponds to white. To obtain the negative image we just have to replace an intensity  $p$  by its 'mirror value'  $1 - p$ . This is also easily translated to code as above. The resulting image is shown in Figure 9.7.



Figure 9.7: The negative versions of the corresponding images in Figure 9.6.

**Example 9.5: Increasing the contrast**

A common problem with images is that the contrast often is not good enough. This typically means that a large proportion of the grey values are concentrated in a rather small subinterval of  $[0, 1]$ . The obvious solution to this problem is to somehow spread out the values. This can be accomplished by applying a monotone function  $f$  which maps  $[0, 1]$  onto  $[0, 1]$ . If we choose  $f$  so that its derivative is large in the area where many intensity values are concentrated, we obtain the desired effect. We will consider two such families of functions:

$$f_n(x) = \frac{\arctan(n(x - 1/2))}{2 \arctan(n/2)} + \frac{1}{2} \quad (9.1)$$

$$g_\epsilon(x) = \frac{\ln(x + \epsilon) - \ln \epsilon}{\ln(1 + \epsilon) - \ln \epsilon}. \quad (9.2)$$

The first type of functions have quite large derivatives near  $x = 0.5$  and will therefore increase the contrast in images with a concentration of intensities with value around 0.5. The second type of functions have a large derivative near  $x = 0$  and will therefore increase the contrast in images with a large proportion of small intensity values, i.e., very dark images. Figure 9.8 shows some examples of these functions. The three functions in the left plot in Figure 9.8 are  $f_4$ ,  $f_{10}$ , and  $f_{100}$ , the ones shown in the plot are  $g_{0.1}$ ,  $g_{0.01}$ , and  $g_{0.001}$ .



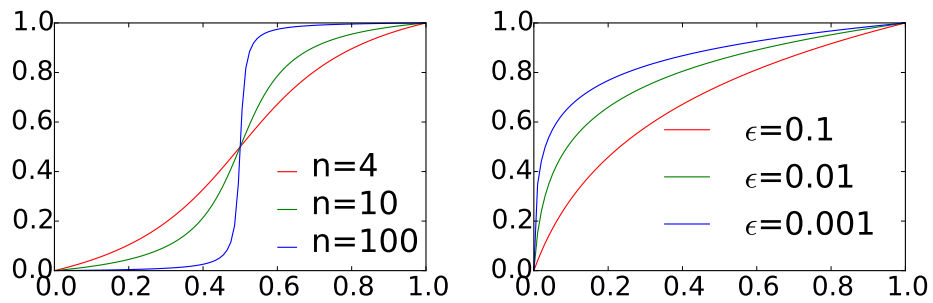


Figure 9.8: Some functions that can be used to improve the contrast of an image.



Figure 9.9: The result after applying  $f_{10}$  and  $g_{0.01}$  to the test image.

In Figure 9.9  $f_{10}$  and  $g_{0.01}$  have been applied to the image in the right part of Figure 9.6. Since the image was quite well balanced,  $f_{10}$  made the dark areas too dark and the bright areas too bright.  $g_{0.01}$  on the other hand has made the image as a whole too bright.

Increasing the contrast is easy to implement. The following function uses the contrast adjusting function from Equation (9.2), with  $\epsilon$  as parameter.

```
def contrastadjust(X,epsilon):
    """
    Assumes that the values are in [0,255]
    """
    X /= 255.
    X += epsilon
    log(X, X)
    X -= log(epsilon)
    X /= (log(1+epsilon)-log(epsilon))
    X *= 255
```

```
def contrastadjust0(X,n):
    """
    Assumes that the values are in [0,255]
    """
    X /= 255.
    X -= 1/2.
    X *= n
    arctan(X, X)
    X /= (2*arctan(n/2.))
    X += 1/2.0
    X *= 255 # Maps the values back to [0,255]
```

This has been used to generate the right image in Figure 9.9.

### Exercise 9.6: Generate black and white images

Black and white images can be generated from greyscale images (with values between 0 and 255) by replacing each pixel value with the one of 0 and 255 which is closest. Use this strategy to generate the black and white image shown in the right part of Figure 9.2.

### Exercise 9.7: Adjust contrast in images

- Write a function `contrastadjust0` which instead uses the function from Equation (9.1) to increase the contrast.  $n$  should be a parameter to the function.
- Generate the left and right images in Figure 9.9 on your own by writing code which uses the two functions `contrastadjust0` and `contrastadjust`.

### Exercise 9.8: Adjust contrast with another function

In this exercise we will look at another function for increasing the contrast of a picture.

- Show that the function  $f : \mathbb{R} \rightarrow \mathbb{R}$  given by

$$f_n(x) = x^n,$$

for all  $n$  maps the interval  $[0, 1] \rightarrow [0, 1]$ , and that  $f'(1) \rightarrow \infty$  as  $n \rightarrow \infty$ .

- The color image `secret.jpg`, shown in Figure 9.10, contains some information that is nearly invisible to the naked eye on most computer monitors. Use the function  $f(x)$ , to reveal the secret message.

**Hint.** You will first need to convert the image to a greyscale image. You can then use the function `contrastadjust` as a starting point for your own program.



Figure 9.10: Secret message.

### 9.3 Filter-based operations on images

The next examples of operations on images we consider will use filters. These examples define what it means to apply a filter to two-dimensional data. We start with the following definition of a computational molecule. This term stems from image processing, and seems at the outset to be unrelated to filters.

**Definition 9.8.** *Computational molecules.*

We say that an operation  $S$  on an image  $X$  is given by the *computational molecule*

$$A = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & a_{-1,-1} & a_{-1,0} & a_{-1,1} & \cdots \\ \cdots & a_{0,-1} & \underline{a_{0,0}} & a_{0,1} & \cdots \\ \cdots & a_{1,-1} & a_{1,0} & a_{1,1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

if we have that

$$(SX)_{i,j} = \sum_{k_1, k_2} a_{k_1, k_2} X_{i-k_1, j-k_2}. \quad (9.3)$$

In the molecule, indices are allowed to be both positive and negative, we underline the element with index  $(0, 0)$  (the center of the molecule), and assume that  $a_{i,j}$  with indices falling outside those listed in the molecule are zero (as for compact filter notation).

In Equation (9.3), it is possible for the indices  $i - k_1$  and  $j - k_2$  to fall outside the legal range for  $X$ . We will solve this case in the same way as we

did for filters, namely that we assume that  $X$  is extended (either periodically or symmetrically) in both directions. The interpretation of a computational molecule is that we place the center of the molecule on a pixel, multiply the pixel and its neighbors by the corresponding weights  $a_{i,j}$  in reverse order, and finally sum up in order to produce the resulting value. This type of operation will turn out to be particularly useful for images. The following result expresses how computational molecules and filters are related. It states that, if we apply one filter to all the columns, and then another filter to all the rows, the end result can be expressed with the help of a computational molecule.

**Theorem 9.9.** *Filtering and computational molecules.*

Let  $S_1$  and  $S_2$  be filters with compact filter notation  $\mathbf{t}_1$  and  $\mathbf{t}_2$ , respectively, and consider the operation  $S$  where  $S_1$  is first applied to the columns in the image, and then  $S_2$  is applied to the rows in the image. Then  $S$  is an operation which can be expressed in terms of the computational molecule  $a_{i,j} = (\mathbf{t}_1)_i(\mathbf{t}_2)_j$ .

*Proof.* Let  $X_{i,j}$  be the pixels in the image. When we apply  $S_1$  to the columns of  $X$  we get the image  $Y$  defined by

$$Y_{i,j} = \sum_{k_1} (t_1)_{k_1} X_{i-k_1,j}.$$

When we apply  $S_2$  to the rows of  $Y$  we get the image  $Z$  defined by

$$\begin{aligned} Z_{i,j} &= \sum_{k_2} (t_2)_{k_2} Y_{i,j-k_2} = \sum_{k_2} (t_2)_{k_2} \sum_{k_1} (t_1)_{k_1} X_{i-k_1,j-k_2} \\ &= \sum_{k_1} \sum_{k_2} (t_1)_{k_1} (t_2)_{k_2} X_{i-k_1,j-k_2}. \end{aligned}$$

Comparing with Equation (9.3) we see that  $S$  is given by the computational molecule with entries  $a_{i,j} = (\mathbf{t}_1)_i(\mathbf{t}_2)_j$ .  $\square$

Note that, when we filter an image with  $S_1$  and  $S_2$  in this way, the order does not matter: since computing  $S_1 X$  is the same as applying  $S_1$  to all columns of  $X$ , and computing  $Y(S_2)^T$  is the same as applying  $S_2$  to all rows of  $Y$ , the combined filtering operation, denoted  $S$ , takes the form

$$S(X) = S_1 X (S_2)^T, \quad (9.4)$$

and the fact that the order does not matter simply boils down to the fact that it does not matter which of the left or right multiplications we perform first. Applying  $S_1$  to the columns of  $X$  is what we call a *vertical filtering operation*, while applying  $S_2$  to the rows of  $X$  is what we call a *horizontal filtering operation*. We can thus state the following.

**Observation 9.10.** *Order of vertical and horizontal filtering.*

The order of vertical and horizontal filtering of an image does not matter.

Most computational molecules we will consider in the following can be expressed in terms of filters as in this theorem, but clearly there exist also computational molecules which are not on this form, since the matrix  $A$  with entries  $a_{i,j} = (\mathbf{t}_1)_i(\mathbf{t}_2)_j$  has rank one, and a general computational molecule can have any rank. In most of the examples the filters are symmetric.

Assume that the image is stored as the matrix  $\mathbf{X}$ . In Exercise 9.13 you will be asked to implement a function `tensor_impl` which computes the transformation  $S(X) = S_1 X (S_2)^T$ , where  $\mathbf{X}$ ,  $\mathbf{S}_1$ , and  $\mathbf{S}_2$  are input. If the computational molecule is obtained by applying the filter  $S_1$  to the columns, and the filter  $S_2$  to the rows, we can compute it with the following code: (we have assumed that the filter lengths are odd, and that the middle filter coefficient has index 0):

```
def S1(x):
    filterS(S1, x, True)

def S2(x):
    filterS(S2, x, True)

tensor_impl(X, S1, S2)
```

We have here used the function `filterS` to implement the filtering, so that we assume that the image is periodically or symmetrically extended. The above code uses symmetric extension, and can thus be used for symmetric filters. If the filter is non-symmetric, we should use a periodic extension instead, for which the last parameter to `filterS` should be changed.

### 9.3.1 Tensor product notation for operations on images

Filter-based operations on images can be written compactly using what we will call *tensor product notation*. This is part of a very general tensor product framework, and we will review parts of this framework for the sake of completeness. Let us first define the tensor product of vectors.

**Definition 9.11.** *Tensor product of vectors.*

If  $\mathbf{x}, \mathbf{y}$  are vectors of length  $M$  and  $N$ , respectively, their tensor product  $\mathbf{x} \otimes \mathbf{y}$  is defined as the  $M \times N$ -matrix defined by  $(\mathbf{x} \otimes \mathbf{y})_{i,j} = x_i y_j$ . In other words,  $\mathbf{x} \otimes \mathbf{y} = \mathbf{x} \mathbf{y}^T$ .

The tensor product  $\mathbf{x} \mathbf{y}^T$  is also called the *outer product* of  $\mathbf{x}$  and  $\mathbf{y}$  (contrary to the inner product  $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y}$ ). In particular  $\mathbf{x} \otimes \mathbf{y}$  is a matrix of rank 1, which means that most matrices cannot be written as a tensor product of two vectors. The special case  $\mathbf{e}_i \otimes \mathbf{e}_j$  is the matrix which is 1 at  $(i, j)$  and 0 elsewhere, and the set of all such matrices forms a basis for the set of  $M \times N$ -matrices.

**Observation 9.12.** *Standard basis for  $L_{M,N}(\mathbb{R})$ .*

Let  $\mathcal{E}_M = \{\mathbf{e}_i\}_{i=0}^{M-1}$   $\mathcal{E}_N = \{\mathbf{e}_j\}_{j=0}^{N-1}$  be the standard bases for  $\mathbb{R}^M$  and  $\mathbb{R}^N$ . Then

$$\mathcal{E}_{M,N} = \{\mathbf{e}_i \otimes \mathbf{e}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$$

is a basis for  $L_{M,N}(\mathbb{R})$ , the set of  $M \times N$ -matrices. This basis is often referred to as the standard basis for  $L_{M,N}(\mathbb{R})$ .

The standard basis thus consists of rank 1-matrices. An image can simply be thought of as a matrix in  $L_{M,N}(\mathbb{R})$ , and a computational molecule is simply a special type of linear transformation from  $L_{M,N}(\mathbb{R})$  to itself. Let us also define the tensor product of matrices.

**Definition 9.13.** *Tensor product of matrices.*

If  $S_1 : \mathbb{R}^M \rightarrow \mathbb{R}^M$  and  $S_2 : \mathbb{R}^N \rightarrow \mathbb{R}^N$  are matrices, we define the linear mapping  $S_1 \otimes S_2 : L_{M,N}(\mathbb{R}) \rightarrow L_{M,N}(\mathbb{R})$  by linear extension of  $(S_1 \otimes S_2)(\mathbf{e}_i \otimes \mathbf{e}_j) = (S_1 \mathbf{e}_i) \otimes (S_2 \mathbf{e}_j)$ . The linear mapping  $S_1 \otimes S_2$  is called the tensor product of the matrices  $S_1$  and  $S_2$ .

A couple of remarks are in order. First, from linear algebra we know that, when  $S$  is linear mapping from  $V$  and  $S(\mathbf{v}_i)$  is known for a basis  $\{\mathbf{v}_i\}_i$  of  $V$ ,  $S$  is uniquely determined. In particular, since the  $\{\mathbf{e}_i \otimes \mathbf{e}_j\}_{i,j}$  form a basis, there exists a unique linear transformation  $S_1 \otimes S_2$  so that  $(S_1 \otimes S_2)(\mathbf{e}_i \otimes \mathbf{e}_j) = (S_1 \mathbf{e}_i) \otimes (S_2 \mathbf{e}_j)$ . This unique linear transformation is what we call the linear extension from the values in the given basis. Clearly, by linearity, also  $(S_1 \otimes S_2)(\mathbf{x} \otimes \mathbf{y}) = (S_1 \mathbf{x}) \otimes (S_2 \mathbf{y})$ , since

$$\begin{aligned} (S_1 \otimes S_2)(\mathbf{x} \otimes \mathbf{y}) &= (S_1 \otimes S_2)\left(\left(\sum_i x_i \mathbf{e}_i\right) \otimes \left(\sum_j y_j \mathbf{e}_j\right)\right) = (S_1 \otimes S_2)\left(\sum_{i,j} x_i y_j (\mathbf{e}_i \otimes \mathbf{e}_j)\right) \\ &= \sum_{i,j} x_i y_j (S_1 \otimes S_2)(\mathbf{e}_i \otimes \mathbf{e}_j) = \sum_{i,j} x_i y_j (S_1 \mathbf{e}_i) \otimes (S_2 \mathbf{e}_j) \\ &= \sum_{i,j} x_i y_j S_1 \mathbf{e}_i ((S_2 \mathbf{e}_j))^T = S_1 \left(\sum_i x_i \mathbf{e}_i\right) (S_2 \left(\sum_j y_j \mathbf{e}_j\right))^T \\ &= S_1 \mathbf{x} (S_2 \mathbf{y})^T = (S_1 \mathbf{x}) \otimes (S_2 \mathbf{y}). \end{aligned}$$

Here we used the result from Exercise 9.17. We can now prove the following.

**Theorem 9.14.** *Compact filter notation and computational molecules.*

If  $S_1 : \mathbb{R}^M \rightarrow \mathbb{R}^M$  and  $S_2 : \mathbb{R}^N \rightarrow \mathbb{R}^N$  are matrices of linear transformations, then  $(S_1 \otimes S_2)X = S_1 X (S_2)^T$  for any  $X \in L_{M,N}(\mathbb{R})$ . In particular  $S_1 \otimes S_2$  is the operation which applies  $S_1$  to the columns of  $X$ , and  $S_2$  to the resulting rows. In other words, if  $S_1, S_2$  have compact filter notations  $\mathbf{t}_1$  and  $\mathbf{t}_2$ , respectively, then  $S_1 \otimes S_2$  has computational molecule  $\mathbf{t}_1 \otimes \mathbf{t}_2$ .

We have not formally defined the tensor product of compact filter notations. This is a straightforward extension of the usual tensor product of vectors, where we additionally mark the element at index  $(0, 0)$ .

*Proof.* We have that

$$\begin{aligned} (S_1 \otimes S_2)(\mathbf{e}_i \otimes \mathbf{e}_j) &= (S_1 \mathbf{e}_i) \otimes (S_2 \mathbf{e}_j) = S_1 \mathbf{e}_i ((S_2 \mathbf{e}_j))^T \\ &= S_1 \mathbf{e}_i (\mathbf{e}_j)^T (S_2)^T = S_1 (\mathbf{e}_i \otimes \mathbf{e}_j) (S_2)^T. \end{aligned}$$

This means that  $(S_1 \otimes S_2)X = S_1X(S_2)^T$  for any  $X \in L_{M,N}(\mathbb{R})$  also, since equality holds on the basis vectors  $e_i \otimes e_j$ . Since the matrix  $A$  with entries  $a_{i,j} = (t_1)_i(t_2)_j$  also can be written as  $t_1 \otimes t_2$ , the result follows.  $\square$

We have thus shown that we alternatively can write  $S_1 \otimes S_2$  for the operations we have considered. This notation also makes it easy to combine several two-dimensional filtering operations:

**Corollary 9.15.** *Composing tensor products.*

We have that  $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_1S_2) \otimes (T_1T_2)$ .

*Proof.* By Theorem 9.14 we have that

$$(S_1 \otimes T_1)(S_2 \otimes T_2)X = S_1(S_2XT_2^T)T_1^T = (S_1S_2)X(T_1T_2)^T = ((S_1S_2) \otimes (T_1T_2))X.$$

for any  $X \in L_{M,N}(\mathbb{R})$ . This proves the result.  $\square$

Suppose that we want to apply the operation  $S_1 \otimes S_2$  to an image. We can factorize  $S_1 \otimes S_2$  as

$$S_1 \otimes S_2 = (S_1 \otimes I)(I \otimes S_2) = (I \otimes S_2)(S_1 \otimes I). \quad (9.5)$$

Moreover, since

$$(S_1 \otimes I)X = S_1X \quad (I \otimes S_2)X = X(S_2)^T = (S_2X^T)^T,$$

$S_1 \otimes I$  is a vertical filtering operation, and  $I \otimes S_2$  is a horizontal filtering operation in this factorization. For filters we have an even stronger result: If  $S_1, S_2, S_3, S_4$  all are filters, we have from Corollary 9.15 that  $(S_1 \otimes S_2)(S_3 \otimes S_4) = (S_3 \otimes S_4)(S_1 \otimes S_2)$ , since all filters commute. This does not hold in general since general matrices do not commute.

We will now consider two important examples of filtering operations on images: smoothing and edge detection/computing partial derivatives. For all examples we will use the tensor product notation for these operations.

### Example 9.9: Smoothing an image

When we considered filtering of digital sound, low-pass filters dampened high frequencies. We will here similarly see that an image can be smoothed by applying a low-pass filters to the rows and the columns. Let us consider such computational molecules. In particular, let us as before take filter coefficients taken from Pascal's triangle. If we use the filter  $S = \frac{1}{4}\{1, 2, 1\}$  (row 2 from Pascal's triangle), Theorem 9.9 gives the computational molecule

$$A = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}. \quad (9.6)$$

If the pixels in the image are  $p_{i,j}$ , this means that we compute the new pixels by

$$\hat{p}_{i,j} = \frac{1}{16} (4p_{i,j} + 2(p_{i,j-1} + p_{i-1,j} + p_{i+1,j} + p_{i,j+1}) + p_{i-1,j-1} + p_{i+1,j-1} + p_{i-1,j+1} + p_{i+1,j+1}).$$

If we instead use the filter  $S = \frac{1}{64}\{1, 6, 15, \underline{20}, 15, 6, 1\}$  (row 6 from Pascal's triangle), we get the computational molecule

$$\frac{1}{4096} \begin{pmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 20 & 120 & 300 & \underline{400} & 300 & 120 & 20 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{pmatrix}. \quad (9.7)$$

We anticipate that both molecules give a smoothing effect, but that the second molecule provides more smoothing. The result of applying the two molecules in (9.6) and (9.7) to our greyscale-image is shown in the two right images in Figure 9.11. With the help of the function `tensor_impl`, smoothing with the first molecule (9.6) above can be obtained by writing

```
def S(x):
    filterS([1., 2., 1.]/4., x, True);
    tensor_impl(X, S, S)
```

To make the smoothing effect visible, we have zoomed in on the face in the image. The smoothing effect is clearly best visible in the second image.

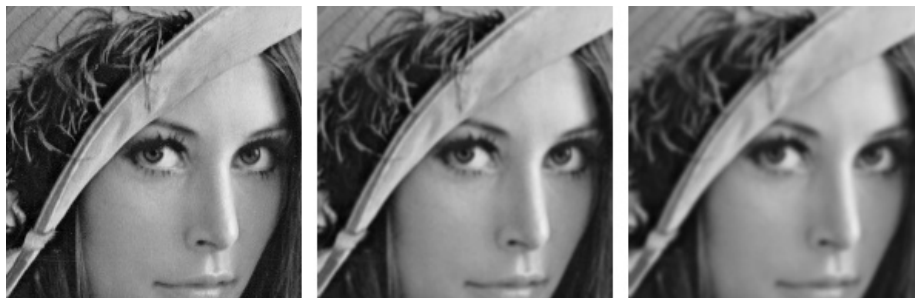


Figure 9.11: The two right images show the effect of smoothing the left image.

Smoothing effects are perhaps more visible if we use a simple image, as the one in the left part of Figure 9.12.

Again we have used the filter  $S = \frac{1}{4}\{1, \underline{2}, 1\}$ . Here we also have shown what happens if we only smooth the image in one of the directions. In the right



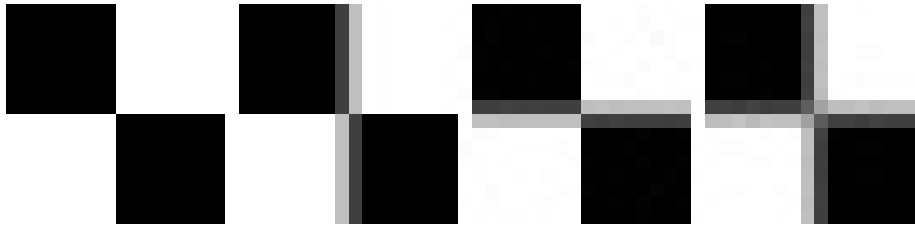


Figure 9.12: The results of smoothing the simple image to the left in three different ways.

image we have smoothed in both directions. We clearly see the union of the two one-dimensional smoothing operations then.

### Example 9.10: Edge detection

Another operation on images which can be expressed in terms of computational molecules is *edge detection*. An edge in an image is characterised by a large change in intensity values over a small distance in the image. For a continuous function this corresponds to a large derivative. An image is only defined at isolated points, so we cannot compute derivatives, but since a grey-level image is a scalar function of two variables, we have a perfect situation for applying numerical differentiation techniques.

**Partial derivative in  $x$ -direction.** Let us first consider computation of the partial derivative  $\partial P/\partial x$  at all points in the image. Note first that it is the second coordinate in an image which refers to the  $x$ -direction used when plotting functions. This means that the familiar symmetric Newton quotient approximation for the partial derivative [31] takes the form

$$\frac{\partial P}{\partial x}(i, j) \approx \frac{p_{i, j+1} - p_{i, j-1}}{2}, \quad (9.8)$$

where we have used the convention  $h = 1$  which means that the derivative is measured in terms of 'intensity per pixel'. This corresponds to applying the bias-reducing filter  $S = \frac{1}{2}\{1, 0, -1\}$  to all the rows (alternatively, applying the tensor product  $I \otimes S$  to the image). We can thus express this in terms of the computational molecule.

$$\frac{1}{2} \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix}.$$

We have included the two rows of 0s just to make it clear how the computational molecule is to be interpreted when we place it over the pixels. Let us first apply this molecule to the usual excerpt of the Lena image. This gives the first image in Figure 9.13. This images shows many artefacts since the pixel values

lie outside the legal range: many of the intensities are in fact negative. More specifically, the intensities turn out to vary in the interval  $[-0.424, 0.418]$ . Let us therefore normalise and map all intensities to  $[0, 1]$ . This gives the second image in Figure 9.13. The predominant color of this image is an average grey, i.e. an intensity of about 0.5. To get more detail in the image we therefore try to increase the contrast by applying the function  $f_{50}$  in equation (9.1) to each intensity value. The result is shown in the third image in Figure 9.13. This does indeed show more detail.

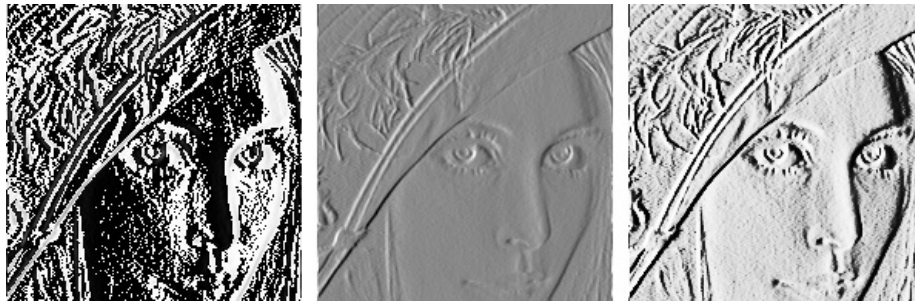


Figure 9.13: Experimenting with the partial derivative in the  $x$ -direction for the image in 9.6. The left image has artefacts, since the pixel values are outside the legal range. We therefore normalise the intensities to lie in  $[0, 255]$  (middle), before we increase the contrast (right).

It is important to understand the colors in these images. We have computed the derivative in the  $x$ -direction, and we recall that the computed values varied in the interval  $[-0.424, 0.418]$ . The negative value corresponds to the largest average decrease in intensity from a pixel  $p_{i-1,j}$  to a pixel  $p_{i+1,j}$ . The positive value on the other hand corresponds to the largest average increase in intensity. A value of 0 in the left image in Figure 9.13 corresponds to no change in intensity between the two pixels.

When the values are mapped to the interval  $[0, 1]$  in the second image, the small values are mapped to something close to 0 (almost black), the maximal values are mapped to something close to 1 (almost white), and the values near 0 are mapped to something close to 0.5 (grey). In the third image these values have just been emphasised even more.

The third image tells us that in large parts of the image there is very little variation in the intensity. However, there are some small areas where the intensity changes quite abruptly, and if you look carefully you will notice that in these areas there is typically both black and white pixels close together, like down the vertical front corner of the bus. This will happen when there is a stripe of bright or dark pixels that cut through an area of otherwise quite uniform intensity.

**Partial derivative in  $y$ -direction.** The partial derivative  $\partial P/\partial y$  can be computed analogously to  $\partial P/\partial x$ , i.e. we apply the filter  $-S = \frac{1}{2}\{-1, \underline{0}, 1\}$  to

all columns of the image (alternatively, apply the tensor product  $-S \otimes I$  to the image), where  $S$  is the filter which we used for edge detection in the  $x$ -direction. Note that the positive direction of this axis in an image is opposite to the direction of the  $y$ -axis we use when plotting functions. We can express this in terms of the computational molecule

$$\frac{1}{2} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}.$$

Let us compare the partial derivatives in both directions. The result is shown in Figure 9.14.



Figure 9.14: The first-order partial derivatives in the  $x$ - and  $y$ -direction, respectively. In both images, the computed numbers have been normalised and the contrast enhanced.

The intensities have been normalised and the contrast enhanced by the function  $f_{50}$  from Equation (9.1).

**The gradient.** The gradient of a scalar function is often used as a measure of the size of the first derivative. The gradient is defined by the vector

$$\nabla P = \left( \frac{\partial P}{\partial x}, \frac{\partial P}{\partial y} \right),$$

so its length is given by

$$|\nabla P| = \sqrt{\left( \frac{\partial P}{\partial x} \right)^2 + \left( \frac{\partial P}{\partial y} \right)^2}.$$

When the two first derivatives have been computed it is a simple matter to compute the gradient vector and its length. Note that, as for the first order derivatives, it is possible for the length of the gradient to be outside the legal range of values. The computed gradient values, the gradient mapped to the legal range of values, and the gradient with contrast adjusted, are shown in Figure 9.15.



Figure 9.15: The computed gradient (left). In the middle the intensities have been normalised to the  $[0, 255]$ , and to the right the contrast has been increased.

The image of the gradient looks quite different from the images of the two partial derivatives. The reason is that the numbers that represent the length of the gradient are (square roots of) sums of squares of numbers. This means that the parts of the image that have virtually constant intensity (partial derivatives close to 0) are colored black. In the images of the partial derivatives these values ended up in the middle of the range of intensity values, with a final color of grey, since there were both positive and negative values. To enhance the contrast for this image we should thus do something different from what was done in the other images, since we now have a large number of intensities near 0. The solution was to apply a function like the ones shown in the right plot in Figure 9.8. Here we have used the function  $g_{0.01}$ .

Figure 9.14 shows the two first-order partial derivatives and the gradient. If we compare the two partial derivatives we see that the  $x$ -derivative seems to emphasise vertical edges while the  $y$ -derivative seems to emphasise horizontal edges. This is precisely what we must expect. The  $x$ -derivative is large when the difference between neighbouring pixels in the  $x$ -direction is large, which is the case across a vertical edge. The  $y$ -derivative enhances horizontal edges for a similar reason.

The gradient contains information about both derivatives and therefore emphasises edges in all directions. It also gives a simpler image since the sign of the derivatives has been removed.

### Example 9.11: Second-order derivatives

To compute the three second order derivatives we can combine the two computational molecules which we already have described. For the mixed second

order derivative we get  $(I \otimes S)((-S) \otimes I) = -S \otimes S$ . For the last two second order derivative  $\frac{\partial^2 P}{\partial x^2}$ ,  $\frac{\partial^2 P}{\partial y^2}$ , we can also use the three point approximation to the second derivative [31]

$$\frac{\partial P}{\partial x^2}(i, j) \approx p_{i,j+1} - 2p_{i,j} + p_{i,j-1} \quad (9.9)$$

(again we have set  $h = 1$ ). This gives a smaller molecule than if we combine the two molecules for order one differentiation (i.e.  $(I \otimes S)(I \otimes S) = (I \otimes S^2)$  and  $((-S) \otimes I)((-S) \otimes I) = (S^2 \otimes I)$ ), since  $S^2 = \frac{1}{2}\{1, 0, -1\}\frac{1}{2}\{1, 0, -1\} = \frac{1}{4}\{1, 0, -2, 0, 1\}$ . The second order derivatives of an image  $P$  can thus be computed by applying the computational molecules

$$\frac{\partial^2 P}{\partial x^2} : \begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \quad (9.10)$$

$$\frac{\partial^2 P}{\partial y \partial x} : \frac{1}{4} \begin{pmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{pmatrix}, \quad (9.11)$$

$$\frac{\partial^2 P}{\partial y^2} : \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \quad (9.12)$$

With these molecules it is quite easy to compute the second-order derivatives. The results are shown in Figure 9.16.



Figure 9.16: The second-order partial derivatives in the  $xx$ -,  $xy$ -, and  $yy$ -directions, respectively. In all images, the computed numbers have been normalised and the contrast enhanced.

The computed derivatives were first normalised and then the contrast enhanced with the function  $f_{100}$  in each image, see equation (9.1).

As for the first derivatives, the  $xx$ -derivative seems to emphasise vertical edges and the  $yy$ -derivative horizontal edges. However, we also see that the second derivatives are more sensitive to noise in the image (the areas of grey are

less uniform). The mixed derivative behaves a bit differently from the other two, and not surprisingly it seems to pick up both horizontal and vertical edges.

This procedure can be generalized to higher order derivatives also. To apply  $\frac{\partial^{k+l} P}{\partial x^k \partial y^l}$  to an image we can compute  $S_l \otimes S_k$  where  $S_r$  corresponds to any point method for computing the  $r$ 'th order derivative. We can also compute  $(S^l) \otimes (S^k)$ , where we iterate the filter  $S = \frac{1}{2}\{1, \underline{0}, -1\}$  for the first derivative, but this gives longer filters.

### Example 9.12: Chess pattern image

Let us apply the molecules for differentiation to a chess pattern test image. In Figure 9.17 we have applied  $S \otimes I$ ,  $I \otimes S$ , and  $S \otimes S$ ,  $I \otimes S^2$ , and  $S^2 \otimes I$  to the example image shown in the upper left.

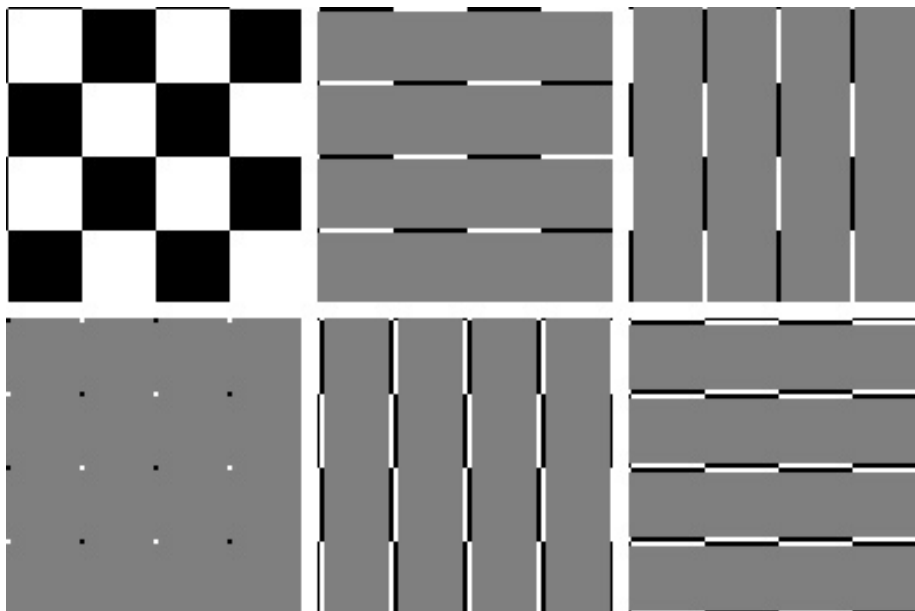


Figure 9.17: Different tensor products applied to the simple chess pattern image shown in the upper left.

These images make it is clear that  $S \otimes I$  detects all horizontal edges,  $I \otimes S$  detects all vertical edges, and that  $S \otimes S$  detects all points where abrupt changes appear in both directions. We also see that the second order partial derivative detects exactly the same edges which the first order partial derivative found. Note that the edges detected with  $I \otimes S^2$  are wider than the ones detected with  $I \otimes S$ . The reason is that the filter  $S^2$  has more filter coefficients than  $S$ . Also, edges are detected with different colors. This reflects whether the difference between the neighbouring pixels is positive or negative. The values after we have applied the tensor product may thus not lie in the legal range

of pixel values (since they may be negative). The figures have taken this into account by mapping the values back to a legal range of values, as we did in Chapter 9. Finally, we also see additional edges at the first and last rows/edges in the images. The reason is that the filter  $S$  is defined by assuming that the pixels repeat periodically (i.e. it is a circulant Toeplitz matrix). Due to this, we have additional edges at the first/last rows/edges. This effect can also be seen in Chapter 9, although there we did not assume that the pixels repeat periodically.

Defining a two-dimensional filter by filtering columns and then rows is not the only way we can define a two-dimensional filter. Another possible way is to let the  $MN \times MN$ -matrix itself be a filter. Unfortunately, this is a bad way to define filtering of an image, since there are some undesirable effects near the boundaries between rows: in the vector we form, the last element of one row is followed by the first element of the next row. These boundary effects are unfortunate when a filter is applied.

### Exercise 9.13: Implement a tensor product

Implement a function `tensor_impl` which takes a matrix  $X$ , and functions  $S1$  and  $S2$  as parameters, and applies  $S1$  to the columns of  $X$ , and  $S2$  to the rows of  $X$ .

### Exercise 9.14: Generate images

Write code which calls the function `tensor_impl` with appropriate filters and which generate the following images:

- a) The right image in Figure 9.11.
- b) The right image in Figure 9.13.
- c) The images in Figure 9.15.
- d) The images in Figure 9.14.
- e) The images in Figure 9.16.

### Exercise 9.15: Interpret tensor products

Let the filter  $S$  be defined by  $S = \{\underline{-1}, 1\}$ .

- a) Let  $X$  be a matrix which represents the pixel values in an image. What can you say about how the new images  $(S \otimes I)X$  og  $(I \otimes S)X$  look? What are the interpretations of these operations?
- b) Write down the  $4 \otimes 4$ -matrix  $X = (1, 1, 1, 1) \otimes (0, 0, 1, 1)$ . Compute  $(S \otimes I)X$  by applying the filters to the corresponding rows/columns of  $X$  as we have learned, and interpret the result. Do the same for  $(I \otimes S)X$ .



**Exercise 9.16: Computational molecule of moving average filter**

Let  $S$  be the moving average filter of length  $2L+1$ , i.e.  $T = \frac{1}{L} \underbrace{\{1, \dots, 1, \underline{1}, 1, \dots, 1\}}_{2L+1 \text{ times}}$ .

What is the computational molecule of  $S \otimes S$ ?

**Exercise 9.17: Bilinearity of the tensor product**

Show that the mapping  $F(\mathbf{x}, \mathbf{y}) = \mathbf{x} \otimes \mathbf{y}$  is bi-linear, i.e. that  $F(\alpha\mathbf{x}_1 + \beta\mathbf{x}_2, \mathbf{y}) = \alpha F(\mathbf{x}_1, \mathbf{y}) + \beta F(\mathbf{x}_2, \mathbf{y})$ , and  $F(\mathbf{x}, \alpha\mathbf{y}_1 + \beta\mathbf{y}_2) = \alpha F(\mathbf{x}, \mathbf{y}_1) + \beta F(\mathbf{x}, \mathbf{y}_2)$ .

**Exercise 9.18: Attempt to write as tensor product**

Attempt to find matrices  $S_1 : \mathbb{R}^M \rightarrow \mathbb{R}^M$  and  $S_2 : \mathbb{R}^N \rightarrow \mathbb{R}^N$  so that the following mappings from  $L_{M,N}(\mathbb{R})$  to  $L_{M,N}(\mathbb{R})$  can be written on the form  $X \rightarrow S_1 X (S_2)^T = (S_1 \otimes S_2) X$ . In all the cases, it may be that no such  $S_1, S_2$  can be found. If this is the case, prove it.

- The mapping which reverses the order of the rows in a matrix.
- The mapping which reverses the order of the columns in a matrix.
- The mapping which transposes a matrix.

**Exercise 9.19: Computational molecules**

Let the filter  $S$  be defined by  $S = \{1, \underline{2}, 1\}$ .

- Write down the computational molecule of  $S \otimes S$ .
- Let us define  $\mathbf{x} = (1, 2, 3)$ ,  $\mathbf{y} = (3, 2, 1)$ ,  $\mathbf{z} = (2, 2, 2)$ , and  $\mathbf{w} = (1, 4, 2)$ . Compute the matrix  $A = \mathbf{x} \otimes \mathbf{y} + \mathbf{z} \otimes \mathbf{w}$ .
- Compute  $(S \otimes S)A$  by applying the filter  $S$  to every row and column in the matrix the way we have learned. If the matrix  $A$  was more generally an image, what can you say about how the new image will look?

**Exercise 9.20: Computational molecules 2**

Let  $S = \frac{1}{4}\{1, \underline{2}, 1\}$  be a filter.

- What is the effect of applying the tensor products  $S \otimes I$ ,  $I \otimes S$ , and  $S \otimes S$  on an image represented by the matrix  $X$ ?
- Compute  $(S \otimes S)(\mathbf{x} \otimes \mathbf{y})$ , where  $\mathbf{x} = (4, 8, 8, 4)$ ,  $\mathbf{y} = (8, 4, 8, 4)$  (i.e. both  $\mathbf{x}$  and  $\mathbf{y}$  are column vectors).



**Exercise 9.21: Comment on code**

Suppose that we have an image given by the  $M \times N$ -matrix  $X$ , and consider the following code:

```

for n in range(N):
    X[0, n] = 0.25*X[N-1, n] + 0.5*X[0, n] + 0.25*X[1, n]
    X[1:(N-1), n] = 0.25*X[0:(N-2), n] + 0.5*X[1:(N-1), n] \
    + 0.25*X[2:N, n]
    X[N-1, n] = 0.25*X[N-2, n] + 0.5*X[N-1, n] + 0.25*X[0, n]
for m in range(m):
    X[m, 0] = 0.25*X[m, M-1] + 0.5*X[m, 0] + 0.25*X[m, 1]
    X[m, 1:(M-1)] = 0.25*X[m, 0:(M-2)] + 0.5*X[m, 1:(M-1)] \
    + 0.25*X[m, 2:M]
    X[m, M-1] = 0.25*X[m, M-2] + 0.5*X[m, M-1] + 0.25*X[m, 0]

```

Which tensor product is applied to the image? Comment what the code does, in particular the first and third line in the inner `for`-loop. What effect does the code have on the image?

**Exercise 9.22: Eigenvectors of tensor products**

Let  $\mathbf{v}_A$  be an eigenvector of  $A$  with eigenvalue  $\lambda_A$ , and  $\mathbf{v}_B$  an eigenvector of  $B$  with eigenvalue  $\lambda_B$ . Show that  $\mathbf{v}_A \otimes \mathbf{v}_B$  is an eigenvector of  $A \otimes B$  with eigenvalue  $\lambda_A \lambda_B$ . Explain from this why  $\|A \otimes B\| = \|A\| \|B\|$ , where  $\|\cdot\|$  denotes the operator norm of a matrix.

**Exercise 9.23: The Kronecker product**

The *Kronecker tensor product* of two matrices  $A$  and  $B$ , written  $A \otimes^k B$ , is defined as

$$A \otimes^k B = \begin{pmatrix} a_{1,1}B & a_{1,2}B & \cdots & a_{1,M}B \\ a_{2,1}B & a_{2,2}B & \cdots & a_{2,M}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{p,1}B & a_{p,2}B & \cdots & a_{p,M}B \end{pmatrix},$$

where the entries of  $A$  are  $a_{i,j}$ . The tensor product of a  $p \times M$ -matrix, and a  $q \times N$ -matrix is thus a  $(pq) \times (MN)$ -matrix. Note that this tensor product in particular gives meaning for vectors: if  $\mathbf{x} \in \mathbb{R}^M$ ,  $\mathbf{y} \in \mathbb{R}^N$  are column vectors, then  $\mathbf{x} \otimes^k \mathbf{y} \in \mathbb{R}^{MN}$  is also a column vector. In this exercise we will investigate how the Kronecker tensor product is related to tensor products as we have defined them in this section.

a) Explain that, if  $\mathbf{x} \in \mathbb{R}^M$ ,  $\mathbf{y} \in \mathbb{R}^N$  are column vectors, then  $\mathbf{x} \otimes^k \mathbf{y}$  is the column vector where the rows of  $\mathbf{x} \otimes \mathbf{y}$  have first been stacked into one large row vector, and this vector transposed. The linear extension of the operation defined by

$$\mathbf{x} \otimes \mathbf{y} \in \mathbb{R}^{M,N} \rightarrow \mathbf{x} \otimes^k \mathbf{y} \in \mathbb{R}^{MN}$$

thus stacks the rows of the input matrix into one large row vector, and transposes the result.

**b)** Show that  $(A \otimes^k B)(\mathbf{x} \otimes^k \mathbf{y}) = (A\mathbf{x}) \otimes^k (B\mathbf{y})$ . We can thus use any of the defined tensor products  $\otimes, \otimes^k$  to produce the same result, i.e. we have the commutative diagram shown in Figure 9.18, where the vertical arrows represent stacking the rows in the matrix, and transposing, and the horizontal arrows represent the two tensor product linear transformations we have defined. In particular, we can compute the tensor product in terms of vectors, or in terms of matrices, and it is clear that the Kronecker tensor product gives the matrix of tensor product operations.

$$\begin{array}{ccc}
 \mathbf{x} \otimes \mathbf{y} & \xrightarrow{A \otimes B} & (A\mathbf{x}) \otimes (B\mathbf{y}) \\
 \downarrow & & \downarrow \\
 \mathbf{x} \otimes^k \mathbf{y} & \xrightarrow{A \otimes^k B} & (A\mathbf{x}) \otimes^k (B\mathbf{y}),
 \end{array}$$

Figure 9.18: Tensor products

**c)** Using the Euclidean inner product on  $L(M, N) = \mathbb{R}^{MN}$ , i.e.

$$\langle X, Y \rangle = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} X_{i,j} \overline{Y_{i,j}}.$$

and the correspondence in a) we can define the inner product of  $\mathbf{x}_1 \otimes \mathbf{y}_1$  and  $\mathbf{x}_2 \otimes \mathbf{y}_2$  by

$$\langle \mathbf{x}_1 \otimes \mathbf{y}_1, \mathbf{x}_2 \otimes \mathbf{y}_2 \rangle = \langle \mathbf{x}_1 \otimes^k \mathbf{y}_1, \mathbf{x}_2 \otimes^k \mathbf{y}_2 \rangle.$$

Show that

$$\langle \mathbf{x}_1 \otimes \mathbf{y}_1, \mathbf{x}_2 \otimes \mathbf{y}_2 \rangle = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle \langle \mathbf{y}_1, \mathbf{y}_2 \rangle.$$

Clearly this extends linearly to an inner product on  $L_{M,N}$ .

**d)** Show that the FFT factorization can be written as

$$\begin{pmatrix} F_{N/2} & D_{N/2} F_{N/2} \\ F_{N/2} & -D_{N/2} F_{N/2} \end{pmatrix} = \begin{pmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{pmatrix} (I_2 \otimes^k F_{N/2}).$$

Also rewrite the sparse matrix factorization for the FFT from Equation (2.18) in terms of tensor products.

## 9.4 Change of coordinates in tensor products

Filter-based operations were not the only operations we considered for sound. We also considered the DFT, the DCT, and the wavelet transform, which were changes of coordinates which gave us useful frequency- or time-frequency information. We would like to define similar changes of coordinates for images, which also give useful such information. Tensor product notation will also be useful in this respect, and we start with the following result.

**Theorem 9.16.** *The basis  $\mathcal{B}_1 \otimes \mathcal{B}_2$ .*

If  $\mathcal{B}_1 = \{\mathbf{v}_i\}_{i=0}^{M-1}$  is a basis for  $\mathbb{R}^M$ , and  $\mathcal{B}_2 = \{\mathbf{w}_j\}_{j=0}^{N-1}$  is a basis for  $\mathbb{R}^N$ , then  $\{\mathbf{v}_i \otimes \mathbf{w}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$  is a basis for  $L_{M,N}(\mathbb{R})$ . We denote this basis by  $\mathcal{B}_1 \otimes \mathcal{B}_2$ .

*Proof.* Suppose that  $\sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\mathbf{v}_i \otimes \mathbf{w}_j) = \mathbf{0}$ . Setting  $\mathbf{h}_i = \sum_{j=0}^{N-1} \alpha_{i,j} \mathbf{w}_j$  we get

$$\sum_{j=0}^{N-1} \alpha_{i,j}(\mathbf{v}_i \otimes \mathbf{w}_j) = \mathbf{v}_i \otimes \left( \sum_{j=0}^{N-1} \alpha_{i,j} \mathbf{w}_j \right) = \mathbf{v}_i \otimes \mathbf{h}_i.$$

where we have used the bi-linearity of the tensor product mapping  $(\mathbf{x}, \mathbf{y}) \rightarrow \mathbf{x} \otimes \mathbf{y}$  (Exercise 9.17). This means that

$$\mathbf{0} = \sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\mathbf{v}_i \otimes \mathbf{w}_j) = \sum_{i=0}^{M-1} \mathbf{v}_i \otimes \mathbf{h}_i = \sum_{i=0}^{M-1} \mathbf{v}_i \mathbf{h}_i^T.$$

Column  $k$  in this matrix equation says  $\mathbf{0} = \sum_{i=0}^{M-1} h_{i,k} \mathbf{v}_i$ , where  $h_{i,k}$  are the components in  $\mathbf{h}_i$ . By linear independence of the  $\mathbf{v}_i$  we must have that  $h_{0,k} = h_{1,k} = \cdots = h_{M-1,k} = 0$ . Since this applies for all  $k$ , we must have that all  $\mathbf{h}_i = \mathbf{0}$ . This means that  $\sum_{j=0}^{N-1} \alpha_{i,j} \mathbf{w}_j = \mathbf{0}$  for all  $i$ , from which it follows by linear independence of the  $\mathbf{w}_j$  that  $\alpha_{i,j} = 0$  for all  $j$ , and for all  $i$ . This means that  $\mathcal{B}_1 \otimes \mathcal{B}_2$  is a basis.  $\square$

In particular, as we have already seen, the standard basis for  $L_{M,N}(\mathbb{R})$  can be written  $\mathcal{E}_{M,N} = \mathcal{E}_M \otimes \mathcal{E}_N$ . This is the basis for a useful convention: For a tensor product the bases are most naturally indexed in two dimensions, rather than the usual sequential indexing. This difference translates also to the meaning of coordinate vectors, which now are more naturally thought of as coordinate matrices:

**Definition 9.17.** *Coordinate matrix.*

Let  $\mathcal{B} = \{\mathbf{b}_i\}_{i=0}^{M-1}$ ,  $\mathcal{C} = \{\mathbf{c}_j\}_{j=0}^{N-1}$  be bases for  $\mathbb{R}^M$  and  $\mathbb{R}^N$ , and let  $A \in L_{M,N}(\mathbb{R})$ . By the coordinate matrix of  $A$  in  $\mathcal{B} \otimes \mathcal{C}$  we mean the  $M \times N$ -matrix  $X$  (with components  $X_{kl}$ ) such that  $A = \sum_{k,l} X_{k,l}(\mathbf{b}_k \otimes \mathbf{c}_l)$ .

We will have use for the following theorem, which shows how change of coordinates in  $\mathbb{R}^M$  and  $\mathbb{R}^N$  translate to a change of coordinates in the tensor product:

**Theorem 9.18.** *Change of coordinates in tensor products.*

Assume that

- $\mathcal{B}_1, \mathcal{C}_1$  are bases for  $\mathbb{R}^M$ , and that  $S_1$  is the change of coordinates matrix from  $\mathcal{B}_1$  to  $\mathcal{C}_1$ ,
- $\mathcal{B}_2, \mathcal{C}_2$  are bases for  $\mathbb{R}^N$ , and that  $S_2$  is the change of coordinates matrix from  $\mathcal{B}_2$  to  $\mathcal{C}_2$ .

Both  $\mathcal{B}_1 \otimes \mathcal{B}_2$  and  $\mathcal{C}_1 \otimes \mathcal{C}_2$  are bases for  $L_{M,N}(\mathbb{R})$ , and if  $X$  is the coordinate matrix in  $\mathcal{B}_1 \otimes \mathcal{B}_2$ , and  $Y$  the coordinate matrix in  $\mathcal{C}_1 \otimes \mathcal{C}_2$ , then the change of coordinates from  $\mathcal{B}_1 \otimes \mathcal{B}_2$  to  $\mathcal{C}_1 \otimes \mathcal{C}_2$  can be computed as

$$Y = S_1 X (S_2)^T. \quad (9.13)$$

*Proof.* Denote the change of coordinates from  $\mathcal{B}_1 \otimes \mathcal{B}_2$  to  $\mathcal{C}_1 \otimes \mathcal{C}_2$  by  $S$ . Since any change of coordinates is linear, it is enough to show that  $S(\mathbf{e}_i \otimes \mathbf{e}_j) = S_1(\mathbf{e}_i \otimes \mathbf{e}_j)(S_2)^T$  for any  $i, j$ . We can write

$$\begin{aligned} \mathbf{b}_{1i} \otimes \mathbf{b}_{2j} &= \left( \sum_k (S_1)_{k,i} \mathbf{c}_{1k} \right) \otimes \left( \sum_l (S_2)_{l,j} \mathbf{c}_{2l} \right) = \sum_{k,l} (S_1)_{k,i} (S_2)_{l,j} (\mathbf{c}_{1k} \otimes \mathbf{c}_{2l}) \\ &= \sum_{k,l} (S_1)_{k,i} ((S_2)^T)_{j,l} (\mathbf{c}_{1k} \otimes \mathbf{c}_{2l}) = \sum_{k,l} (S_1 \mathbf{e}_i (\mathbf{e}_j)^T (S_2)^T)_{k,l} (\mathbf{c}_{1k} \otimes \mathbf{c}_{2l}) \\ &= \sum_{k,l} (S_1 (\mathbf{e}_i \otimes \mathbf{e}_j) (S_2)^T)_{k,l} (\mathbf{c}_{1k} \otimes \mathbf{c}_{2l}) \end{aligned}$$

This shows that the coordinate matrix of  $\mathbf{b}_{1i} \otimes \mathbf{b}_{2j}$  in  $\mathcal{C}_1 \otimes \mathcal{C}_2$  is  $S_1(\mathbf{e}_i \otimes \mathbf{e}_j)(S_2)^T$ . Since the coordinate matrix of  $\mathbf{b}_{1i} \otimes \mathbf{b}_{2j}$  in  $\mathcal{B}_1 \otimes \mathcal{B}_2$  is  $\mathbf{e}_i \otimes \mathbf{e}_j$ , this shows that  $S(\mathbf{e}_i \otimes \mathbf{e}_j) = S_1(\mathbf{e}_i \otimes \mathbf{e}_j)(S_2)^T$ . The result follows.  $\square$

In both cases of filtering and change of coordinates in tensor products, we see that we need to compute the mapping  $X \rightarrow S_1 X (S_2)^T$ . As we have seen, this amounts to a row/column-wise operation, which we restate as follows:

**Observation 9.19.** *Change of coordinates in tensor products.*

The change of coordinates from  $\mathcal{B}_1 \otimes \mathcal{B}_2$  to  $\mathcal{C}_1 \otimes \mathcal{C}_2$  can be implemented as follows:

- For every column in the coordinate matrix in  $\mathcal{B}_1 \otimes \mathcal{B}_2$ , perform a change of coordinates from  $\mathcal{B}_1$  to  $\mathcal{C}_1$ .
- For every row in the resulting matrix, perform a change of coordinates from  $\mathcal{B}_2$  to  $\mathcal{C}_2$ .

We can again use the function `tensor_impl` in order to implement change of coordinates for a tensor product. We just need to replace the filters with the functions `S1` and `S2` for computing the corresponding changes of coordinates:

```
tensor_impl(X, S1, S2)
```

The operation  $X \rightarrow (S_1)X(S_2)^T$ , which we now have encountered in two different ways, is one particular type of linear transformation from  $\mathbb{R}^{N^2}$  to itself (see Exercise 9.23 for how the matrix of this linear transformation can be constructed). While a general such linear transformation requires  $N^4$  multiplications (i.e. when we perform a full matrix multiplication),  $X \rightarrow (S_1)X(S_2)^T$  can be implemented generally with only  $2N^3$  multiplications (since multiplication of two  $N \times N$ -matrices require  $N^3$  multiplications in general). The operation  $X \rightarrow (S_1)X(S_2)^T$  is thus computationally simpler than linear transformations in general. In practice the operations  $S_1$  and  $S_2$  are also computationally simpler, since they can be filters, FFT's, or wavelet transformations, so that the complexity in  $X \rightarrow (S_1)X(S_2)^T$  can be even lower.

In the following examples, we will interpret the pixel values in an image as coordinates in the standard basis, and perform a change of coordinates.

### Example 9.24: Change of coordinates with the DFT

The DFT is one particular change of coordinates which we have considered. It was the change of coordinates from the standard basis to the Fourier basis. A corresponding change of coordinates in a tensor product is obtained by substituting the DFT as the functions  $S_1, S_2$  for implementing the changes of coordinates above. The change of coordinates in the opposite direction is obtained by using the IDFT instead of the DFT.

Modern image standards do typically not apply a change of coordinates to the entire image. Rather the image is split into smaller squares of appropriate size, called blocks, and a change of coordinates is performed independently for each block. In this example we have split the image into blocks of size  $8 \times 8$ .

Recall that the DFT values express frequency components. The same applies for the two-dimensional DFT and thus for images, but frequencies are now represented in two different directions. Let us introduce a neglection threshold in the same way as in Example 2.17, to view the image after we set certain frequencies to zero. As for sound, this has little effect on the human perception of the image, if we use a suitable neglection threshold. After we have performed the two-dimensional DFT on an image, we can neglect DFT-coefficients below a threshold on the resulting matrix  $X$  with the following code:

```
X **= (abs(X) >= threshold)
```

`abs(X) >= threshold` now instead returns a *threshold matrix* with 1 and 0 of the same size as  $X$ .

In Figure 9.19 we have applied the two-dimensional DFT to our test image. We have then neglected DFT coefficients which are below certain thresholds, and transformed the samples back to reconstruct the image. When increasing the threshold, the image becomes more and more unclear, but the image is quite clear in the first case, where as much as more than 76.6% of the samples have been zeroed out. A blocking effect at the block boundaries is clearly visible.

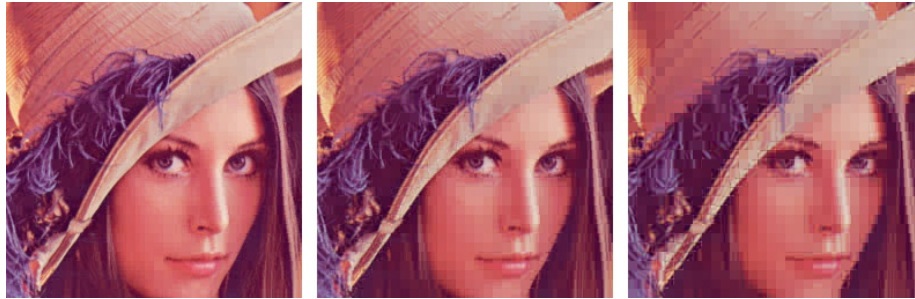


Figure 9.19: The effect on an image when it is transformed with the DFT, and the DFT-coefficients below a certain threshold are zeroed out. The threshold has been increased from left to right, from 100, to 200, and 400. The percentage of pixel values that were zeroed out are 76.6, 89.3, and 95.3, respectively.

### Example 9.25: Change of coordinates with the DCT

Similarly to the DFT, the DCT was the change of coordinates from the standard basis to what we called the DCT basis. Change of coordinates in tensor products between the standard basis and the DCT basis is obtained by substituting with the DCT and the IDCT for the changes of coordinates  $S_1, S_2$  above.

The DCT is used more than the DFT in image processing. In particular, the JPEG standard applies a two-dimensional DCT, rather than a two-dimensional DFT. With the JPEG standard, the blocks are always  $8 \times 8$ , as in the previous example. It is of course not a coincidence that a power of 2 is chosen here: the DCT, as the DFT, has an efficient implementation for powers of 2.

If we follow the same strategy for the DCT as for the DFT example, so that we zero out DCT-coefficients which are below a given threshold <sup>1</sup>, and use the same block sizes, we get the images shown in Figure 9.20. We see similar effects as with the DFT.

It is also interesting to compare with what happens when we drop splitting the image into blocks. Of course, when we neglect many of the DCT-coefficients, we should see some artifacts, but there is no reason to believe that these should be at the old block boundaries. The new artifacts can be seen in Figure 9.21, where the same thresholds as before have been used. Clearly, the new artifacts take a completely different shape.

In the exercises you will be asked to implement functions which generate the images shown in these examples.

### Exercise 9.26: Implement DFT and DCT on blocks

In this section we have used functions which apply the DCT and the DFT either to subblocks of size  $8 \times 8$ , or to the full image. Implement functions `DFTImp18`,

<sup>1</sup>The JPEG standard does not do exactly the kind of thresholding described here. Rather it performs what is called a quantization.

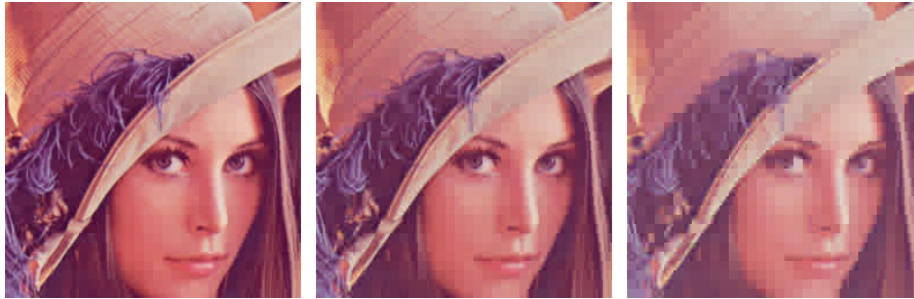


Figure 9.20: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold are zeroed out. The threshold has been increased from left to right, from 30, to 50, and 100. The percentage of pixel values that were zeroed out are 93.2, 95.8, and 97.7, respectively.

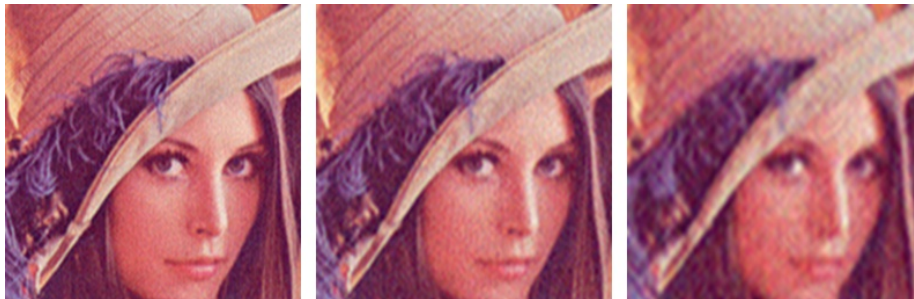


Figure 9.21: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold are zeroed out. The image has not been split into blocks here, and the same thresholds as in Figure 9.20 were used. The percentage of pixel values that were zeroed out are 93.2, 96.6, and 98.8, respectively.

`IDFTImp18`, `DCTImp18`, and `IDCTImp18` which apply the DFT, IDFT, DCT, and IDCT, to consecutive segments of length 8.

### Exercise 9.27: Implement two-dimensional FFT and DCT

Implement functions `DFTImp1Full`, `IDFTImp1Full`, `DCTImp1Full`, and `IDCTImp1Full` which applies the DFT, IDFT, DCT, and IDCT, to the entire vector, and use these to implement. `FFT2`, `IFFT2`, `DCT2`, and `IDCT2` on an image, with the help of the function `tensor_impl`.

**Exercise 9.28: Zeroing out DCT coefficients**

The function `forw_comp_rev_DCT2` in the module `forw_comp_rev` applies the DCT to a part of our sample image, and sets DCT coefficients below a certain threshold to be zero. This is very similar to what the JPEG standard does.

Run `forw_comp_rev_DCT2` for different threshold parameters, and with the functions `DCTImpl8`, `IDCTImpl8`, `DCTImplFull`, and `IDCTImplFull` as parameters. Check that this reproduces the DCT test images of this section, and that the correct numbers of values which have been neglected (i.e. which are below the threshold) are printed on screen.

**Exercise 9.29: Comment code**

Suppose that we have given an image by the matrix `X`. Consider the following code:

```
threshold = 30
[M, N] = shape(X)[0:2]
for n in range(N):
    FFTImpl(X[:, n], FFTKernelStandard)
for m in range(M):
    FFTImpl(X[m, :], FFTKernelStandard)

X = X.*(abs(X) >= threshold)

for n in range(N):
    FFTImpl(X[:, n], FFTKernelStandard, 0)
for m in range(M):
    FFTImpl(X[m, :], FFTKernelStandard, 0)
```

Comment what the code does. Comment in particular on the meaning of the parameter `threshold`, and what effect this has on the image.

**9.5 Summary**

We started by discussing the basic question what an image is, and took a closer look at digital images. We then went through several operations which give meaning for digital images. Many of these operations could be described in terms of a row/column-wise application of filters, and more generally in term of what we called computational molecules. We defined the tensor product, and saw how our operations could be expressed within this framework. The tensor product framework could also be used to state change of coordinates for images, so that we could consider changes of coordinates such as the DFT and the DCT also for images. The algorithm for computing filtering operations or changes of coordinates for images turned out to be similar, in the sense that the one-dimensional counterparts were simply assplied to the rows and the columns in the image.

In introductory image processing textbooks, many other image processing methods are presented. We have limited to the techniques presented here, since



our interest in images is mainly for transformation operations which are useful for compression. An excellent textbook on image processing which uses Matlab is [18]. This contains important topics such as image restoration and reconstruction, geometric transformations, morphology, and object recognition. None of these are considered in this book.

In much literature, one only mentions that filtering can be extended to images by performing one-dimensional filtering for the rows, followed by one-dimensional filtering for the columns, without properly explaining why this is the natural thing to do. The tensor product may be the most natural concept to explain this, and a concept which is firmly established in mathematical literature. Tensor products are usually not part of beginning courses in linear algebra. We have limited the focus here to an introduction to tensor products, and the theory needed to explain filtering an image, and computing the two-dimensional wavelet transform. Some linear algebra books (such as [30]) present tensor products in exercise form only, and often only mentions the Kronecker tensor product, as we defined it.

Many international standards exist for compression of images, and we will take a closer look at two of them in this book. The JPEG standard, perhaps the most popular format for images on the Internet, applies a change of coordinates with a two-dimensional DCT, as described in this chapter. The compression level in JPEG images is selected by the user and may result in conspicuous artefacts if set too high. JPEG is especially prone to artefacts in areas where the intensity changes quickly from pixel to pixel. JPEG is usually lossy, but may also be lossless and has become. The standard defines both the algorithms for encoding and decoding and the storage format. The extension of a JPEG-file is `.jpg` or `.jpeg`. JPEG is short for *Joint Photographic Experts Group*, and was approved as an international standard in 1994. A more detailed description of the standard can be found in [36].

The second standard we will consider is JPEG2000. It was developed to address some of the shortcomings of JPEG, and is based on wavelets. The standard document for this [21] does not focus on explaining the theory behind the standard. As the MP3 standard document, it rather states step-by-step procedures for implementing the standard.

The theory we present related to these image standards concentrate on transforming the image (either with a DWT or a DCT) to obtain something which is more suitable for (lossless or lossy) compression. However, many other steps are also needed in order to obtain a full image compression system. One of these is *quantization*. In the simplest form of quantization, every resulting sample from the transformation is rounded to a fixed number of bits. Quantization can also be done in more advanced ways than this: We have already mentioned that the MP3 standard may use different number of bits for values in the different subbands, depending on the importance of the samples for the human perception. The JPEG2000 standard quantizes in such a way that there is bigger interval around 0 which is quantized to 0, i.e. the rounding error is allowed to be bigger in an interval around 0. Standards which are lossless do not apply quantization, since this always leads to loss.

Somewhere in the image processing or sound processing pipeline, we also need a step which actually achieves compression of the data. Different standards use different lossless coding techniques for this. JPEG2000 uses an advanced type of arithmetic coding for this. JPEG can also use arithmetic coding, but also Huffman coding.

Besides transformation, quantization, and coding, many other steps are used, which have different tasks. Many standards preprocess the pixel values before a transform is applied. Preprocessing may mean to center the pixel values around a certain value (JPEG2000 does this), or extracting the different image components before they are processed separately. Also, the image is often split into smaller parts (often called *tiles*), which are processed separately. For big images this is very important, since it allows users to zoom in on a small part of the image, without processing larger uninteresting parts of the image. Independent processing of the separate tiles makes the image compression what we call *error-resilient*, to errors such as transmission errors, since errors in one tile does not propagate to errors in the other tiles. It is also much more memory-friendly to process the image in several smaller parts, since it is not required to have the entire image in memory at any time. It also gives possibilities for parallel computing. For standards such as JPEG and JPEG2000, tiles are split into even smaller parts, called *blocks*, where parts of the processing within each block also is performed independently. This makes the possibilities for parallel computing even bigger.

An image standard also defines how to store metadata about an image, and what metadata is accepted, like resolution, time when the image was taken, where the image was taken (such as GPS coordinates), and similar information. Metadata can also tell us how the color in the image are represented. As we have already seen, in most color images the color of a pixel is represented in terms of the amount of red, green and blue or  $(r, g, b)$ . But there are other possibilities as well: Instead of storing all 24 bits of color information in cases where each of the three color components needs 8 bits, it is common to create a table of up to 256 colors with which a given image could be represented quite well. Instead of storing the 24 bits, one then just stores a color table in the metadata, and at each pixel, the eight bits corresponding to the correct entry in the table. This is usually referred to as *eight-bit color*, and the table is called a *look-up* table or *palette*. For large photographs, however, 256 colors is far from sufficient to obtain reasonable colour reproduction. Metadata is usually stored in the beginning of the file, formatted in a very specific way.

#### **What you should have learned in this chapter.**

- How to read, write, and show images on your computer.
- How to extract different color components.
- How to convert from color to grey-level images.
- How to use functions for adjusting the contrast.

- The operation  $X \rightarrow S_1 X (S_2)^T$  can be used to define operations on images, based on one-dimensional operations  $S_1$  and  $S_2$ . This amounts to applying  $S_1$  to all columns in the image, and then  $S_2$  to all rows in the result. You should know how this operation can be conveniently expressed with tensor product notation, and that in the typical case when  $S_1$  and  $S_2$  are filters, this can equivalently be expressed in terms of computational molecules.
- The case when the  $S_i$  are smoothing filters gives rise to smoothing operations on images.
- A simple highpass filter, corresponding to taking the derivative, gives rise to edge-detection operations on images.
- The operation  $X \rightarrow S_1 X (S_2)^T$  can also be used to facilitate change of coordinates in images, in addition to filtering images. In other words, change of coordinates is done first column by column, then row by row. The DCT and the DFT are particular changes of coordinates used for images.

## Chapter 10

# Using tensor products to apply wavelets to images

Previously we have used the theory of wavelets to analyze sound. We would also like to use wavelets in a similar way to analyze images. Since the tensor product concept constructs two dimensional objects (matrices) from one-dimensional objects (vectors), we are lead to believe that tensor products can also be used to apply wavelets to images. In this chapter we will see that this can indeed be done. The vector spaces we  $V_m$  encountered for wavelets were function spaces, however. What we therefore need first is to establish a general definition of tensor products of function spaces. This will be done in the first section of this chapter. In the second section we will then specialize the function spaces to the spaces  $V_m$  we use for wavelets, and interpret the tensor product of these and the wavelet transform applied to images more carefully. Finally we will look at some examples on this theory applied to some example images.

The examples in this chapter can be run from the notebook `applinalgnbchap10.ipynb`.

### 10.1 Tensor product of function spaces

In the setting of functions, it will turn out that the tensor product of two univariate functions can be most intuitively defined as a function in two variables. This seems somewhat different from the strategy of Chapter 9, but we will see that the results we obtain will be very similar.

**Definition 10.1.** *Tensor product of function spaces.*

Let  $U_1$  and  $U_2$  be vector spaces of functions, defined on the intervals  $[0, M]$  and  $[0, N]$ , respectively, and suppose that  $f_1 \in U_1$  and  $f_2 \in U_2$ . The tensor product of  $f_1$  and  $f_2$ , denoted  $f_1 \otimes f_2$ , is the function in two variables defined on  $[0, M] \times [0, N]$  by

$$(f_1 \otimes f_2)(t_1, t_2) = f_1(t_1)f_2(t_2).$$

$f_1 \otimes f_2$  is also called the separable extension of  $f_1$  and  $f_2$  to two variables. The tensor product of the spaces  $U_1 \otimes U_2$  is the vector space spanned by the two-variable functions  $\{f_1 \otimes f_2\}_{f_1 \in U_1, f_2 \in U_2}$ .

We will always assume that the spaces  $U_1$  and  $U_2$  consist of functions which are at least integrable. In this case  $U_1 \otimes U_2$  is also an inner product space, with the inner product given by a double integral,

$$\langle f, g \rangle = \int_0^N \int_0^M f(t_1, t_2)g(t_1, t_2)dt_1dt_2. \quad (10.1)$$

In particular, this says that

$$\begin{aligned} \langle f_1 \otimes f_2, g_1 \otimes g_2 \rangle &= \int_0^N \int_0^M f_1(t_1)f_2(t_2)g_1(t_1)g_2(t_2)dt_1dt_2 \\ &= \int_0^M f_1(t_1)g_1(t_1)dt_1 \int_0^N f_2(t_2)g_2(t_2)dt_2 = \langle f_1, g_1 \rangle \langle f_2, g_2 \rangle. \end{aligned} \quad (10.2)$$

This means that for tensor products, a double integral can be computed as the product of two one-dimensional integrals. This formula also ensures that inner products of tensor products of functions obey the same rule as we found for tensor products of vectors in Exercise 9.23.

The tensor product space defined in Definition 10.1 is useful for approximation of functions of two variables if each of the two spaces of univariate functions have good approximation properties.

**Idea 10.2.** *Using tensor products for approximation.*

If the spaces  $U_1$  and  $U_2$  can be used to approximate functions in one variable, then  $U_1 \otimes U_2$  can be used to approximate functions in two variables.

We will not state this precisely, but just consider some important examples.

### 10.1.1 Tensor products of polynomials

Let  $U_1 = U_2$  be the space of all polynomials of finite degree. We know that  $U_1$  can be used for approximating many kinds of functions, such as continuous functions, for example by Taylor series. The tensor product  $U_1 \otimes U_1$  consists of all functions on the form  $\sum_{i,j} \alpha_{i,j} t_1^i t_2^j$ . It turns out that polynomials in several variables have approximation properties analogous to univariate polynomials.

### 10.1.2 Tensor products of Fourier spaces

Let  $U_1 = U_2 = V_{N,T}$  be the  $N$ th order Fourier space which is spanned by the functions

$$e^{-2\pi i Nt/T}, \dots, e^{-2\pi it/T}, 1, e^{2\pi it/T}, \dots, e^{2\pi i Nt/T}$$

The tensor product space  $U_1 \otimes U_1$  now consists of all functions on the form

$$\sum_{k,l=-N}^N \alpha_{k,l} e^{2\pi i k t_1/T} e^{2\pi i l t_2/T}.$$

One can show that this space has approximation properties similar to  $V_{N,T}$  for functions in two variables. This is the basis for the theory of Fourier series in two variables.

In the following we think of  $U_1 \otimes U_2$  as a space which can be used for approximating a general class of functions. By associating a function with the vector of coordinates relative to some basis, and a matrix with a function in two variables, we have the following parallel to Theorem 9.16:

**Theorem 10.3.** *Bases for tensor products of function spaces.*

If  $\{f_i\}_{i=0}^{M-1}$  is a basis for  $U_1$  and  $\{g_j\}_{j=0}^{N-1}$  is a basis for  $U_2$ , then  $\{f_i \otimes g_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$  is a basis for  $U_1 \otimes U_2$ . Moreover, if the bases for  $U_1$  and  $U_2$  are orthogonal/orthonormal, then the basis for  $U_1 \otimes U_2$  is orthogonal/orthonormal.

*Proof.* The proof is similar to that of Theorem 9.16: if

$$\sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j} (f_i \otimes g_j) = 0,$$

we define  $h_i(t_2) = \sum_{j=0}^{N-1} \alpha_{i,j} g_j(t_2)$ . It follows as before that  $\sum_{i=0}^{M-1} h_i(t_2) f_i = 0$  for any  $t_2$ , so that  $h_i(t_2) = 0$  for any  $t_2$  due to linear independence of the  $f_i$ . But then  $\alpha_{i,j} = 0$  also, due to linear independence of the  $g_j$ . The statement about orthogonality follows from Equation (10.2).  $\square$

We can now define the tensor product of two bases of functions as before, and coordinate matrices as before:

**Definition 10.4.** *Coordinate matrix.*

if  $\mathcal{B} = \{f_i\}_{i=0}^{M-1}$  and  $\mathcal{C} = \{g_j\}_{j=0}^{N-1}$ , we define  $\mathcal{B} \otimes \mathcal{C}$  as the basis  $\{f_i \otimes g_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$  for  $U_1 \otimes U_2$ . We say that  $X$  is the coordinate matrix of  $f$  if  $f(t_1, t_2) = \sum_{i,j} X_{i,j} (f_i \otimes g_j)(t_1, t_2)$ , where  $X_{i,j}$  are the elements of  $X$ .

Theorem 9.18 can also be proved in the same way in the context of function spaces. We state this as follows:

**Theorem 10.5.** *Change of coordinates in tensor products of function spaces.*

Assume that  $U_1$  and  $U_2$  are function spaces, and that

- $\mathcal{B}_1, \mathcal{C}_1$  are bases for  $U_1$ , and that  $S_1$  is the change of coordinates matrix from  $\mathcal{B}_1$  to  $\mathcal{C}_1$ ,
- $\mathcal{B}_2, \mathcal{C}_2$  are bases for  $U_2$ , and that  $S_2$  is the change of coordinates matrix from  $\mathcal{B}_2$  to  $\mathcal{C}_2$ .

Both  $\mathcal{B}_1 \otimes \mathcal{B}_2$  and  $\mathcal{C}_1 \otimes \mathcal{C}_2$  are bases for  $U_1 \otimes U_2$ , and if  $X$  is the coordinate matrix in  $\mathcal{B}_1 \otimes \mathcal{B}_2$ ,  $Y$  the coordinate matrix in  $\mathcal{C}_1 \otimes \mathcal{C}_2$ , then the change of coordinates from  $\mathcal{B}_1 \otimes \mathcal{B}_2$  to  $\mathcal{C}_1 \otimes \mathcal{C}_2$  can be computed as

$$Y = S_1 X (S_2)^T. \quad (10.3)$$

## 10.2 Tensor product of function spaces in a wavelet setting

We will now specialize the spaces  $U_1, U_2$  from Definition 10.1 to the resolution spaces  $V_m$  and the detail spaces  $W_m$ , arising from a given wavelet. We can in particular form the tensor products  $\phi_{0,n_1} \otimes \phi_{0,n_2}$ . We will assume that

- the first component  $\phi_{0,n_1}$  has period  $M$  (so that  $\{\phi_{0,n_1}\}_{n_1=0}^{M-1}$  is a basis for the first component space),
- the second component  $\phi_{0,n_2}$  has period  $N$  (so that  $\{\phi_{0,n_2}\}_{n_2=0}^{N-1}$  is a basis for the second component space).

When we speak of  $V_0 \otimes V_0$  we thus mean an  $MN$ -dimensional space with basis  $\{\phi_{0,n_1} \otimes \phi_{0,n_2}\}_{(n_1,n_2)=(0,0)}^{(M-1,N-1)}$ , where the coordinate matrices are  $M \times N$ . This difference in the dimension of the two components is done to allow for images where the number of rows and columns may be different. In the following we will implicitly assume that the component spaces have dimension  $M$  and  $N$ , to ease notation. If we use that  $(\phi_{m-1}, \psi_{m-1})$  also is a basis for  $V_m$ , we get the following corollary to Theorem 10.3:

**Corollary 10.6.** *Bases for tensor products.*

Let  $\phi, \psi$  be a scaling function and a mother wavelet. Then the two sets of tensor products given by

$$\phi_m \otimes \phi_m = \{\phi_{m,n_1} \otimes \phi_{m,n_2}\}_{n_1,n_2}$$

and

$$\begin{aligned} & (\phi_{m-1}, \psi_{m-1}) \otimes (\phi_{m-1}, \psi_{m-1}) \\ &= \{\phi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\ & \quad \phi_{m-1,n_1} \otimes \psi_{m-1,n_2}, \\ & \quad \psi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\ & \quad \psi_{m-1,n_1} \otimes \psi_{m-1,n_2}\}_{n_1,n_2} \end{aligned}$$

are both bases for  $V_m \otimes V_m$ . This second basis is orthogonal/orthonormal whenever the first basis is.

From this we observe that, while the one-dimensional wavelet decomposition splits  $V_m$  into a direct sum of the two vector spaces  $V_{m-1}$  and  $W_{m-1}$ , the corresponding two-dimensional decomposition splits  $V_m \otimes V_m$  into a direct sum of four tensor product vector spaces. These vector spaces deserve individual names:

**Definition 10.7.** *Tensor product spaces.*

We define the following tensor product spaces:

- The space  $W_m^{(0,1)}$  spanned by  $\{\phi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$ ,
- The space  $W_m^{(1,0)}$  spanned by  $\{\psi_{m,n_1} \otimes \phi_{m,n_2}\}_{n_1,n_2}$ ,
- The space  $W_m^{(1,1)}$  spanned by  $\{\psi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$ .

Since these spaces are linearly independent, we can write

$$V_m \otimes V_m = (V_{m-1} \otimes V_{m-1}) \oplus W_{m-1}^{(0,1)} \oplus W_{m-1}^{(1,0)} \oplus W_{m-1}^{(1,1)}. \quad (10.4)$$

Also in the setting of tensor products we refer to  $V_{m-1} \otimes V_{m-1}$  as the space of low-resolution approximations. The remaining parts,  $W_{m-1}^{(0,1)}$ ,  $W_{m-1}^{(1,0)}$ , and  $W_{m-1}^{(1,1)}$ , are referred to as detail spaces. The coordinate matrix of

$$\sum_{n_1, n_2=0}^{2^{m-1}N} (c_{m-1, n_1, n_2} (\phi_{m-1, n_1} \otimes \phi_{m-1, n_2}) + w_{m-1, n_1, n_2}^{(0,1)} (\phi_{m-1, n_1} \otimes \psi_{m-1, n_2}) + w_{m-1, n_1, n_2}^{(1,0)} (\psi_{m-1, n_1} \otimes \phi_{m-1, n_2}) + w_{m-1, n_1, n_2}^{(1,1)} (\psi_{m-1, n_1} \otimes \psi_{m-1, n_2})) \quad (10.5)$$

in the basis  $(\phi_{m-1}, \psi_{m-1}) \otimes (\phi_{m-1}, \psi_{m-1})$  is

$$\left( \begin{array}{cc|cc} c_{m-1,0,0} & \cdots & w_{m-1,0,0}^{(0,1)} & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ \hline w_{m-1,0,0}^{(1,0)} & \cdots & w_{m-1,0,0}^{(1,1)} & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{array} \right). \quad (10.6)$$

The coordinate matrix is thus split into four submatrices:

- The  $c_{m-1}$ -values, i.e. the coordinates for  $V_{m-1} \oplus V_{m-1}$ . This is the upper left corner in Equation (10.6).
- The  $w_{m-1}^{(0,1)}$ -values, i.e. the coordinates for  $W_{m-1}^{(0,1)}$ . This is the upper right corner in Equation (10.6).
- The  $w_{m-1}^{(1,0)}$ -values, i.e. the coordinates for  $W_{m-1}^{(1,0)}$ . This is the lower left corner in Equation (10.6).



- The  $w_{m-1}^{(1,1)}$ -values, i.e. the coordinates for  $W_{m-1}^{(1,1)}$ . This is the lower right corner in Equation (10.6).

The  $w_{m-1}^{(i,j)}$ -values are as in the one-dimensional situation often referred to as wavelet coefficients. Let us consider the Haar wavelet as an example.

### Example 10.1: Piecewise constant functions

If  $V_m$  is the vector space of piecewise constant functions on any interval of the form  $[k2^{-m}, (k+1)2^{-m})$  (as in the piecewise constant wavelet),  $V_m \otimes V_m$  is the vector space of functions in two variables which are constant on any square of the form  $[k_12^{-m}, (k_1+1)2^{-m}) \times [k_22^{-m}, (k_2+1)2^{-m})$ . Clearly  $\phi_{m,k_1} \otimes \phi_{m,k_2}$  is constant on such a square and 0 elsewhere, and these functions are a basis for  $V_m \otimes V_m$ .

Let us compute the orthogonal projection of  $\phi_{1,k_1} \otimes \phi_{1,k_2}$  onto  $V_0 \otimes V_0$ . Since the Haar wavelet is orthonormal, the basis functions in (10.4) are orthonormal, and we can thus use the orthogonal decomposition formula to find this projection. Clearly  $\phi_{1,k_1} \otimes \phi_{1,k_2}$  has different support from all except one of  $\phi_{0,n_1} \otimes \phi_{0,n_2}$ . Since

$$\langle \phi_{1,k_1} \otimes \phi_{1,k_2}, \phi_{0,n_1} \otimes \phi_{0,n_2} \rangle = \langle \phi_{1,k_1}, \phi_{0,n_1} \rangle \langle \phi_{1,k_2}, \phi_{0,n_2} \rangle = \frac{\sqrt{2}}{2} \frac{\sqrt{2}}{2} = \frac{1}{2}$$

when the supports intersect, we obtain

$$\text{proj}_{V_0 \otimes V_0}(\phi_{1,k_1} \otimes \phi_{1,k_2}) = \begin{cases} \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) & \text{when } k_1, k_2 \text{ are even} \\ \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,(k_2-1)/2}) & \text{when } k_1 \text{ is even, } k_2 \text{ is odd} \\ \frac{1}{2}(\phi_{0,(k_1-1)/2} \otimes \phi_{0,k_2/2}) & \text{when } k_1 \text{ is odd, } k_2 \text{ is even} \\ \frac{1}{2}(\phi_{0,(k_1-1)/2} \otimes \phi_{0,(k_2-1)/2}) & \text{when } k_1, k_2 \text{ are odd} \end{cases}$$

So, in this case there were 4 different formulas, since there were 4 different combinations of even/odd. Let us also compute the projection onto the orthogonal complement of  $V_0 \otimes V_0$  in  $V_1 \otimes V_1$ , and let us express this in terms of the  $\phi_{0,n}, \psi_{0,n}$ , like we did in the one-variable case. Also here there are 4 different formulas. When  $k_1, k_2$  are both even we obtain

$$\begin{aligned}
 & \phi_{1,k_1} \otimes \phi_{1,k_2} - \text{proj}_{V_0 \otimes V_0}(\phi_{1,k_1} \otimes \phi_{1,k_2}) \\
 &= \phi_{1,k_1} \otimes \phi_{1,k_2} - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\
 &= \left( \frac{1}{\sqrt{2}}(\phi_{0,k_1/2} + \psi_{0,k_1/2}) \right) \otimes \left( \frac{1}{\sqrt{2}}(\phi_{0,k_2/2} + \psi_{0,k_2/2}) \right) - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\
 &= \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\phi_{0,k_1/2} \otimes \psi_{0,k_2/2}) \\
 &+ \frac{1}{2}(\psi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \psi_{0,k_2/2}) - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\
 &= \frac{1}{2}(\phi_{0,k_1/2} \otimes \psi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \psi_{0,k_2/2}).
 \end{aligned}$$

Here we have used the relation  $\phi_{1,k_i} = \frac{1}{\sqrt{2}}(\phi_{0,k_i/2} + \psi_{0,k_i/2})$ , which we have from our first analysis of the Haar wavelet. Checking the other possibilities we find similar formulas for the projection onto the orthogonal complement of  $V_0 \otimes V_0$  in  $V_1 \otimes V_1$  when either  $k_1$  or  $k_2$  is odd. In all cases, the formulas use the basis functions for  $W_0^{(0,1)}$ ,  $W_0^{(1,0)}$ ,  $W_0^{(1,1)}$ . These functions are shown in Figure 10.1, together with the function  $\phi \otimes \phi \in V_0 \otimes V_0$ .

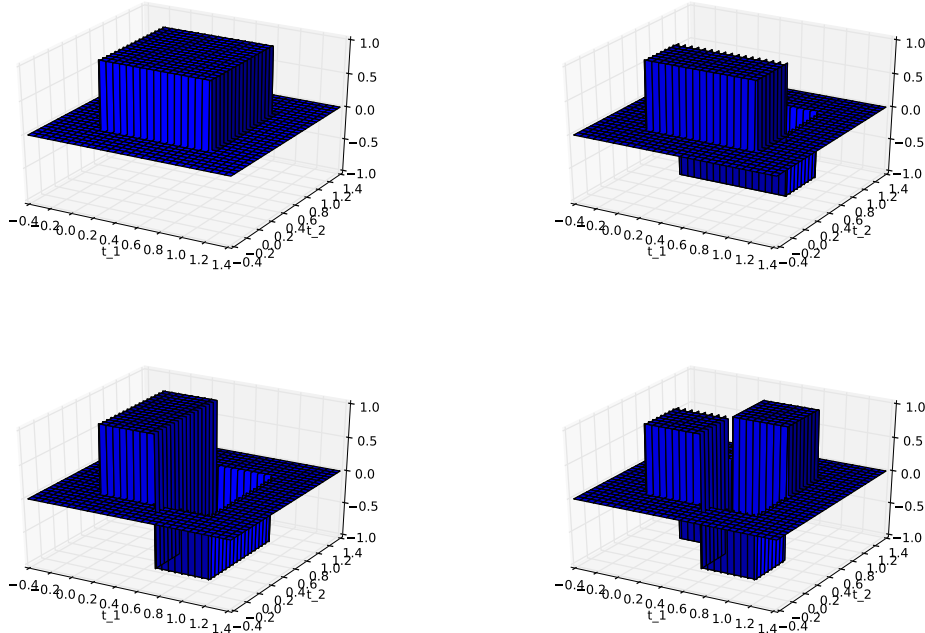


Figure 10.1: The functions  $\phi \otimes \phi$ ,  $\phi \otimes \psi$ ,  $\psi \otimes \phi$ ,  $\psi \otimes \psi$ , which are bases for  $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$  for the Haar wavelet.

**Example 10.2: Piecewise linear functions**

If we instead use any of the wavelets for piecewise linear functions, the wavelet basis functions are not orthogonal anymore, just as in the one-dimensional case. The new basis functions are shown in Figure 10.2 for the alternative piecewise linear wavelet.

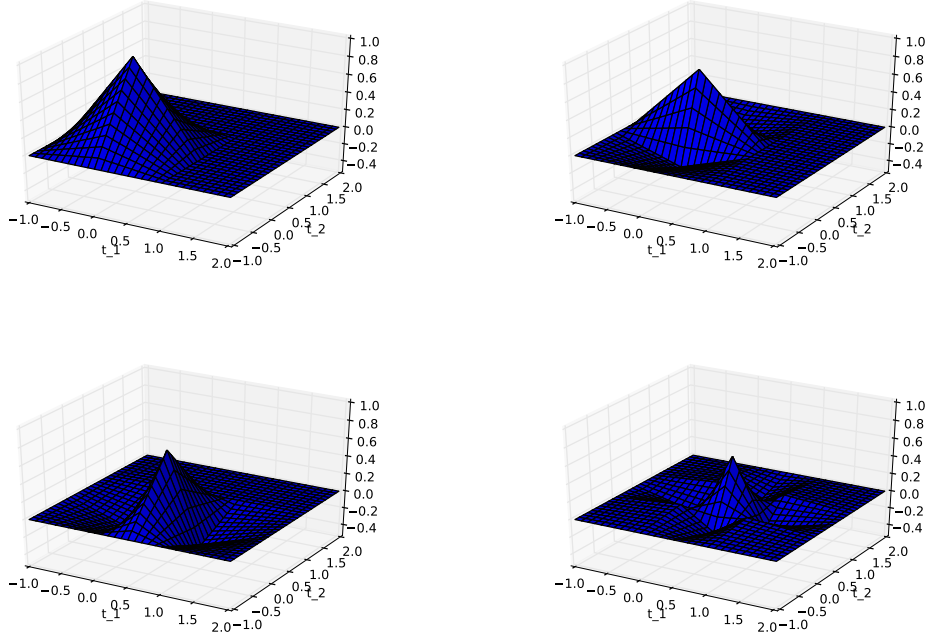


Figure 10.2: The functions  $\phi \otimes \phi$ ,  $\phi \otimes \psi$ ,  $\psi \otimes \phi$ ,  $\psi \otimes \psi$ , which are bases for  $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$  for the alternative piecewise linear wavelet.

**10.2.1 Interpretation**

An immediate corollary of Theorem 10.5 is the following:

**Corollary 10.8.** *Implementing tensor product.*

Let

$$A_m = P_{(\phi_{m-1}, \psi_{m-1}) \leftarrow \phi_m}$$

$$B_m = P_{\phi_m \leftarrow (\phi_{m-1}, \psi_{m-1})}$$

be the stages in the DWT and the IDWT, and let

$$X = (c_{m,i,j})_{i,j} \quad Y = \begin{pmatrix} (c_{m-1,i,j})_{i,j} & (w_{m-1,i,j}^{(0,1)})_{i,j} \\ (w_{m-1,i,j}^{(1,0)})_{i,j} & (w_{m-1,i,j}^{(1,1)})_{i,j} \end{pmatrix} \quad (10.7)$$

be the coordinate matrices in  $\phi_m \otimes \phi_m$ , and  $(\phi_{m-1}, \psi_{m-1}) \otimes (\phi_{m-1}, \psi_{m-1})$ , respectively. Then

$$Y = A_m X A_m^T \quad (10.8)$$

$$X = B_m Y B_m^T \quad (10.9)$$

By the  $m$ -level two-dimensional DWT/IDWT (or DWT2/IDWT2) we mean the change of coordinates where this is repeated  $m$  times as in a DWT/IDWT.

It is straightforward to make implementations of DWT2 and IDWT2, in the same way we implemented DWTImp1 and IDWTImp1. In Exercise 10.7 you will be asked to program functions DW2TImp1 and IDW2TImp1 for this. Each stage in DWT2 and IDWT2 can now be implemented by substituting the matrices  $A_m, B_m$  above into the code following Theorem 9.18. When using many levels of the DWT2, the next stage is applied only to the upper left corner of the matrix, just as the DWT at the next stage only is applied to the first part of the coordinates. At each stage, the upper left corner of the coordinate matrix (which gets smaller at each iteration), is split into four equally big parts. This is illustrated in Figure 10.3, where the different types of coordinates which appear in the first two stages in a DWT2 are indicated.

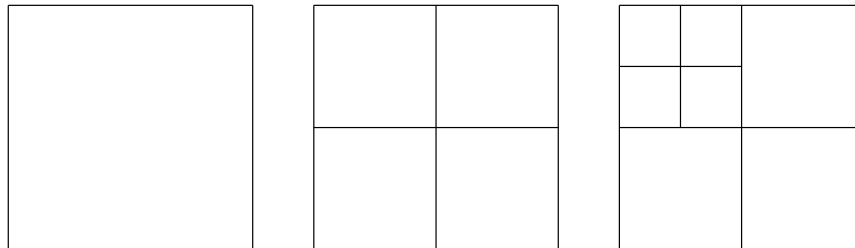


Figure 10.3: Illustration of the different coordinates in a two level DWT2 before the first stage is performed (left), after the first stage (middle), and after the second stage (right).

It is instructive to see what information the different types of coordinates in an image represent. In the following examples we will discard some types of coordinates, and view the resulting image. Discarding a type of coordinates will be illustrated by coloring the corresponding regions from Figure 10.3 black. As an example, if we perform a two-level DWT2 (i.e. we start with a coordinate matrix in the basis  $\phi_2 \otimes \phi_2$ ), Figure 10.4 illustrates first the collection of all coordinates, and then the resulting collection of coordinates after removing subbands at the first level successively.

Figure 10.5 illustrates in the same way incremental removal of the subbands at the second level.

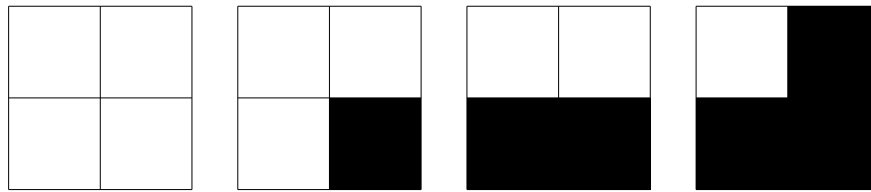


Figure 10.4: Graphical representation of neglecting the wavelet coefficients at the first level. After applying DWT2, the wavelet coefficients are split into four parts, as shown in the left figure. In the following figures we have removed coefficients from  $W_1^{(1,1)}$ ,  $W_1^{(1,0)}$ , and  $W_1^{(0,1)}$ , in that order.

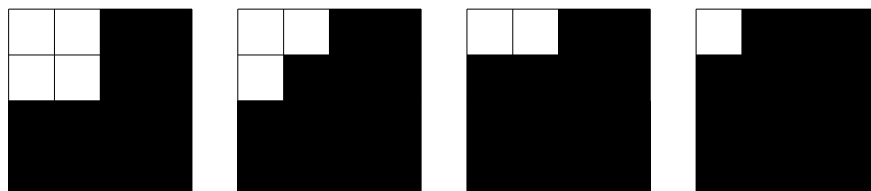


Figure 10.5: Graphical representation of neglecting the wavelet coefficients at the second level. After applying the second stage in DWT2, the wavelet coefficients from the upper left corner are also split into four parts, as shown in the left figure. In the following figures we have removed coefficients from  $W_2^{(1,1)}$ ,  $W_2^{(1,0)}$ , and  $W_2^{(0,1)}$ , in that order.

Before we turn to experiments on images using wavelets, we would like to make another interpretation on the corners in the matrices after the DWT2, which correspond to the different coordinates  $(c_{m-1,i,j})_{i,j}$ ,  $(w^{(0,1)})_{m-1,i,j}$ ,  $(w^{(1,0)})_{m-1,i,j}$ , and  $(w^{(1,1)})_{m-1,i,j}$ . It turns out that these corners have natural interpretations in terms of the filter characterization of wavelets, as given in Chapter 6. Recall again that in a DWT2, the DWT is first applied to the columns in the image, then to the rows in the image. Recall first that the DWT2 applies first the DWT to all columns, and then to all rows in the resulting matrix.

First the DWT is applied to all columns in the image. Since the first half of the coordinates in a DWT are outputs from a lowpass filter  $H_0$  (Theorem 6.3), the upper half after the DWT has now been subject to a lowpass filter to the columns. Similarly, the second half of the coordinates in a DWT are outputs from a highpass filter  $H_1$  (Theorem 6.3 again), so that the bottom half after the DWT has been subject to a highpass filter to the columns.

Then the DWT is applied to all rows in the image. Similarly as when we applied the DWT to the columns, the left half after the DWT has been subject

to the same lowpass filter to the rows, and the right half after the DWT has been subject to the same highpass filter to the rows.

These observations split the resulting matrix after DWT2 into four blocks, with each block corresponding to a combination of lowpass and highpass filters. The following names are thus given to these blocks:

- The upper left corner is called the LL-subband,
- The upper right corner is called the LH-subband,
- The lower left corner is called the HL-subband,
- The lower right corner is called the HH-subband.

The two letters indicate the type of filters which have been applied (L=lowpass, H=highpass). The first letter indicates the type of filter which is applied to the columns, the second indicates which is applied to the rows. The order is therefore important. The name *subband* comes from the interpretation of these filters as being selective on a certain frequency band. In conclusion, a block in the matrix after the DWT2 corresponds to applying a combination of lowpass/highpass filters to the rows of the columns of the image. Due to this, and since lowpass filters extract slow variations, highpass filters abrupt changes, the following holds:

**Observation 10.9.** *Visual interpretation of the DWT2.*

After the DWT2 has been applied to an image, we expect to see the following:

- In the upper left corner, slow variations in both the vertical and horizontal directions are captured, i.e. this is a low-resolution version of the image.
- In the upper right corner, slow variations in the vertical direction are captured, together with abrupt changes in the horizontal direction.
- In the lower left corner, slow variations in the horizontal direction are captured, together with abrupt changes in the vertical direction.
- In the lower right corner, abrupt changes in both directions appear are captured.

These effects will be studied through examples in the next section.

### 10.3 Experiments with images using wavelets

In this section we will make some experiments with images using the wavelets we have considered. The wavelet theory is applied to images in the following way: We first visualize the pixels in the image as coordinates in the basis  $\phi_m \otimes \phi_m$  (so that the image has size  $(2^m M) \times (2^m N)$ ). As in the case for sound, this will represent a good approximation when  $m$  is large. We then perform a change of coordinates with the DWT2. As we did for sound, we can then either set

the detail components from the  $W_k^{(i,j)}$ -spaces to zero, or the low-resolution approximation from  $V_0 \otimes V_0$  to zero, depending on whether we want to inspect the detail components or the low-resolution approximation. Finally we apply the IDWT2 to end up with coordinates in  $\phi_m \otimes \phi_m$  again, and display the new image with pixel values equal to these coordinates.

### Example 10.3: Applying the Haar wavelet to a very simple example image

Let us apply the Haar wavelet to the sample chess pattern example image from Figure 9.17. The lowpass filter of the Haar wavelet was essentially a smoothing filter with two elements. Also, as we have seen, the highpass filter essentially computes an approximation to the partial derivative. Clearly, abrupt changes in the vertical and horizontal directions appear here only at the edges in the chess pattern, and abrupt changes in both directions appear only at the grid points in the chess pattern. Due to Observation 10.9, after a DWT2 we expect to see the following:

- In the upper left corner, we should see a low-resolution version of the image.
- In the upper right corner, only the vertical edges in the chess pattern should be visible.
- In the lower left corner, only the horizontal edges in the chess pattern should be visible.
- In the lower right corner, only the grid points in the chess pattern should be visible.

These effects are seen clearly if we apply one level of the DWT2 to the chess pattern example image. The result of this can be seen in Figure 10.6.

### Example 10.4: Creating thumbnail images

Let us apply the Haar wavelet to our sample image. After the DWT2, the upper left submatrices represent the low-resolution approximations from  $V_{m-1} \otimes V_{m-1}$ ,  $V_{m-2} \otimes V_{m-2}$ , and so on. We can now use the following code to store the low-resolution approximation for  $m = 1$ :

```
DWT2Impl(X, 1, 'Haar')
X = X[0:(shape(X)[0]/2), 0:(shape(X)[1]/2)]
mapto01(X); X *= 255
```

Note that here it is necessary to map the result back to  $[0, 255]$ .

In Figure 10.7 the results are shown up to 4 resolutions. In Figure 10.8 we have also shown the entire result after a 1- and 2-stage DWT2 on the image. The first two thumbnail images can be seen as the the upper left corners of the first two images. The other corners represent detail.

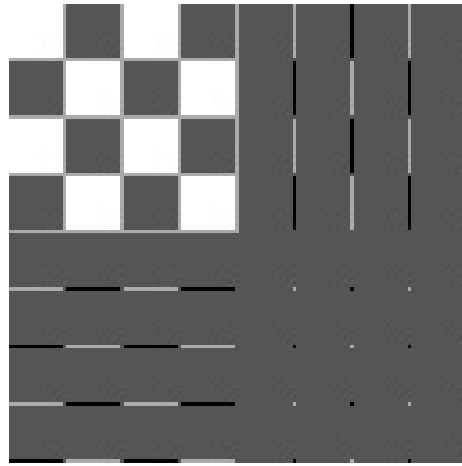


Figure 10.6: The chess pattern example image after application of the DWT2. The Haar wavelet was used.



Figure 10.7: The corresponding thumbnail images for the Image of Lena, obtained with a DWT of 1, 2, 3, and 4 levels.

### Example 10.5: Detail and low-resolution approximations for different wavelets

Let us take a closer look at the images generated when we use different wavelets. Above we viewed the low-resolution approximation as a smaller image. Let us compare with the image resulting from setting the wavelet detail coefficients to zero, and viewing the result as an image of the same size. In particular, let us neglect the wavelet coefficients as pictured in Figure 10.4 and Figure 10.5. We should expect that the lower order resolution approximations from  $V_0$  are worse when  $m$  increase.

Figure 10.9 confirms this for the lower order resolution approximations for the Haar wavelet. Alternatively, we should see that the higher order detail spaces contain more information as  $m$  increases. The result is shown in Figure 10.10.

Figures 10.11 and 10.12 confirms the same for the CDF 9/7 wavelet, which also shows some improvement in the low resolution approximations. The black



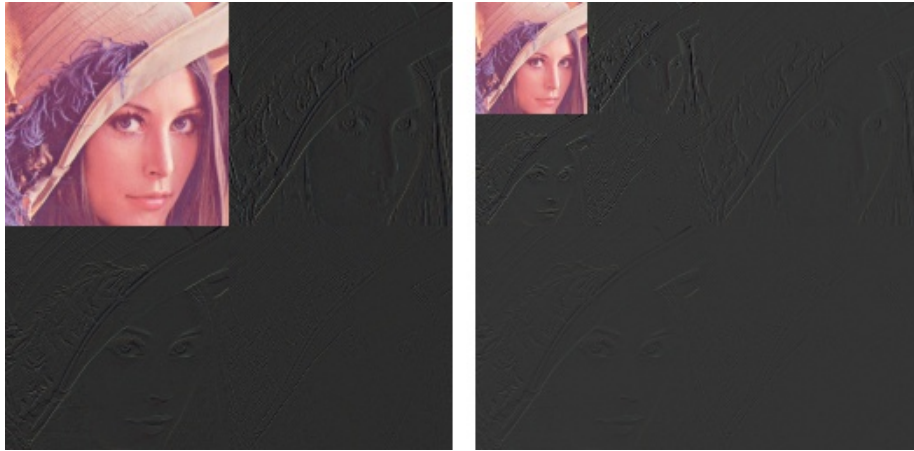


Figure 10.8: The corresponding image resulting from a wavelet transform with the Haar-wavelet for  $m = 1$  and  $m = 2$ .

color indicates values which are close to 0. In other words, most of the coefficients are close to 0.

### Example 10.6: The Spline 5/3 wavelet and removing bands in the detail spaces

Since the detail components in images are split into three bands, another thing we can try is to neglect only parts of the detail components (i.e.e some of  $W_m^{(1,1)}$ ,  $W_m^{(1,0)}$ ,  $W_m^{(0,1)}$ ), contrary to the one-dimensional case. Let us use the Spline 5/3 wavelet.

The resulting images when the bands on the first level indicated in Figure 10.4 are removed are shown in Figure 10.13. The corresponding plot for the second level is shown in Figure 10.14.

The image is seen still to resemble the original one, even after two levels of wavelets coefficients have been neglected. This in itself is good for compression purposes, since we may achieve compression simply by dropping the given coefficients. However, if we continue to neglect more levels of coefficients, the result will look poorer.

In Figure 10.15 we have also shown the resulting image after the third and fourth levels of detail have been neglected. Although we still can see details in the image, the quality in the image is definitely poorer. Although the quality is poorer when we neglect levels of wavelet coefficients, all information is kept if we additionally include the detail/bands.

In Figure 10.16, we have shown the corresponding detail for DWT's with 1, 2, 3, and 4 levels. Clearly, more detail can be seen in the image when more of the detail is included.

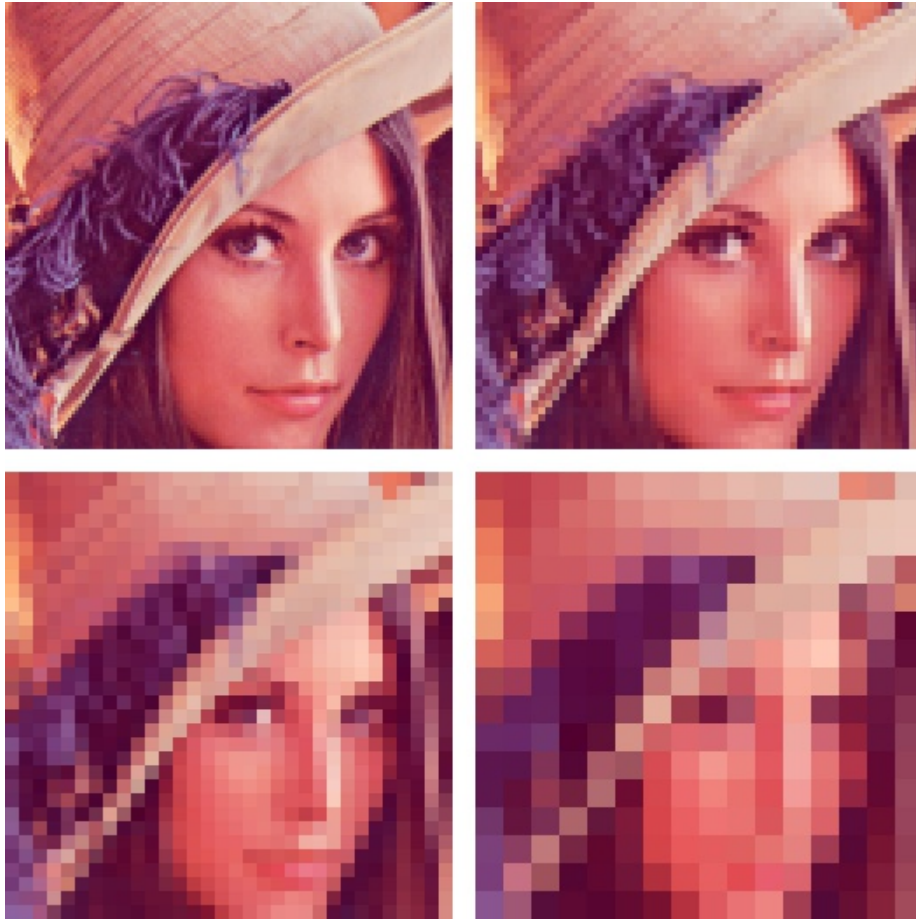


Figure 10.9: Low resolution approximations of the Lena image, for the Haar wavelet.

As mentioned, the procedure developed in this section for applying a wavelet transform to an image with the help of the tensor product construction, is adopted in the JPEG2000 standard. This lossy (can also be used as lossless) image format was developed by the Joint Photographic Experts Group and published in 2000. After significant processing of the wavelet coefficients, the final coding with JPEG2000 uses an advanced version of arithmetic coding. At the cost of increased encoding and decoding times, JPEG2000 leads to as much as 20 % improvement in compression ratios for medium compression rates, possibly more for high or low compression rates. The artefacts are less visible than in JPEG and appear at higher compression rates. Although a number of components in JPEG2000 are patented, the patent holders have agreed that the core software should be available free of charge, and JPEG2000 is part of most



Figure 10.10: Detail of the Lena image, for the Haar wavelet.

Linux distributions. However, there appear to be some further, rather obscure, patents that have not been licensed, and this may be the reason why JPEG2000 is not used more. The extension of JPEG2000 files is `.jp2`.

### Exercise 10.7: Implement two-dimensional DWT

Implement functions

```
DWT2Impl_internal(x, nres, f, bd_mode)
IDWT2Impl_internal(x, nres, f, bd_mode)
```

which perform the  $m$ -level DWT2 and the IDWT2, respectively, on an image. The arguments are the as those in `DWTImpl_internal` and `IDWTImpl_internal`, with the input vector `x` replaced with a two-dimensional object/image. The

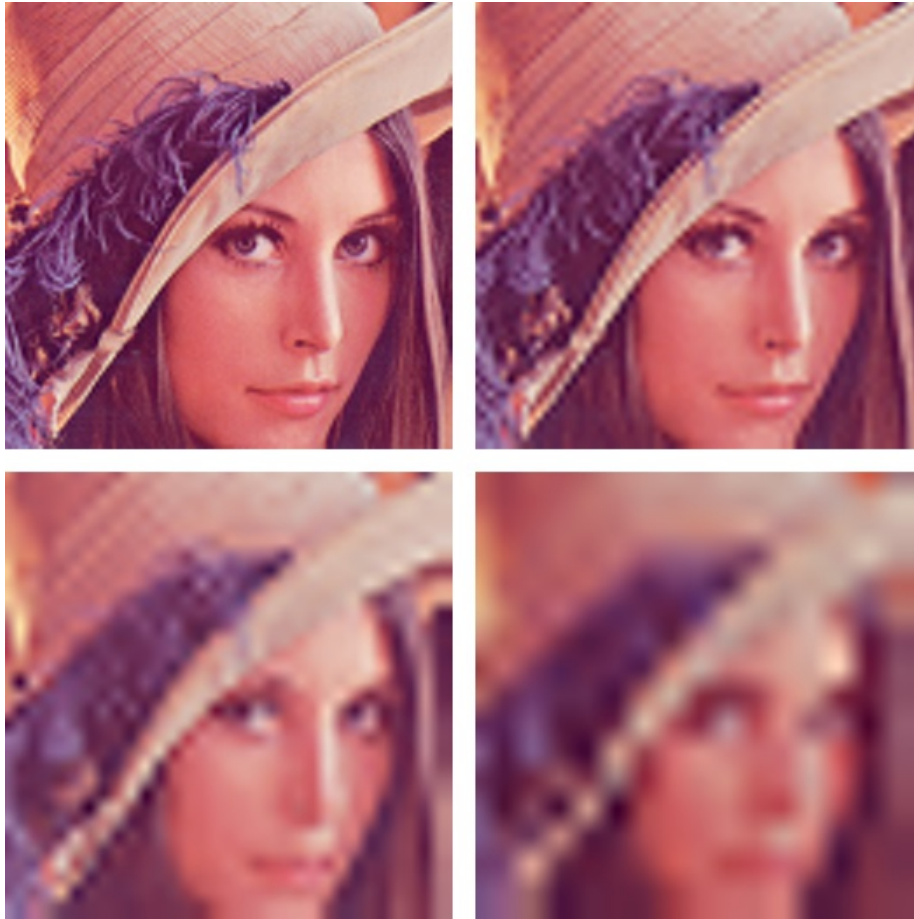


Figure 10.11: Low resolution approximations of the Lena image, for the CDF 9/7 wavelet.

functions should at each stage apply the kernel function  $\mathbf{f}$  to the appropriate rows and columns. If the image has several color components, the functions should be applied to each color component (there are three color components in the test image 'lena.png').

### Exercise 10.8: Comment code

Assume that we have an image represented by the  $M \times N$ -matrix  $\mathbf{X}$ , and consider the following code:

```
for n in range(N):
    c = (X[0:M:2, n] + X[1:M:2, n])/sqrt(2)
    w = (X[0:M:2, n] - X[1:M:2, n])/sqrt(2)
```



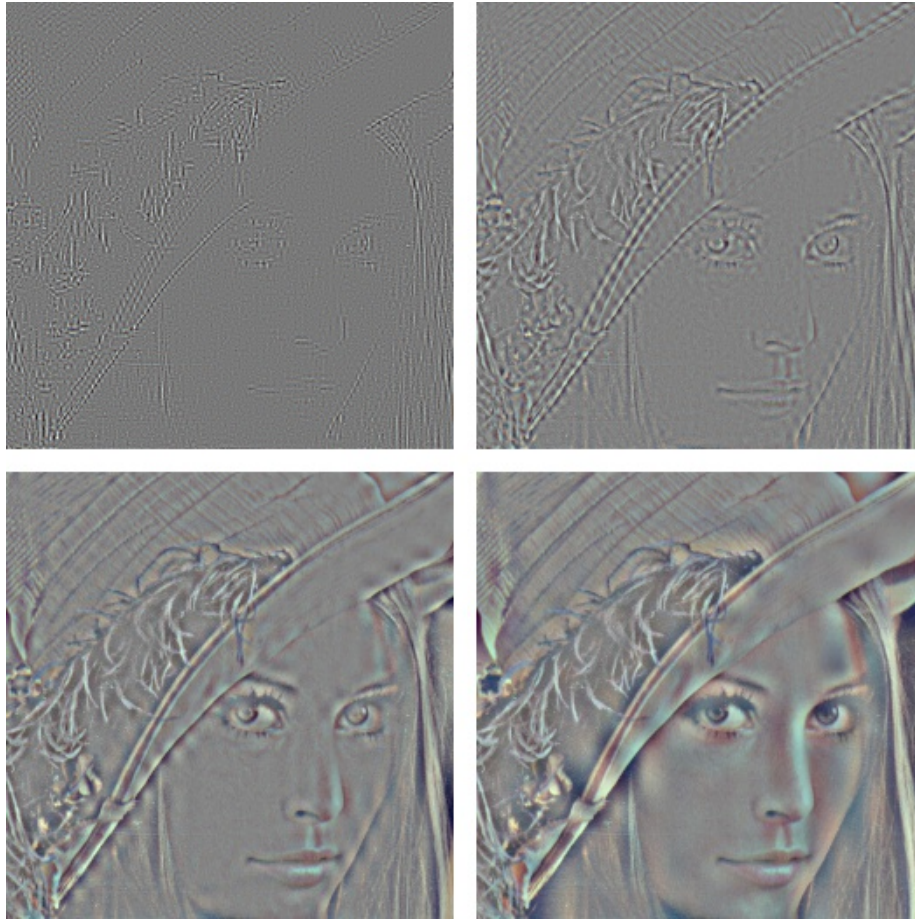


Figure 10.12: Detail of the Lena image, for the CDF 9/7 wavelet.

```

X[:, n] = concatenate([c, w])

for m in range(M):
    c = (X[m, 0:N:2] + X[m, 1:N:2])/sqrt(2)
    w = (X[m, 0:N:2] - X[m, 1:N:2])/sqrt(2)
    X[m, :] = concatenate([c,w])

```

- a) Comment what the code does, and explain what you will see if you display  $X$  as an image after the code has run.
- b) The code above has an inverse transformation, which reproduce the original image from the transformed values which we obtained. Assume that you zero out the values in the lower left and the upper right corner of the matrix  $X$  after the code above has run, and that you then reproduce the image by applying this inverse transformation. What changes can you then expect in the image?



Figure 10.13: Image of Lena, with various bands of detail at the first level zeroed out. From left to right, the detail at  $W_1^{(1,1)}$ ,  $W_1^{(1,0)}$ ,  $W_1^{(0,1)}$ , as illustrated in Figure 10.4. The Spline 5/3 wavelet was used.

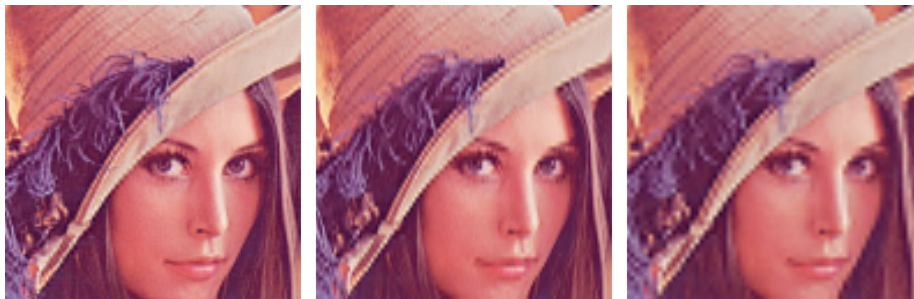


Figure 10.14: Image of Lena, with various bands of detail at the second level zeroed out. From left to right, the detail at  $W_2^{(1,1)}$ ,  $W_2^{(1,0)}$ ,  $W_2^{(0,1)}$ , as illustrated in Figure 10.5. The Spline 5/3 wavelet was used.

### Exercise 10.9: Comment code

In this exercise we will use the filters  $G_0 = \{1, 1\}$ ,  $G_1 = \{1, -1\}$ .

- Let  $X$  be a matrix which represents the pixel values in an image. Define  $\mathbf{x} = (1, 0, 1, 0)$  and  $\mathbf{y} = (0, 1, 0, 1)$ . Compute  $(G_0 \otimes G_0)(\mathbf{x} \otimes \mathbf{y})$ .
- For a general image  $X$ , describe how the images  $(G_0 \otimes G_0)X$ ,  $(G_0 \otimes G_1)X$ ,  $(G_1 \otimes G_0)X$ , and  $(G_1 \otimes G_1)X$  may look.
- Assume that we run the following code on an image represented by the matrix  $X$ :

```
M, N = shape(X)
for n in range(N):
    c = X[0:M:2, n] + X[1:M:2, n]
    w = X[0:M:2, n] - X[1:M:2, n]
    X[:, n] = vstack([c,w])

for m in range(M):
```

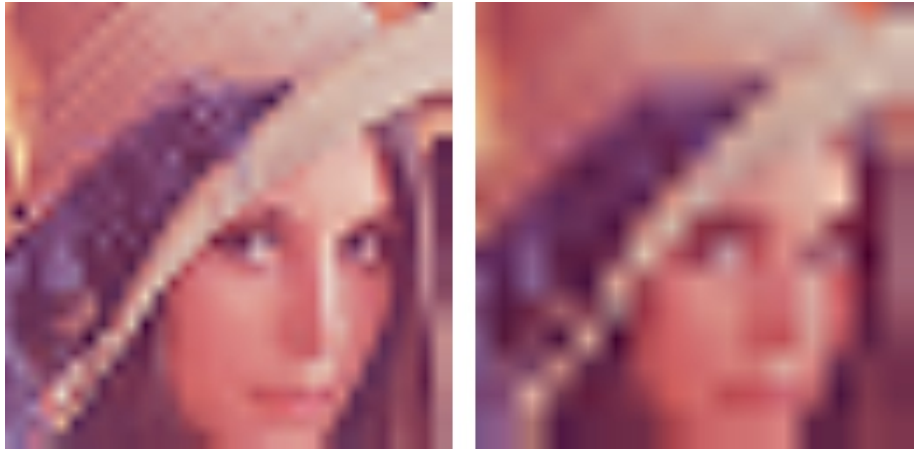


Figure 10.15: Image of Lena, with detail including level 3 and 4 zeroed out. The Spline 5/3 wavelet was used.

```
c = X[m, 0:N:2] + X[m, 1:N:2]
w = X[m, 0:N:2] - X[m, 1:N:2]
X[m, :] = hstack([c,w])
```

Comment the code. Describe what will be shown in the upper left corner of  $X$  after the code has run. Do the same for the lower left corner of the matrix. What is the connection with the images  $(G_0 \otimes G_0)X$ ,  $(G_0 \otimes G_1)X$ ,  $(G_1 \otimes G_0)X$ , and  $(G_1 \otimes G_1)X$ ?

### Exercise 10.10: Experiments on a test image

In Figure 10.17 we have applied the DWT2 with the Haar wavelet to an image very similar to the one you see in Figure 10.6. You see here, however, that there seems to be no detail components, which is very different from what you saw in Example 10.3, even though the images are very similar. Attempt to explain what causes this to happen.

**Hint.** Compare with Exercise 5.21.



Figure 10.16: The corresponding detail for the image of Lena. The Spline 5/3 wavelet was used.



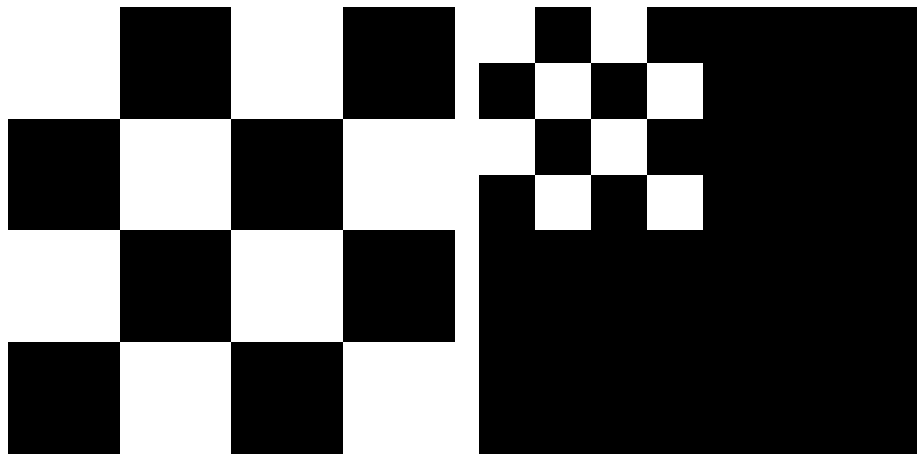


Figure 10.17: A simple image before and after one level of the DWT2. The Haar wavelet was used.

## 10.4 An application to the FBI standard for compression of fingerprint images

In the beginning of the 1990s, the FBI had a major problem when it came to their archive of fingerprint images. With more than 200 million fingerprint records, their digital storage exploded in size, so that some compression strategy needed to be employed. Several strategies were tried, for instance the widely adopted JPEG standard. The problem with JPEG had to do with the blocking artefacts, which we saw in Section 9.4. Among other strategies, FBI chose a wavelet-based strategy due to its nice properties. The particular way wavelets are applied in this strategy is called *Wavelet transform/scalar quantization* (WSQ).



Figure 10.18: A typical fingerprint image.

Fingerprint images are a very specific type of images, as seen in Figure 10.18. They differ from natural images by having a large number of abrupt changes. One may ask whether other wavelets than the ones we have used up to now are more suitable for compressing such images. After all, the technique of vanishing moments we have used for constructing wavelets are most suitable when the images display some regularity (as many natural images do). Extensive tests were undertaken to compare different wavelets, and the CDF 9/7 wavelet used by JPEG2000 turned out to perform very well, also for fingerprint images. One advantage with the choice of this wavelet for the FBI standard is that one then can exploit existing wavelet transformations from the JPEG2000 standard.

Besides the choice of wavelet, one can also ask other questions in the quest to compress fingerprint images: What number of levels is optimal in the application of the DWT2? And, while the levels in a DWT2 (see Figure 10.3) have an interpretation as change of coordinates, one can apply a DWT2 to the other subbands as well. This can not be interpreted as a change of coordinates, but if we assume that these subbands have the same characteristics as the original

image, the DWT2 will also help us with compression when applied to them. Let us illustrate how the FBI standard applies the DWT2 to the different subbands. We will split this process into five stages. The subband structures and the resulting images after stage 1-4 are illustrated in Figure 10.19 and in Figure 10.20, respectively.

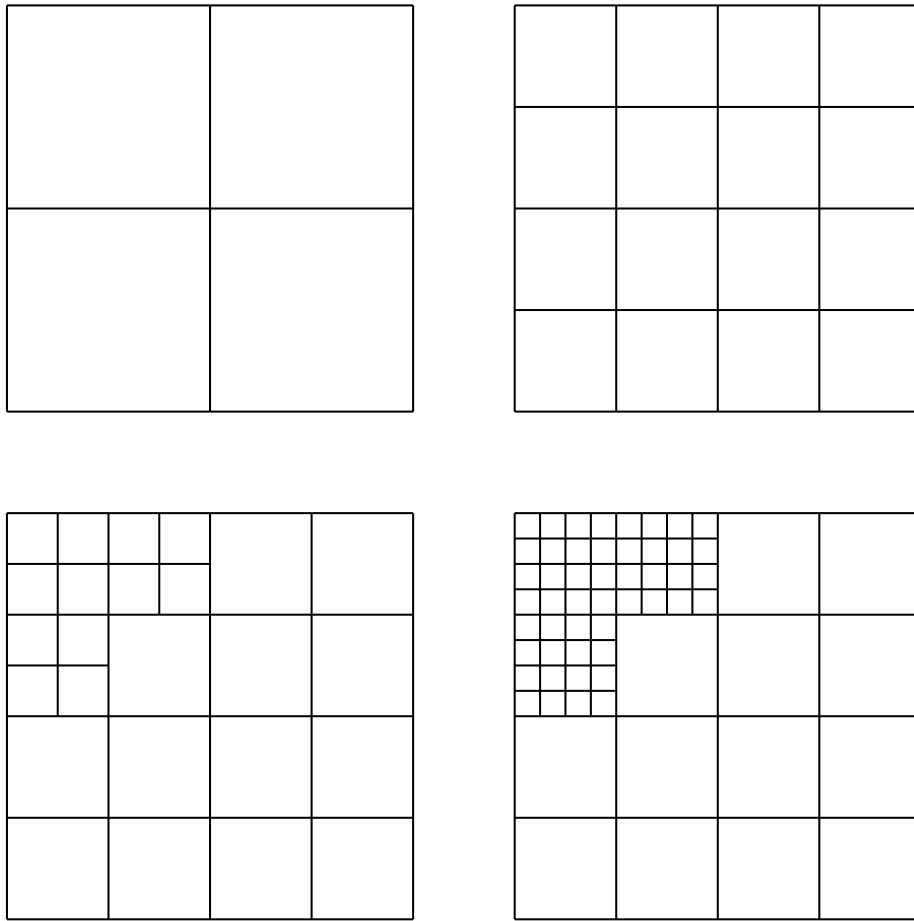


Figure 10.19: Subband structure after the different stages of the wavelet applications in the FBI fingerprint compression scheme.

1. First apply the first stage in a DWT2. This gives the upper left corners in the two figures.
2. Then apply a DWT2 to all four resulting subbands. This is different from the DWT2, which only continues on the upper left corner. This gives the upper right corners in the two figures.

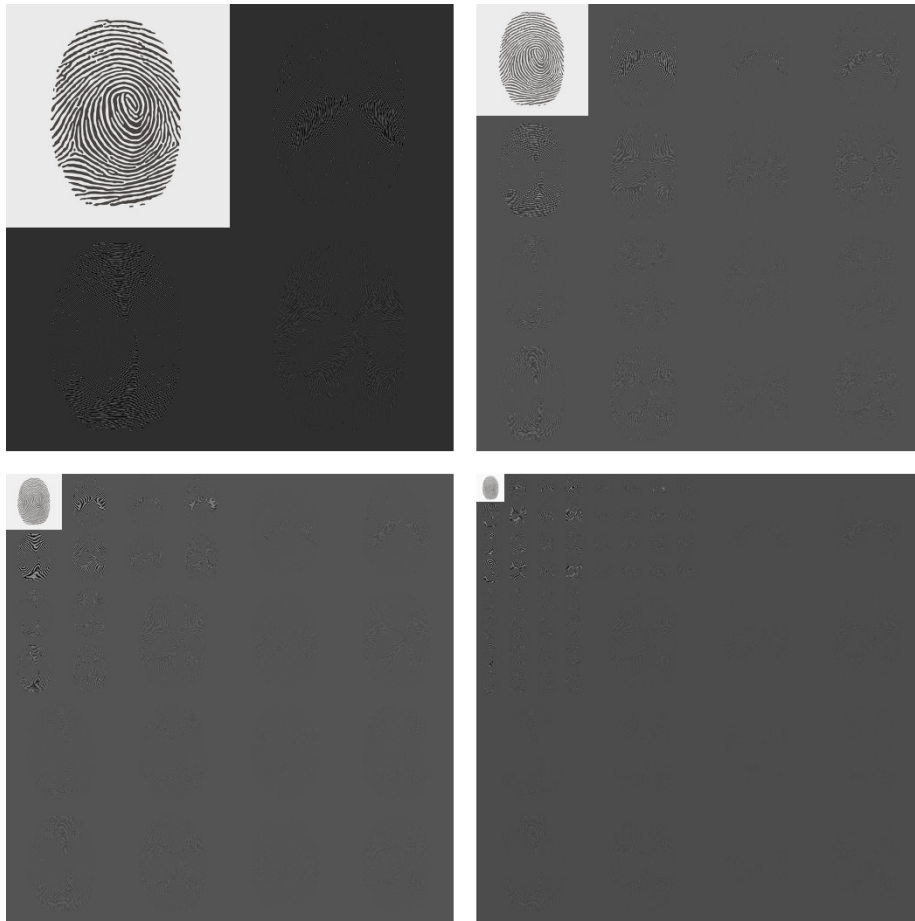


Figure 10.20: The fingerprint image after several DWT's.

3. Then apply a DWT2 in three of the four resulting subbands. This gives the lower left corners.
4. In all remaining subbands, the DWT2 is again applied. This gives the lower right corners.

Now for the last stage. A DWT2 is again applied, but this time only to the upper left corner. The subbands are illustrated in Figure 10.21, and in Figure 10.22 the resulting image is shown.

When establishing the standard for compression of fingerprint images, the FBI chose this subband decomposition. In Figure 10.23 we also show the corresponding low resolution approximation and detail.

As can be seen from the subband decomposition, the low-resolution approximation is simply the approximation after a five stage DWT2.

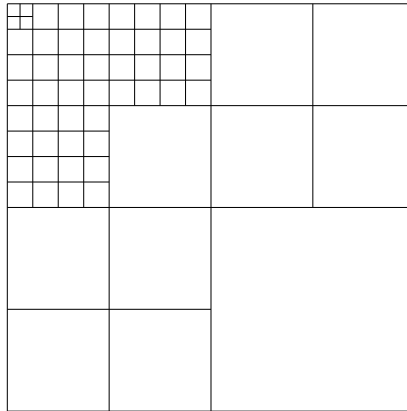


Figure 10.21: Subbands structure after all stages.



Figure 10.22: The resulting image obtained with the subband decomposition employed by the FBI.

The original JPEG2000 standard did not give the possibility for this type of subband decomposition. This has been added to a later extension of the standard, which makes the two standards more compatible. IN FBI's system, there are also other important parts besides the actual compression strategy, such as *fingerprint pattern matching*: In order to match a fingerprint quickly with the records in the database, several characteristics of the fingerprints are stored, such as the number of lines in the fingerprint, and points where the lines split or join. When the database is indexed with this information, one may not



Figure 10.23: The low-resolution approximation and the detail obtained by the FBI standard for compression of fingerprint images, when applied to our sample fingerprint image.

need to decompress all images in the database to perform matching. We will not go into details on this here.

### Exercise 10.11: Implement the fingerprint compression scheme

Write code which generates the images shown in figures 10.20, 10.22, and 10.23. Use the functions `DW2TImpl` and `IDW2TImpl` with the CDF 9/7 wavelet kernel functions as input.

## 10.5 Summary

We extended the tensor product construction to functions by defining the tensor product of functions as a function in two variables. We explained with some examples that this made the tensor product formalism useful for approximation of functions in several variables. We extended the wavelet transform to the tensor product setting, so that it too could be applied to images. We also performed several experiments on our test image, such as creating low-resolution images and neglecting wavelet coefficients. We also used different wavelets, such as the Haar wavelet, the Spline 5/3 wavelet, and the CDF 9/7 wavelet. The experiments confirmed what we previously have proved, that wavelets with many vanishing moments are better suited for compression purposes.

The specification of the JPGE2000 standard can be found in [21]. In [46], most details of this theory is covered, in particular details on how the wavelet coefficients are coded (which is not covered here).

One particular application of wavelets in image processing is the compression of fingerprint images. The standard which describes how this should be performed

can be found in [15]. In [4], the theory is described. The book [16] uses the application to compression of fingerprint images as an example of the usefulness of recent developments in wavelet theory.

**What you should have learned in this chapter.**

- The special interpretation of DWT2 applied to an image as splitting into four types of coordinates (each being one corner of the image), which represent lowpass/highpass combinations in the horizontal/vertical directions.
- How to call functions which perform different wavelet transformations on an image.
- Be able to interpret the detail components and low-resolution approximations in what you see.

# Chapter 11

## The basics and applications

The problem of minimizing a function of several variables, possibly subject to constraints on these variables, is what optimization is about. So the main problem is easy to state! And, more importantly, such problems arise in many applications in natural science, engineering, economics and business as well as in mathematics itself.

Nonlinear optimization differs from Fourier analysis and wavelet theory in that classical multivariate analysis also is an important ingredient. A recommended book on this, used here at the University of Oslo, is [26] (in Norwegian). It contains a significant amount of fixed point theory, nonlinear equations, and optimization.

There are many excellent books on nonlinear optimization (or nonlinear programming, as it is also called). Some of these books that have influenced these notes are [2, 3, 27, 19, 41, 32]. These are all recommended books for those who want to go deeper into the subject. These lecture notes are particularly influenced by the presentations in [2, 3].

Optimization has its mathematical foundation in linear algebra and multivariate calculus. In analysis the area of convexity is especially important. For the brief presentation of convexity given here the author's own lecture notes [11] (originally from 2001), and the very nice book [49], have been useful sources. But, of course, anyone who wants to learn convexity should study the work by R.T. Rockafellar, see e.g. the classic text [40].

Linear optimization (LP, linear programming) is a special case of nonlinear optimization, but we do not discuss this in any detail here. The reason for this is that we, at the University of Oslo, have a separate course in linear optimization which covers many parts of that subject in some detail.

This first chapter introduces some of the basic concepts in optimization and discusses some applications. Many of the ideas and results that you will find in these lecture notes may be extended to more general linear spaces, even infinite-dimensional. However, to keep life a bit easier and still cover most applications, we will only be working in  $\mathbb{R}^n$ .



Due to its character this chapter is a “proof-free zone”, but in the remaining text we usually give full proofs of the main results.

**Notation.** For  $\mathbf{z} \in \mathbb{R}^n$  and  $\delta > 0$  define the (closed) *ball*  $\bar{B}(\mathbf{z}; \epsilon) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{z}\| \leq \epsilon\}$ . It consists of all points with distance at most  $\epsilon$  from  $\mathbf{z}$ . Similarly, define the open ball  $B(\mathbf{z}; \epsilon) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{z}\| < \epsilon\}$ . A *neighborhood* of  $\mathbf{z}$  is a set  $N$  containing  $B(\mathbf{z}; \epsilon)$  for some  $\epsilon > 0$ . Vectors are treated as column vectors and they are identified with the corresponding  $n$ -tuple, denoted by  $\mathbf{x} = (x_1, x_2, \dots, x_n)$ . A statement like

$$P(\mathbf{x}) \quad (\mathbf{x} \in H)$$

means that the statement  $P(\mathbf{x})$  is true for all  $\mathbf{x} \in H$ .

## 11.1 The basic concepts

Optimization deals with finding optimal solutions! So we need to define what this is.

Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a real-valued function in  $n$  variables. The function value is written as  $f(\mathbf{x})$ , for  $\mathbf{x} \in \mathbb{R}^n$ , or  $f(x_1, x_2, \dots, x_n)$ . This is the function we want to minimize (or maximize) and it is often called the *objective function*. Let  $\mathbf{x}^* \in \mathbb{R}^n$ . Then  $\mathbf{x}^*$  is a *local minimum* (or local minimizer) of  $f$  if there is an  $\epsilon > 0$  such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in B(\mathbf{x}^*; \epsilon).$$

So, no point “sufficiently near”  $\mathbf{x}^*$  has smaller  $f$ -value than  $\mathbf{x}^*$ . A *local maximum* is defined similarly, but with the inequality reversed. A stronger notion is that  $\mathbf{x}^*$  is a *global minimum* of  $f$  which means that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathbb{R}^n.$$

A *global maximum* satisfies the opposite inequality.

The definition of local minimum has a “variational character”; it concerns the behavior of  $f$  near  $\mathbf{x}^*$ . Due to this it is perhaps natural that Taylor’s formula, which gives an approximation of  $f$  in such a neighborhood, becomes a main tool for characterizing and finding local minima. We present Taylor’s formula, in different versions, in Section 11.3.

An extension of the notion of minimum and maximum is for *constrained* problems where we want, for instance, to minimize  $f(\mathbf{x})$  over all  $\mathbf{x}$  lying in a given set  $C$ . Then  $\mathbf{x}^* \in C$  is a *local minimum* of  $f$  over the set  $C$ , or subject to  $\mathbf{x} \in C$  as we shall say, provided no point in  $C$  in some neighborhood of  $\mathbf{x}^*$  has smaller  $f$ -value than  $\mathbf{x}^*$ . A similar extension holds for global minimum over  $C$ , and for maxima.

**An example from plane geometry.** Consider the point set  $C = \{(x_1, x_2) : x_1 \geq 0, x_2 \geq 0, x_1 + x_2 \leq 1\}$  in the plane. We want to find a point  $\mathbf{x} = (x_1, x_2) \in C$  which is closest possible to the point  $\mathbf{a} = (3, 2)$ . This can be formulated as the minimization problem

$$\begin{aligned} & \text{minimize} && (x_1 - 3)^2 + (x_2 - 2)^2 \\ & \text{subject to} && \\ & && x_1 + x_2 \leq 1 \\ & && x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

The function we want to minimize is  $f(\mathbf{x}) = (x_1 - 3)^2 + (x_2 - 2)^2$  which is a quadratic function. This is the square of the distance between  $\mathbf{x}$  and  $\mathbf{a}$ ; and minimizing the distance or the square of the distance is equivalent (why?). A minimum here is  $\mathbf{x}^* = (1, 0)$ , as can be seen from a simple geometric argument where we draw the normal from  $(3, 2)$  to the line  $x_1 + x_2 = 1$ . If we instead minimize  $f$  over  $\mathbb{R}^2$ , the unique global minimum is clearly  $\mathbf{x}^* = \mathbf{a} = (3, 2)$ . It is also useful, and not too hard, to find these minima analytically.

In optimization one considers minimization and maximization problems. As

$$\max\{f(\mathbf{x}) : \mathbf{x} \in S\} = -\min\{-f(\mathbf{x}) : \mathbf{x} \in S\}$$

it is clear how to convert a maximization problem into a minimization problem (or vice versa). This transformation may, however, change the properties of the function you work with. For instance, if  $f$  is convex (definitions come later!), then  $-f$  is not convex (unless  $f$  is linear), so rewriting between minimization and maximization may take you out of a class of “good problems”. Note that a minimum or maximum may not exist. A main tool one uses to establish that optimal solutions really exist is the *extreme value theorem* as stated next. You may want to look these notions up in [26].

**Theorem 11.1.** *Continuous functions on closed and bounded sets.*

Let  $C$  be a subset of  $\mathbb{R}^n$  which is closed and bounded, and let  $f : C \rightarrow \mathbb{R}$  be a continuous function. Then  $f$  attains both its (global) minimum and maximum, so these are points  $\mathbf{x}^1, \mathbf{x}^2 \in C$  with

$$f(\mathbf{x}^1) \leq f(\mathbf{x}) \leq f(\mathbf{x}^2) \quad (\mathbf{x} \in C).$$

## 11.2 Some applications

It is useful to see some application areas for optimization. They are many, and here we mention a few in some detail. The methods we will learn later will be applied to these examples.

**Portfolio optimization.** The following optimization problem was introduced by Markowitz in order to find an optimal portfolio in a financial market; he later

received the Nobel prize in economics <sup>1</sup> (in 1990) for his contributions in this area:

$$\begin{aligned} & \text{minimize} && \alpha \sum_{i,j \leq n} c_{ij} x_i x_j - \sum_{j=1}^n \mu_j x_j \\ & \text{subject to} && \sum_{j=1}^n x_j = 1 \\ & && x_j \geq 0 \quad (j \leq n). \end{aligned}$$

The model may be understood as follows. The decision variables are  $x_1, x_2, \dots, x_n$  where  $x_i$  is the fraction of a total investment that is made in (say) stock  $i$ . Thus one has available a set of stocks in different companies (Statoil, IBM, Apple etc.) or bonds. The fractions  $x_i$  must be nonnegative (so we consider no short sale) and add up to 1. The function  $f$  to be minimized is

$$f(\mathbf{x}) = \alpha \sum_{i,j \leq n} c_{ij} x_i x_j - \sum_{j=1}^n \mu_j x_j.$$

It can be explained in terms of random variables. Let  $R_j$  be the return on stock  $j$ , this is a random variable, and let  $\mu_j = \mathbb{E}R_j$  be the expectation of  $R_j$ . So if  $X$  denotes the random variable  $X = \sum_{j=1}^n x_j R_j$ , which is the return on our portfolio (= mix among investments), then  $\mathbb{E}X = \sum_{j=1}^n \mu_j x_j$  which is the second term in  $f$ . The minus sign in front explains that we really want to maximize the expected return. The first term in  $f$  is there because just looking at expected return is too simple. We want to spread our investments to reduce the risk. The first term in  $f$  is the variance of  $X$  multiplied by a weight factor  $\alpha$ ; the constant  $c_{ij}$  is the covariance of  $R_i$  and  $R_j$ , defined by

$$c_{ij} = \mathbb{E}(R_i - \mu_i)(R_j - \mu_j).$$

$c_{ii}$  is also called the variance of  $R_i$ .

So  $f$  is a weighted difference of variance and expected return. This is what we want to minimize. The optimization problem is to minimize a quadratic function subject to linear constraints. We shall discuss theory and methods for such problems later.

In order to use such a model one needs to find good values for all the parameters  $\mu_j$  and  $c_{ij}$ ; this is done using historical data from the stock markets. The weight parameter  $\alpha$  is often varied and the optimization problem is solved for each such "interesting" value. This makes it possible to find a so-called efficient frontier of expectation versus variance for optimal solutions.

The Markowitz model is a useful tool for financial investments, and now extensions and variations of the model exist, e.g., by using different ways of measuring risk. All such models involve a balance between risk and expected return.

---

<sup>1</sup>The precise term is "Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel".

**Fitting a model.** In many applications one has a mathematical model of some phenomenon where the model has some parameters. These parameters represent a flexibility of the model, and they may be adjusted so that the model explains the phenomenon best possible.

To be more specific consider a model

$$y = F_\alpha(\mathbf{x})$$

for some function  $F_\alpha : \mathbb{R}^m \rightarrow \mathbb{R}$ . Here  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n) \in \mathbb{R}^n$  is a parameter vector (so we may have several parameters). Perhaps there are natural constraints on the parameter, say  $\alpha \in A$  for a given set  $A$  in  $\mathbb{R}^n$ .

For instance, consider

$$y = \alpha_1 \cos x_1 + x_2^{\alpha_2}$$

so here  $n = m = 2$ ,  $\alpha = (\alpha_1, \alpha_2)$  and  $F_\alpha(\mathbf{x}) = \alpha_1 \cos x_1 + x_2^{\alpha_2}$  where (say)  $\alpha_1 \in \mathbb{R}$  and  $\alpha_2 \in [1, 2]$ .

The general model may also be thought of as

$$y = F_\alpha(\mathbf{x}) + \text{error}$$

since it is usually a simplification of the system one considers. In statistics one specifies this error term as a random variable with some (partially) known distribution. Sometimes one calls  $y$  the *dependent variable* and  $\mathbf{x}$  the *explaining variable*. The goal is to understand how  $y$  depends on  $\mathbf{x}$ .

To proceed, assume we are given a number of observations of the phenomenon given by points

$$(\mathbf{x}^i, y^i) \quad (i = 1, 2, \dots, m).$$

meaning that one has observed  $y^i$  corresponding to  $\mathbf{x} = \mathbf{x}^i$ . We have  $m$  such observations. Usually (but not always) we have  $m \geq n$ . The *model fit problem* is to adjust the parameter  $\alpha$  so that the model fits the given data as good as possible. This leads to the optimization problem

$$\text{minimize } \sum_{i=1}^m (y^i - F_\alpha(\mathbf{x}^i))^2 \text{ subject to } \alpha \in A.$$

The optimization variable is the parameter  $\alpha$ . Here the model error is quadratic (corresponding to the Euclidean norm), but other norms are also used.

This optimization problem above is a constrained nonlinear optimization problem. When the function  $F_\alpha$  depends linearly on  $\alpha$ , which often is the case in practice, the problem becomes the classical *least squares approximation problem* which is treated in basic linear algebra courses. The solution is then characterized by a certain linear system of equations, the so-called normal equations.

**Maximum likelihood.** A very important problem in statistics, arising in many applications, is parameter estimation and, in particular, *maximum likelihood estimation*. It leads to optimization.

Let  $X$  be a “continuous” real-valued random variable with probability density  $p(x; \alpha)$ . Here  $\alpha$  is a parameter (often one uses other symbols for the parameter,

like  $\xi$ ,  $\theta$  etc.). For instance, if  $X$  is a normal (Gaussian) variable with expectation  $\alpha$  and variance 1, then  $p(x; \alpha) = \frac{1}{\sqrt{2\pi}} e^{-(x-\alpha)^2/2}$  and

$$\mathbb{P}(a \leq X \leq b) = \int_a^b \frac{1}{\sqrt{2\pi}} e^{-(x-\alpha)^2/2} dx$$

where  $\mathbb{P}$  denotes probability.

Assume  $X$  is the outcome of an experiment, and that we have observed  $X = x$  (so  $x$  is a known real number or a vector, if several observations were made). On the basis of  $x$  we want to *estimate* the value of the parameter  $\alpha$  which “explains” best possible our observation  $X = x$ . We have now available the probability density  $p(x; \cdot)$ . The function  $\alpha \rightarrow p(x; \alpha)$ , for fixed  $x$ , is called the *likelihood* function. It gives the “probability mass” in  $x$  as a function of the parameter  $\alpha$ . The *maximum likelihood* problem is to find a parameter value  $\alpha$  which maximizes the likelihood, i.e., which maximizes the probability of getting precisely  $y$ . This is an optimization problem

$$\max_{\alpha} p(x; \alpha)$$

where  $x$  is fixed and the optimization variable is  $\alpha$ . We may here add a constraint on  $\alpha$ , say  $\alpha \in C$  for some set  $C$ , which may incorporate possible knowledge of  $\alpha$  and assure that  $p(x; \alpha)$  is positive for  $\alpha \in C$ . Often it is easier to solve the equivalent optimization problem of maximizing the logarithm of the likelihood function

$$\max_{\alpha} \ln p(x; \alpha)$$

This is a nonlinear optimization problem. Often, in statistics, there are several parameters, so  $\alpha \in \mathbb{R}^n$  for some  $n$ , and we need to solve a nonlinear optimization problem in several variables, possibly with constraints on these variables. If the likelihood function, or its logarithm, is a concave function, we have (after multiplying by  $-1$ ) a convex optimization problem. Such problems are easier to solve than general optimization problems. This will be discussed later.

As a specific example assume we have the linear statistical model

$$\mathbf{x} = A\boldsymbol{\alpha} + \mathbf{w}$$

where  $A$  is a given  $m \times n$  matrix,  $\boldsymbol{\alpha} \in \mathbb{R}^n$  is an unknown parameter,  $\mathbf{w} \in \mathbb{R}^m$  is a random variable (the “noise”), and  $\mathbf{x} \in \mathbb{R}^m$  is the observed quantity. We assume that the components of  $\mathbf{w}$ , i.e.,  $w_1, w_2, \dots, w_m$  are independent and identically distributed with common density function  $p$  on  $\mathbb{R}$ . This leads to the likelihood function

$$p(\mathbf{x}; \boldsymbol{\alpha}) = \prod_{i=1}^m p(x_i - \mathbf{a}_i \boldsymbol{\alpha})$$

where  $\mathbf{a}_i$  is the  $i$ 'th row in  $A$ . Taking the logarithm we obtain the maximum likelihood problem

$$\max \sum_{i=1}^m \ln p(x_i - \mathbf{a}_i \boldsymbol{\alpha}).$$

In many applications of statistics it is central to solve this optimization problem numerically.

Let us take a look at a model taken from physics for desintegration of muons. The angle  $\theta$  in electron radiation for desintegration of muons has a probability density

$$p(x; \alpha) = \frac{1 + \alpha x}{2} \quad (11.1)$$

for  $x \in [-1, 1]$ , where  $x = \cos \theta$ , and where  $\alpha$  is an unknown parameter in  $[-1, 1]$ . Our goal is to estimate  $\alpha$  from  $n$  measurements  $\mathbf{x} = (x_1, \dots, x_n)$ . In this case the likelihood function, which we seek to maximize, takes the form  $g(\alpha) = \prod_{i=1}^n p(x_i; \alpha)$ . Taking logarithms and multiplying by  $-1$ , our problem is to minimize

$$f(\alpha) = -\ln g(\alpha) = -\ln \left( \prod_{i=1}^n p(x_i; \alpha) \right) = -\sum_{i=1}^n \ln((1 + \alpha x_i)/2). \quad (11.2)$$

We compute

$$f'(\alpha) = -\sum_{i=1}^n \frac{x_i/2}{(1 + \alpha x_i)/2} = -\sum_{i=1}^n \frac{x_i}{1 + \alpha x_i}$$

$$f''(\alpha) = \sum_{i=1}^n \frac{x_i^2}{(1 + \alpha x_i)^2}$$

We see that  $f''(\alpha) \geq 0$ , so that  $f$  is convex. As explained, this will make the problem easier to solve using numerical methods. If we try to solve  $f'(\alpha) = 0$  we will run into problems, however. We see, however, that  $f'(\alpha) \rightarrow 0$  when  $\alpha \rightarrow \pm\infty$ , and since  $\frac{x_i}{1 + \alpha x_i} = \frac{1}{1/x_i + \alpha}$ , we must have that  $f'(\alpha) \rightarrow \infty$  when  $\alpha \rightarrow -1/x_i$  from below, and  $f'(\alpha) \rightarrow -\infty$  when  $\alpha \rightarrow -1/x_i$  from above. It is therefore clear that  $f$  has exactly one minimum in every interval of the form  $[-1/x_i, -1/x_{i+1}]$  when we list the  $x_i$  in increasing order. It is not for sure that there is a minimum within  $[-1, 1]$  at all. If all measurements have the same sign we are guaranteed to find no such point. In this case the minimum must be one of the end points in the interval. We will later look into numerical method for finding this minimum.

**Optimal control problems.** Recall that a discrete dynamical system is an equation

$$\mathbf{x}_{t+1} = h_t(\mathbf{x}_t) \quad (t = 0, 1, \dots)$$

where  $\mathbf{x}_t \in \mathbb{R}^n$ ,  $\mathbf{x}_0$  is the initial solution, and  $h_t$  is a given function for each  $t$ . We here think of  $t$  as time and  $\mathbf{x}_t$  is the state of the process at time  $t$ . For instance, let  $n = 1$  and consider  $h_t(\mathbf{x}) = a\mathbf{x}$  ( $t = 0, 1, \dots$ ) for some  $a \in \mathbb{R}$ . Then the solution is  $\mathbf{x}_t = a^t \mathbf{x}_0$ . Another example is when  $A$  is an  $n \times n$  matrix,  $\mathbf{x}_t \in \mathbb{R}^n$  and  $h_t(\mathbf{x}) = A\mathbf{x}$  for each  $t$ . Then the solution is  $\mathbf{x}_t = A^t \mathbf{x}_0$ . For the more general situation, where the system functions  $h_t$  may be different, it may be difficult to find an explicit solution for  $\mathbf{x}_t$ . Numerically, however, we compute  $\mathbf{x}_t$  simply in a for-loop by computing  $\mathbf{x}_0$ , then  $\mathbf{x}_1 = f_1(\mathbf{x}_0)$  and then  $\mathbf{x}_2 = f_2(\mathbf{x}_1)$  etc.

Now, consider a dynamical system where we may “control” the system in each time step. We restrict the attention to a finite time span,  $t = 0, 1, \dots, T$ . A proper model is then

$$\mathbf{x}_{t+1} = h_t(\mathbf{x}_t, \mathbf{u}_t) \quad (t = 0, 1, \dots, T-1)$$

where  $\mathbf{x}_t$  is the state of the system at time  $t$  and the new variable  $\mathbf{u}_t$  is the control at time  $t$ . We assume  $\mathbf{x}_t \in \mathbb{R}^n$  and  $\mathbf{u}_t \in \mathbb{R}^m$  for each  $t$  (but these things also work if these vectors lie in spaces of different dimensions). Thus, when we choose the controls  $\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1}$  and  $\mathbf{x}_0$  is known, the sequence  $\{\mathbf{x}_t\}$  of states is uniquely determined. Next, assume there are given functions  $f_t : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$  that we call cost functions. We think of  $f_t(\mathbf{x}_t, \mathbf{u}_t)$  as the “cost” at time  $t$  when the system is in state  $\mathbf{x}_t$  and we choose control  $\mathbf{u}_t$ . The *optimal control* problem is

$$\begin{aligned} & \text{minimize} && f_T(\mathbf{x}_T) + \sum_{t=0}^{T-1} f_t(\mathbf{x}_t, \mathbf{u}_t) \\ & \text{subject to} && \mathbf{x}_{t+1} = h_t(\mathbf{x}_t, \mathbf{u}_t) \quad (t = 0, 1, \dots, T-1) \end{aligned} \quad (11.3)$$

where the control is the sequence  $(\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1})$  to be determined. This problem arises in many applications, in engineering, finance, economics etc. We now rewrite this problem. First, let  $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_T) \in \mathbb{R}^N$  where  $N = Tn$ . Since, as we noted,  $\mathbf{x}_t$  is uniquely determined by  $\mathbf{u}$ , there is a function  $\mathbf{v}_t$  such that  $\mathbf{x}_t = \mathbf{v}_t(\mathbf{u})$  ( $t = 1, 2, \dots, T$ );  $\mathbf{x}_0$  is given. Therefore the total cost may be written

$$f_T(\mathbf{x}_T) + \sum_{t=0}^{T-1} f_t(\mathbf{x}_t, \mathbf{u}_t) = f_T(\mathbf{v}_T(\mathbf{u})) + \sum_{t=0}^{T-1} f_t(\mathbf{v}_t(\mathbf{u}), \mathbf{u}_t) := f(\mathbf{u})$$

which is a function of  $\mathbf{u}$ . Thus, we see that the optimal control problem may be transformed to the unconstrained optimization problem

$$\min_{\mathbf{u} \in \mathbb{R}^N} f(\mathbf{u})$$

Sometimes there may be constraints on the control variables, for instance that they each lie in some interval, and then the transformation above results in a constrained optimization problem.

**Linear optimization.** This is not an application, but rather a special case of the general nonlinear optimization problem where all functions are linear. A *linear optimization* problem, also called *linear programming*, has the form

$$\text{minimize } \mathbf{c}^T \mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}. \quad (11.4)$$

Here  $A$  is an  $m \times n$  matrix,  $\mathbf{b} \in \mathbb{R}^m$  and  $\mathbf{x} \geq \mathbf{0}$  means that  $x_i \geq 0$  for each  $i \leq n$ . So in linear optimization one minimizes (or maximizes) a linear function subject to linear equations and nonnegativity on the variables. Actually, one can show any problem with constraints that are linear equations and/or linear inequalities may be transformed into the form above. Such problems have a wide range of application in science, engineering, economics, business etc. Applications include portfolio optimization and many planning problems for e.g. production, transportation etc. Some of these problems are of a combinatorial nature, but linear optimization is a main tool here as well.

We shall not treat linear optimization in detail here since this is the topic of a separate course, MAT-INF3100 Linear optimization. In that course one presents some powerful methods for such problems, the simplex algorithm and interior point methods. In addition one considers applications in network flow models and game theory.

### 11.3 Multivariate calculus and linear algebra

We first recall some useful facts from linear algebra.

The *spectral theorem* says that if  $A$  is a real symmetric matrix, then there is an orthogonal matrix  $P$  (i.e., its columns are orthonormal) and a diagonal matrix  $D$  such that  $A = PDP^T$ . The diagonal of  $D$  contains the eigenvalues of  $A$ , and  $A$  has an orthonormal set of eigenvectors (the columns of  $P$ ).

A real symmetric matrix is positive semidefinite<sup>2</sup> if  $\mathbf{x}^T A \mathbf{x} \geq 0$  for all  $\mathbf{x} \in \mathbb{R}^n$ . The following statements are equivalent

1.  $A$  is positive semi-definite
2. all eigenvalues of  $A$  are nonnegative
3.  $A = W^T W$  for some matrix  $W$ .

Similarly, a real symmetric matrix is *positive definite* if  $\mathbf{x}^T A \mathbf{x} > 0$  for all nonzero  $\mathbf{x} \in \mathbb{R}^n$ . The following statements are equivalent.

1.  $A$  is positive definite
2. all eigenvalues of  $A$  are positive
3.  $A = W^T W$  for some invertible matrix  $W$ .

---

<sup>2</sup>See Section 7.2 in [25].



Every positive definite matrix is therefore invertible.

We also recall some central facts from multivariate calculus. They will be used repeatedly in these notes. Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a real-valued function defined on  $\mathbb{R}^n$ . The *gradient* of  $f$  at  $\mathbf{x}$  is the  $n$ -tuple

$$\nabla f(\mathbf{x}) = \left( \frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right).$$

We will always identify an  $n$ -tuple with the corresponding column vector<sup>3</sup>. Of course, the gradient only exists if all the partial derivatives exist. Second order information is contained in a matrix: assuming  $f$  has second order partial derivatives we define the *Hessian* matrix<sup>4</sup>  $\nabla^2 f(\mathbf{x})$  as the  $n \times n$  matrix whose  $(i, j)$ 'th entry is

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}.$$

If these second order partial derivatives are continuous, then we may switch the order in the derivations, and  $\nabla^2 f(\mathbf{x})$  is a symmetric matrix.

For vector-valued functions we also need the derivative. Consider the vector-valued function  $F$  given by

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} F_1(\mathbf{x}) \\ F_2(\mathbf{x}) \\ \vdots \\ F_n(\mathbf{x}) \end{bmatrix}$$

so  $F_i : \mathbb{R}^n \rightarrow \mathbb{R}$  is the  $i$ th component function of  $\mathbf{F}$ .  $\mathbf{F}'$  denotes the *Jacobi* matrix<sup>5</sup>, or simply the *derivative*, of  $\mathbf{F}$

$$\mathbf{F}'(x) = \begin{bmatrix} \frac{\partial F_1(\mathbf{x})}{\partial x_1} & \frac{\partial F_1(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial F_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial F_2(\mathbf{x})}{\partial x_1} & \frac{\partial F_2(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial F_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n(\mathbf{x})}{\partial x_1} & \frac{\partial F_n(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial F_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

The  $i$ th row of this matrix is therefore the gradient of  $F_i$ , now viewed as a row vector.

Next we recall Taylor's theorems from multivariate calculus<sup>6</sup>:

**Theorem 11.2.** *First order Taylor theorem.*

<sup>3</sup>This is somewhat different from [26], since the gradient there is always considered as a row vector.

<sup>4</sup>See Section 5.9 in [26].

<sup>5</sup>See Section 2.6 in [26].

<sup>6</sup>This theorem is also the mean value theorem of functions in several variables, see Section 5.5 in [26].

Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a function having continuous partial derivatives in some ball  $B(\mathbf{x}; r)$ . Then, for each  $\mathbf{h} \in \mathbb{R}^n$  with  $\|\mathbf{h}\| < r$  there is some  $t \in (0, 1)$  such that

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x} + t\mathbf{h})^T \mathbf{h}.$$

The next one is known as Taylor's formula, or the second order Taylor's theorem<sup>7</sup>:

**Theorem 11.3.** *Second order Taylor theorem.*

Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a function having second order partial derivatives that are continuous in some ball  $B(\mathbf{x}; r)$ . Then, for each  $\mathbf{h} \in \mathbb{R}^n$  with  $\|\mathbf{h}\| < r$  there is some  $t \in (0, 1)$  such that

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x} + t\mathbf{h}) \mathbf{h}.$$

This may be shown by considering the one-variable function  $g(t) = f(\mathbf{x} + t\mathbf{h})$  and applying the chain rule and Taylor's formula in one variable.

There is another version of the second order Taylor theorem in which the Hessian is evaluated in  $\mathbf{x}$  and, as a result, we get an error term. This theorem shows how  $f$  may be approximated by a quadratic polynomial in  $n$  variables<sup>8</sup>:

**Theorem 11.4.** *Second order Taylor theorem, version 2.*

Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a function having second order partial derivatives that are continuous in some ball  $B(\mathbf{x}; r)$ . Then there is a function  $\epsilon : \mathbb{R}^n \rightarrow \mathbb{R}$  such that, for each  $\mathbf{h} \in \mathbb{R}^n$  with  $\|\mathbf{h}\| < r$ ,

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h} + \epsilon(\mathbf{h}) \|\mathbf{h}\|^2.$$

Here  $\epsilon(\mathbf{y}) \rightarrow \mathbf{0}$  when  $\mathbf{y} \rightarrow \mathbf{0}$ .

The first and second order Taylor approximations can thus be summarized as follows:

$$\begin{aligned} f(\mathbf{x} + \mathbf{h}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + O(\|\mathbf{h}\|) \\ f(\mathbf{x} + \mathbf{h}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h} + O(\|\mathbf{h}\|^2). \end{aligned}$$

We introduce the following notation for the approximations

$$\begin{aligned} T_f^1(\mathbf{x}; \mathbf{x} + \mathbf{h}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} \\ T_f^2(\mathbf{x}; \mathbf{x} + \mathbf{h}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h}. \end{aligned}$$

<sup>7</sup>See Section 5.9 in [26].

<sup>8</sup>See Section 5.9 in [26].

As we shall see, one can get a lot of optimization out of these approximations!

We also need a Taylor theorem for vector-valued functions, which follows by applying Taylor' theorem above to each component function:

**Theorem 11.5.** *First order Taylor theorem for vector-valued functions.*

Let  $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  be a vector-valued function which is continuously differentiable in a neighborhood  $N$  of  $\mathbf{x}$ . Then

$$\mathbf{F}(\mathbf{x} + \mathbf{h}) = \mathbf{F}(\mathbf{x}) + \mathbf{F}'(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|)$$

when  $\mathbf{x} + \mathbf{h} \in N$ .

Finally, if  $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  and  $\mathbf{G} : \mathbb{R}^k \rightarrow \mathbb{R}^n$  the *composition*  $\mathbf{H}(\mathbf{x}) = \mathbf{F}(\mathbf{G}(\mathbf{x}))$  is a function from  $\mathbb{R}^k$  to  $\mathbb{R}^m$ . Under the natural differentiability assumptions the following chain rule<sup>9</sup> holds:

$$\mathbf{H}'(\mathbf{x}) = \mathbf{F}'(\mathbf{G}(\mathbf{x}))\mathbf{G}'(\mathbf{x}).$$

Here the right-hand side is a product of two matrices, the respective Jacobi matrices evaluated in the right points.

Finally, we discuss some notions concerning the convergence of sequences.

**Definition 11.6.** *Linear convergence.*

We say that a sequence  $\{\mathbf{x}_k\}_{k=1}^{\infty}$  converges to  $\mathbf{x}^*$  *linearly* (or that the convergence speed is linear) if there is a  $\gamma < 1$  such that

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \gamma \|\mathbf{x}_k - \mathbf{x}^*\| \quad (k = 0, 1, \dots).$$

A faster convergence rate is *superlinear convergence* which means that

$$\lim_{k \rightarrow \infty} \|\mathbf{x}_{k+1} - \mathbf{x}^*\| / \|\mathbf{x}_k - \mathbf{x}^*\| = 0$$

A special type of superlinear convergence is *quadratic convergence* where

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \gamma \|\mathbf{x}_k - \mathbf{x}^*\|^2 \quad (k = 0, 1, \dots)$$

for some  $\gamma < 1$ .

### Exercise 11.1: Solve

Give an example of a function  $f : \mathbb{R} \rightarrow \mathbb{R}$  with 10 global minima.

### Exercise 11.2: Solve

Consider the function  $f(x) = x \sin(1/x)$  defined for  $x > 0$ . Find its local minima. What about global minimum?

<sup>9</sup>See Section 2.7 in [26].

**Exercise 11.3: Solve**

Let  $f : X \rightarrow \mathbb{R}_+$  be a function (with nonnegative function values). Explain why it is equivalent to minimize  $f$  over  $x \in X$  or minimize  $f^2(x)$  over  $X$ .

**Exercise 11.4: Solve**

In Example 11.2 we mentioned that optimizing the function  $p_x(y)$  is equivalent to optimizing the function  $\ln p_x(y)$ . Explain why maximizing/minimizing  $g$  is the same as maximizing/minimizing  $\ln g$  for any positive function  $g$ .

**Exercise 11.5: Solve**

Consider  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  given by  $f(\mathbf{x}) = (x_1 - 3)^2 + (x_2 - 2)^2$ . How would you explain to *anyone* that  $\mathbf{x}^* = (3, 2)$  is a minimum point?

**Exercise 11.6: Level sets**

The *level sets* of a function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  are sets of the form  $L_\alpha = \{\mathbf{x} \in \mathbb{R}^2 : f(\mathbf{x}) = \alpha\}$ . Let  $f(\mathbf{x}) = \frac{1}{4}(x_1 - 1)^2 + (x_2 - 3)^2$ . Draw the level sets in the plane for  $\alpha = 10, 5, 1, 0.1$ .

**Exercise 11.7: Sub-level sets**

The *sub-level set* of a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is the set  $S_\alpha(f) = \{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) \leq \alpha\}$ , where  $\alpha \in \mathbb{R}$ . Assume that  $\inf\{f(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n\} = \eta$  exists.

- What happens to the sub-level sets  $S_\alpha$  as  $\alpha$  decreases? Give an example.
- Show that if  $f$  is continuous and there is an  $\mathbf{x}'$  such that with  $\alpha = f(\mathbf{x}')$  the sub-level set  $S_\alpha(f)$  is bounded, then  $f$  attains its minimum.

**Exercise 11.8: Portfolio optimization**

Consider the portfolio optimization problem in Section 11.2.

- Assume that  $c_{ij} = 0$  for each  $i \neq j$ . Find, analytically, an optimal solution. Describe the set of all optimal solutions.
- Consider the special case where  $n = 2$ . Solve the problem and discuss how minimum point depends on  $\alpha$ .

**Hint.** Eliminate one variable.

**Exercise 11.9: Solve**

Later in these notes we will need the expression for the gradient of functions which are expressed in terms of matrices.

- a) Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be defined by  $f(\mathbf{x}) = \mathbf{q}^T \mathbf{x} = \mathbf{x}^T \mathbf{q}$ , where  $\mathbf{q}$  is a vector. Show that  $\nabla f(\mathbf{x}) = \mathbf{q}$ , and that  $\nabla^2 f(\mathbf{x}) = \mathbf{0}$ .
- b) Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be the quadratic function  $f(\mathbf{x}) = (1/2)\mathbf{x}^T A \mathbf{x}$ , where  $A$  is symmetric. Show that  $\nabla f(\mathbf{x}) = A\mathbf{x}$ , and that  $\nabla^2 f(\mathbf{x}) = A$ .
- c) Show that, with  $f$  defined as in b., but with  $A$  not symmetric, we obtain that  $\nabla f(\mathbf{x}) = \frac{1}{2}(A + A^T)\mathbf{x}$ , and  $\nabla^2 f = \frac{1}{2}(A + A^T)$ . Verify that these formulas are compatible with what you found in b. when  $A$  is symmetric.

**Exercise 11.10: Solve**

Consider  $f(\mathbf{x}) = f(x_1, x_2) = x_1^2 + 3x_1x_2 - 5x_2^2 + 3$ . Determine the first order Taylor approximation to  $f$  at each of the points  $(0, 0)$  and  $(2, 1)$ .

**Exercise 11.11: Solve**

Let  $A = \begin{pmatrix} 1 & 2 \\ 2 & 8 \end{pmatrix}$ . Show that  $A$  is positive definite. (Try to give two different proofs.)

**Exercise 11.12: Solve**

Show that if  $A$  is positive definite, then its inverse is also positive definite.

## Chapter 12

# A crash course in convexity

Convexity is a branch of mathematical analysis dealing with convex sets and convex functions. It also represents a foundation for optimization.

We just summarize concepts and some results. For proofs one may consult [11] or [49], see also [2].

### 12.1 Convex sets

A set  $C \subseteq \mathbb{R}^n$  is called *convex* if  $(1 - \lambda)\mathbf{x} + \lambda\mathbf{y} \in C$  whenever  $\mathbf{x}, \mathbf{y} \in C$  and  $0 \leq \lambda \leq 1$ . Geometrically, this means that  $C$  contains the line segment between each pair of points in  $C$ , so, loosely speaking, a convex set contains no “holes”.

For instance, the ball  $B(\mathbf{a}; \delta) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{a}\| \leq \delta\}$  is a convex set. Let us show this. Recall the triangle inequality which says that  $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$  whenever  $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ . Let  $\mathbf{x}, \mathbf{y} \in B(\mathbf{a}; \delta)$  and  $\lambda \in [0, 1]$ . Then

$$\begin{aligned} \|((1 - \lambda)\mathbf{x} + \lambda\mathbf{y}) - \mathbf{a}\| &= \|(1 - \lambda)(\mathbf{x} - \mathbf{a}) + \lambda(\mathbf{y} - \mathbf{a})\| \\ &\leq \|(1 - \lambda)(\mathbf{x} - \mathbf{a})\| + \|\lambda(\mathbf{y} - \mathbf{a})\| \\ &= (1 - \lambda)\|\mathbf{x} - \mathbf{a}\| + \lambda\|\mathbf{y} - \mathbf{a}\| \\ &\leq (1 - \lambda)\delta + \lambda\delta = \delta. \end{aligned}$$

Therefore  $B(\mathbf{a}; \delta)$  is convex.

Every linear subspace is also a convex set, as well as the translate of every subspace (which is called an affine set). Some other examples of convex sets in  $\mathbb{R}^2$  are shown in Figure 12.1.

We will come back to why each of these sets are convex later. Another important property is that the intersection of a family of convex sets is a convex set.

By a *linear system* we mean a finite system of linear equations and/or linear inequalities involving  $n$  variables. For example

$$x_1 + x_2 = 3, x_1 \geq 0, x_2 \geq 0$$

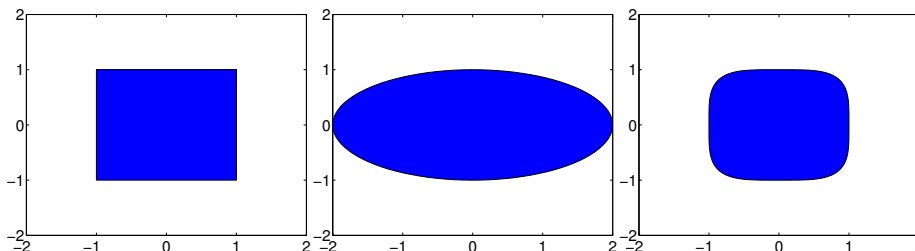


Figure 12.1: Examples of some convex sets. A square, the ellipse  $\frac{x^2}{4} + y^2 \leq 1$ , and the area  $x^4 + y^4 \leq 1$ .

is a linear system in the variables  $x_1, x_2$ . The solution set is the set of points  $(x_1, 3 - x_1)$  where  $0 \leq x_1 \leq 3$ . The set of solutions of a linear system is called a *polyhedron*. These sets often occur in optimization. Thus, a polyhedron has the form

$$P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$$

where  $A \in \mathbb{R}^{m,n}$  and  $\mathbf{b} \in \mathbb{R}^m$  ( $m$  is arbitrary, but finite) and  $\leq$  means componentwise inequality. There are simple techniques for rewriting any linear system in the form  $A\mathbf{x} \leq \mathbf{b}$ .

**Proposition 12.1.** *Polyhedra are convex.*

Every polyhedron is a convex set.

*Proof.* Assume that  $P$  is the polyhedron given by all points where  $A\mathbf{x} \leq \mathbf{b}$ . Assume that  $\mathbf{x}$  and  $\mathbf{y}$  lie in  $P$ , so that  $A\mathbf{x} \leq \mathbf{b}$ , and  $A\mathbf{y} \leq \mathbf{b}$ . We then have that

$$A(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}) = \lambda A\mathbf{x} + (1 - \lambda)A\mathbf{y} \leq \lambda\mathbf{b} + (1 - \lambda)\mathbf{b} = \mathbf{b}.$$

This shows that  $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}$  also lies in  $P$ , so that  $P$  is convex.  $\square$

The square from Figure 12.1(a) is defined by the inequalities  $-1 \leq x, y \leq 1$ . It is therefore a polyhedron, and therefore convex. The next result shows that convex sets are preserved under linear maps.

**Proposition 12.2.** *Linear transformations of convex sets are convex.*

If  $T : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is a linear transformation, and  $C \subseteq \mathbb{R}^n$  is a convex set, then the image  $T(C)$  of this set is also convex.

## 12.2 Convex functions

The notion of a convex function also makes sense for real-valued functions of several variables. Consider a real-valued function  $f : C \rightarrow \mathbb{R}$  where  $C \subseteq \mathbb{R}^n$  is a convex set. We say that  $f$  is *convex* provided that

$$f((1 - \lambda)\mathbf{x} + \lambda\mathbf{y}) \leq (1 - \lambda)f(\mathbf{x}) + \lambda f(\mathbf{y}) \quad (\mathbf{x}, \mathbf{y} \in C, 0 \leq \lambda \leq 1) \quad (12.1)$$

(This inequality holds for all  $\mathbf{x}, \mathbf{y}$  and  $\lambda$  as specified). Due to the convexity of  $C$ , the point  $(1 - \lambda)\mathbf{x} + \lambda\mathbf{y}$  lies in  $C$ , so the inequality is well-defined. The geometrical interpretation in one dimension is that, for any  $x, y$ , the graph of  $f$  on  $[x, y]$  lies below the secant through  $(x, f(x))$  and  $(y, f(y))$ . For  $z \in (x, y)$ , since  $f(z)$  lies below that secant, the secant through  $(x, f(x))$  and  $(z, f(z))$  has a smaller slope than the secant through  $(x, f(x))$  and  $(y, f(y))$ . Since the slope of the secant through  $(x, f(x))$  and  $(y, f(y))$  is  $(f(y) - f(x))/(y - x)$ , it follows that the *slope function*

$$g_x(y) = \frac{f(y) - f(x)}{y - x}$$

is increasing for any  $x$ . This characterizes all convex functions in one dimension in terms of slope functions.

A function  $g$  is called *concave* if  $-g$  is convex.

For every linear function we have that  $f((1 - \lambda)\mathbf{x} + \lambda\mathbf{y}) = (1 - \lambda)f(\mathbf{x}) + \lambda f(\mathbf{y})$ , so that every linear function is convex. Some other examples of convex functions in  $n$  variables are

- $f(\mathbf{x}) = L(\mathbf{x}) + \alpha$  where  $L$  is a linear function from  $\mathbb{R}^n$  into  $\mathbb{R}$  (a linear functional) and  $\alpha$  is a real number. In fact, for such functions we have that  $f((1 - \lambda)\mathbf{x} + \lambda\mathbf{y}) = (1 - \lambda)f(\mathbf{x}) + \lambda f(\mathbf{y})$ , just as for linear functions. Functions on the form  $f(\mathbf{x}) = L(\mathbf{x}) + \alpha$  are called *affine functions*, and may be written on the form  $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + \alpha$  for a suitable vector  $\mathbf{c}$ .
- $f(\mathbf{x}) = \|\mathbf{x}\|$  (Euclidean norm). That this is convex can be proved by writing  $\|(1 - \lambda)\mathbf{x} + \lambda\mathbf{y}\| \leq \|(1 - \lambda)\mathbf{x}\| + \|\lambda\mathbf{y}\| = (1 - \lambda)\|\mathbf{x}\| + \lambda\|\mathbf{y}\|$ . In fact, the same argument can be used to show that *every* norm defines a convex function. Such an example is the  $l_1$ -norm, also called the *sum norm*, defined by  $\|\mathbf{x}\|_1 = \sum_{j=1}^n |x_j|$ .
- $f(\mathbf{x}) = e^{h(\mathbf{x})}$  where  $h : \mathbb{R}^n \rightarrow \mathbb{R}$  is a convex function (exercise 12.4 gives a more general result).
- $f(\mathbf{x}) = \max_i g_i(\mathbf{x})$  where  $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$  is an affine function ( $i \leq m$ ). This means that the pointwise maximum of affine functions is a convex function. Note that such convex functions are typically not differentiable everywhere. A more general result is that the pointwise supremum of an arbitrary family of affine functions (or even convex functions) is convex. This is a very useful fact in convexity and its applications.

The following result is an exercise to prove, and it gives a method for proving convexity of a function.



**Proposition 12.3.** *Composition of convex and affine maps are convex.*

Assume that  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is convex and  $\mathbf{H} : \mathbb{R}^m \rightarrow \mathbb{R}^n$  is affine. Then the composition  $f(\mathbf{H}(\mathbf{x}))$  is convex.

As a consequence,  $f(\mathbf{x}) = e^{\sum_{j=1}^n x_j}$  is convex. The next result is often used, and is called *Jensen's inequality*. It can be proved using induction.

**Theorem 12.4.** *Jensen's inequality.*

Let  $f : C \rightarrow \mathbb{R}$  be a convex function defined on a convex set  $C \subseteq \mathbb{R}^n$ . If  $\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^r \in C$  and  $\lambda_1, \dots, \lambda_r \geq 0$  satisfy  $\sum_{j=1}^r \lambda_j = 1$ , then

$$f\left(\sum_{j=1}^r \lambda_j \mathbf{x}^j\right) \leq \sum_{j=1}^r \lambda_j f(\mathbf{x}^j). \quad (12.2)$$

A point of the form  $\sum_{j=1}^r \lambda_j \mathbf{x}^j$ , where the  $\lambda_j$ 's are nonnegative and sum to 1, is called a *convex combination* of the points  $\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^r$ . One can show that a set is convex if and only if it contains all convex combinations of its points.

Finally, one connection between convex sets and convex functions is the following fact whose proof is an exercise.

**Proposition 12.5.** *sub-level sets of convex functions are convex.*

Let  $C \subseteq \mathbb{R}^n$  be a convex set and consider a convex function  $f : C \rightarrow \mathbb{R}$ . Let  $\alpha \in \mathbb{R}$ . Then the “sub-level” set

$$\{\mathbf{x} \in C : f(\mathbf{x}) \leq \alpha\}$$

is a convex set.

## 12.3 Properties of convex functions

A convex function may not be differentiable in every point. However, one can show that a convex function always has one-sided directional derivatives at any point. But what about continuity? To address this, we will first establish the following lemma.

**Lemma 12.6.** *Convex functions on unit balls.*

Define the norm  $\|\cdot\|_1$  on  $\mathbb{R}^n$  by  $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$ , and let  $B_1(\mathbf{x}, r)$  consist of all points  $\mathbf{z}$  where  $\|\mathbf{z} - \mathbf{x}\|_1 \leq r$ . Any convex function  $f$  defined on  $B_1(\mathbf{x}, r)$  has a maximum in  $B_1(\mathbf{x}, r)$ , and this maximum is achieved in at least one of the points  $\mathbf{x} \pm r\mathbf{e}_i$ .

*Proof.* Since  $\mathbf{z} \rightarrow \mathbf{z} - \mathbf{x}$  is affine (and affine mappings preserve convexity), we can assume without loss of generality that  $\mathbf{x} = \mathbf{0}$ . If  $\mathbf{z} = (z_1, \dots, z_n) \in B_1(\mathbf{0}, r)$  we have that  $\sum_{i=1}^n |z_i| \leq r$ , and we can write

$$\mathbf{z} = \frac{(r - \sum_{i=1}^n |z_i|)}{r} \mathbf{0} + \frac{|z_1|}{r} \text{sign}(z_1) r \mathbf{e}_1 + \dots + \frac{|z_n|}{r} \text{sign}(z_n) r \mathbf{e}_n.$$

This shows that any point in  $B_1(\mathbf{0}, r)$  can be written as a convex combination of the points  $0, \{\pm r\mathbf{e}_i\}_{i=1}^n$ . Labeling these as  $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{2n+1}$  and using the convexity of  $f$  we obtain

$$f(\mathbf{z}) = f\left(\sum_{i=1}^{2n+1} \lambda_i \mathbf{y}_i\right) \leq \sum_{i=1}^{2n+1} \lambda_i f(\mathbf{y}_i) \leq \max_i f(\mathbf{y}_i),$$

which proves that  $f$  has a maximum on  $B_1(\mathbf{0}, r)$ , and this maximum is achieved in one of the  $\mathbf{y}_i$ . Since

$$f(\mathbf{0}) = f\left(\frac{1}{2}r\mathbf{e}_i + \frac{1}{2}(-r\mathbf{e}_i)\right) \leq \frac{1}{2}f(r\mathbf{e}_i) + \frac{1}{2}f(-r\mathbf{e}_i),$$

this maximum must be achieved in a point of the form  $\pm r\mathbf{e}_i$ . The result follows.  $\square$

**Theorem 12.7.** *Convex functions are continuous on open sets.*

Let  $f : C \rightarrow \mathbb{R}$  be a convex function defined on an open set  $C \subseteq \mathbb{R}^n$ . Then  $f$  is continuous on  $C$ .

*Proof.* Let  $\mathbf{x}$  be in  $C$ , and let us show that  $f$  is continuous at  $\mathbf{x}$ . Since  $C$  is open we can find an  $r$  so that  $B(\mathbf{x}, r) \subset C$ . We claim first that we can assume that  $f$  is bounded from above on  $B(\mathbf{x}, r)$ . To prove this, note first that  $\|\mathbf{x}\|_1 > \|\mathbf{x}\|$ , so that  $B_1(\mathbf{x}, r) \subset B(\mathbf{x}, r)$ . On the other hand  $B(\mathbf{x}, r/n) \subset B_1(\mathbf{x}, r)$ . Using Lemma 12.6 we see that  $f$  is also bounded from above on a set of the form  $B(\mathbf{x}, s)$  (choose  $s = r/n$  for instance).

Assume now that  $f(\mathbf{y}) \leq M$  on  $B(\mathbf{x}, r)$ , and let  $z \in B(\mathbf{x}, r)$ . Define the function  $g(t) = f\left(\mathbf{x} + t\frac{z-\mathbf{x}}{\|z-\mathbf{x}\|}\right)$  for  $t \in (-r, r)$ . Note that  $g(\|z-\mathbf{x}\|) = f(\mathbf{z})$ .  $H(t) = \mathbf{x} + (z-\mathbf{x})/ \|z-\mathbf{x}\| t$  takes its values in  $B(\mathbf{x}, r)$ , and since  $f$  is convex and  $H$  is affine,  $g(t) = f(H(t))$  is convex, and then  $g$  has an increasing slope function  $s \rightarrow (g(s) - g(t))/(s - t)$ . In particular, with  $s = -r, \|z-\mathbf{x}\|, r$  and  $t = 0$  we obtain

$$\frac{g(-r) - g(0)}{-r} \leq \frac{g(\|z-\mathbf{x}\|) - g(0)}{\|z-\mathbf{x}\|} \leq \frac{g(r) - g(0)}{r}.$$

The expression in the middle can be written as  $(f(\mathbf{z}) - f(\mathbf{x}))/\|z-\mathbf{x}\|$ . Since  $g(t)$  is bounded above by  $M$ ,  $-M \leq -g(-r)$  and  $g(r) \leq M$ , so that

$$\frac{-M + f(\mathbf{x})}{r} \leq \frac{f(\mathbf{z}) - f(\mathbf{x})}{\|z-\mathbf{x}\|} \leq \frac{M - f(\mathbf{x})}{r}.$$

From this it follows that

$$|f(\mathbf{z}) - f(\mathbf{x})| \leq \frac{|M| + |f(\mathbf{x})|}{r} \|z-\mathbf{x}\|,$$

and the continuity of  $f$  follows.  $\square$

However, a convex function may be discontinuous in points on the *boundary* of its domain. For instance, the function  $f : [0, 1] \rightarrow \mathbb{R}$  given by  $f(0) = 1$  and  $f(x) = 0$  for  $x \in (0, 1]$  is convex, but discontinuous at  $x = 0$ . Next we give a useful technique for checking that a function is convex.

**Theorem 12.8.** *Convex functions and positive semidefinite Hessians.*

Let  $f$  be a real-valued function defined on an open convex set  $C \subseteq \mathbb{R}^n$  and assume that  $f$  has continuous second-order partial derivatives on  $C$ .

Then  $f$  is convex if and only if the Hessian matrix  $\nabla^2 f(\mathbf{x})$  is positive semidefinite for each  $\mathbf{x} \in C$ .

Using Theorem 12.8 it is straightforward to prove that the remaining sets from Figure 12.1 are convex. They can be written as sub-level sets of the functions  $f(x, y) = \frac{x^2}{4} + y^2$ , and  $f(x, y) = x^4 + y^4$ . For the first of these the level sets are ellipses, and are shown in Figure 12.2, together with  $f$  itself. One can quickly verify that the Hessian matrices of these functions are positive semidefinite. It follows from Proposition 12.5 that the corresponding sets are convex.

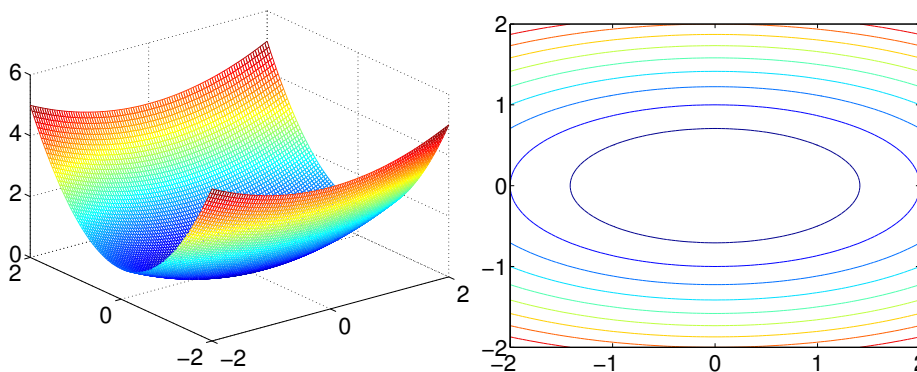


Figure 12.2: The function  $f(x, y) = \frac{x^2}{4} + y^2$  and some of its level curves.

An important class of convex functions consists of (certain) quadratic functions. Let  $A \in \mathbb{R}^{n \times n}$  be a symmetric matrix which is positive semidefinite and consider the quadratic function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  given by

$$f(\mathbf{x}) = (1/2) \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x} = (1/2) \sum_{i,j} a_{ij} x_i x_j - \sum_{j=1}^n b_j x_j.$$

(If  $A = \mathbf{0}$ , then the function is linear, and it may be strange to call it quadratic. But we still do this, for simplicity.) Then (Exercise 11.9) the Hessian matrix of  $f$  is  $A$ , i.e.,  $\nabla^2 f(\mathbf{x}) = A$  for each  $\mathbf{x} \in \mathbb{R}^n$ . Therefore, by Theorem 12.8 is a convex function.

We remark that sometimes it may be easy to check that a symmetric matrix  $A$  is positive semidefinite. A (real) symmetric  $n \times n$  matrix  $A$  is called *diagonally dominant* if  $|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$  for  $i = 1, \dots, n$ . These matrices arise in many

applications, e.g. splines and differential equations. It can be shown that every symmetric diagonally dominant matrix is positive semidefinite. For a simple proof of this fact using convexity, see [10]. Thus, we get a simple criterion for convexity of a function: check if the Hessian matrix  $\nabla^2 f(\mathbf{x})$  is diagonally dominant for each  $\mathbf{x}$ . Be careful here: this matrix may be positive semidefinite without being diagonally dominant!

We now look at differentiability properties of convex functions.

**Theorem 12.9.** *Convexity, partial derivatives and differentiability.*

Let  $f$  be a real-valued convex function defined on an open convex set  $C \subseteq \mathbb{R}^n$ . Assume that all the partial derivatives  $\partial f(\mathbf{x})/\partial x_1, \dots, \partial f(\mathbf{x})/\partial x_n$  exist at a point  $\mathbf{x} \in C$ . Then  $f$  is differentiable at  $\mathbf{x}$ .

A convex function may not be differentiable everywhere, but it is differentiable “almost everywhere”. More precisely, for a convex function defined on an open convex set in  $\mathbb{R}^n$ , the set of points for which  $f$  is not differentiable has Lebesgue measure zero. We do not go into further details on this here, but refer to e.g. [19] for a proof and a discussion.

Another characterization of convex functions that involves the gradient may now be presented.

**Theorem 12.10.** *Equivalent conditions for convex functions.*

Let  $f : C \rightarrow \mathbb{R}$  be a differentiable function defined on an open convex set  $C \subseteq \mathbb{R}^n$ . Then the following conditions are equivalent:

1.  $f$  is convex.
2.  $f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0)$  for all  $\mathbf{x}, \mathbf{x}_0 \in C$ .
3.  $(\nabla f(\mathbf{x}) - \nabla f(\mathbf{x}_0))^T(\mathbf{x} - \mathbf{x}_0) \geq 0$  for all  $\mathbf{x}, \mathbf{x}_0 \in C$ .

This theorem is important. Property 2 says that the first-order Taylor approximation of  $f$  at  $x_0$  (which is the right-hand side of the inequality) always underestimates  $f$ . This result has interesting consequences for optimization as we shall see later.

*Proof.* Assume first that  $n = 1$ . If  $f$  is convex we have that

$$f(x_0 + t(x - x_0)) = f((1 - t)x_0 + tx) \leq (1 - t)f(x_0) + tf(x),$$

which can be rewritten as

$$f(x) \geq f(x_0) + \frac{f(x_0 + t(x - x_0)) - f(x_0)}{t} = f(x_0) + \frac{f(x_0 + t(x - x_0)) - f(x_0)}{t(x - x_0)}(x - x_0).$$

Taking the limit as  $t \rightarrow 0$  shows that (ii) holds. (iii) follows from (ii) by adding the two equations

$$\begin{aligned} f(\mathbf{x}) &\geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) \\ f(\mathbf{x}_0) &\geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{x}_0 - \mathbf{x}) \end{aligned}$$

and reorganizing the terms (actually this holds for any  $n$ ). (iii) says that the derivative is increasing. Given  $x_1 < x_2 < x_3$ , the mean value theorem says that there exist  $x_1 \leq c \leq x_2$ ,  $x_2 \leq d \leq x_3$ , so that

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1} = f'(c), \text{ and } \frac{f(x_3) - f(x_2)}{x_3 - x_2} = f'(d).$$

Since  $f'(c) \leq f'(d)$ , the slope of the secant from  $x_1$  to  $x_2$  is smaller than the slope of the secant from  $x_2$  to  $x_3$ . But then clearly the slope function is increasing, so that  $f$  is convex. This completes the proof for  $n = 1$ .

When  $n > 1$ , define  $g(t) = f(t\mathbf{x} + (1-t)\mathbf{x}_0)$ . If  $f$  is convex, then  $g$  is also convex, and from (ii) it follows that  $g(1) \geq g(0) + g'(0)$ . The chain rule then gives that  $g'(0) = \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0)$ , and (ii) follows since  $g(0) = f(\mathbf{x}_0)$ ,  $g(1) = f(\mathbf{x})$ .

If we also show that (iii) implies (i) the proof will be complete. Let  $0 \leq t_1 \leq t_2 < 1$ , and define  $\mathbf{y}_i = t_i\mathbf{x} + (1-t_i)\mathbf{x}_0$  for  $i = 1, 2$ . Note first that (iii) is the same as

$$\nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) \leq \nabla f(\mathbf{x})^T(\mathbf{x} - \mathbf{x}_0).$$

We have that  $\mathbf{y}_2 - \mathbf{y}_1 = (t_2 - t_1)(\mathbf{x} - \mathbf{x}_0)$ , and we get

$$g'(t_i) = \nabla f(\mathbf{y}_i)^T(\mathbf{x} - \mathbf{x}_0) = \nabla f(\mathbf{y}_i)^T(\mathbf{y}_2 - \mathbf{y}_1)/(t_2 - t_1).$$

Since (iii) also holds if we replace  $\mathbf{x}_0, \mathbf{x}$  with  $\mathbf{y}_1, \mathbf{y}_2$ , it follows that  $g'(t_1) \leq g'(t_2)$ , and it follows that  $g$  is convex. From this it also follows that  $f$  is convex, since

$$g(t) = f((1-t)\mathbf{x}_0 + t\mathbf{x}) = g((1-t) \cdot 0 + t \cdot 1) \leq (1-t)g(0) + tg(1) = (1-t)f(\mathbf{x}_0) + tf(\mathbf{x}).$$

□

### Exercise 12.1: The intersection of convex sets is convex.

We recall that  $A \cap B$  consists of all points which lie both in  $A$  and  $B$ . Show that  $A \cap B$  is convex when  $A$  and  $B$  are.

### Exercise 12.2: Solve

Suppose that  $f$  is a convex function defined on  $\mathbb{R}$  which also is positive. Show that  $g(x) = (f(x))^n$  also is convex.

**Exercise 12.3: The convexity of a product of functions.**

a) Assume that  $f, g$  are convex, positive, and increasing functions, both two times differentiable and defined on  $\mathbb{R}$ . Compute the second derivative of  $h(x) = f(x)g(x)$ , consider its sign, and prove from this that  $f(x)g(x)$  is convex.

This can also be generalized to functions which are not differentiable. For this we first need the following result.

b) Show that, for any functions  $f, g$  defined on  $\mathbb{R}$  which are convex, positive, and increasing, we have that

$$\begin{aligned} & \lambda f(x)g(x) + (1 - \lambda)f(y)g(y) - f(\lambda x + (1 - \lambda)y)g(\lambda x + (1 - \lambda)y) \\ & \geq \lambda(1 - \lambda)(f(x)g(x) + f(y)g(y) - f(x)g(y) - f(y)g(x)) \end{aligned}$$

c) Explain why it follows from b) that  $f(x)g(x)$  is convex, under the same conditions on  $f$  and  $g$ .

**Hint.** Start by writing  $f(x)g(x) + f(y)g(y) - f(x)g(y) - f(y)g(x)$  as a product.

**Exercise 12.4: The convexity of the composition of functions.**

a) Let  $f$  and  $g$  both be two times (continuously) differentiable functions both-defined on  $\mathbb{R}$ . Suppose also that  $f$  and  $g$  are convex, and that  $f$  is increasing. Compute the second derivative of  $h(x) = f(g(x))$ , consider its sign, and deduce from this that  $f(g(x))$  is convex. This states that, in particular the function  $f(\mathbf{x}) = e^{h(\mathbf{x})}$  (which we previously just stated as convex without proof), is convex when  $h$  is.

b) Construct two convex functions  $f, g$  so that  $h(x) = f(g(x))$  is not convex.

The result from a) holds also when  $f$  and  $g$  are not differentiable. In fact,  $g$  can be defined on  $\mathbb{R}^n$ :

c) Let  $f$  and  $g$  be convex functions, and suppose that  $f$  is increasing and defined on  $\mathbb{R}$ ,  $g$  defined on  $\mathbb{R}^n$ . Show that  $h(\mathbf{x}) = f(g(\mathbf{x}))$  also is convex.

**Exercise 12.5: Solve**

Let  $S = \{(x, y, z) : z \geq x^2 + y^2\} \subset \mathbb{R}^3$ . Sketch the set and verify that it is a convex set.

**Exercise 12.6: Solve**

Let  $f : S \rightarrow \mathbb{R}$  be a differentiable function, where  $S$  is an open set in  $\mathbb{R}$ . Check that  $f$  is convex if and only if  $f''(x) \geq 0$  for all  $x \in S$ .

**Exercise 12.7: Solve**

Prove Proposition 12.3.

**Exercise 12.8: Solve**

Prove Proposition 12.5.

**Exercise 12.9: Solve**

Explain how you can write the LP problem  $\max \{c^T x : Ax \geq b, Bx = d, x \geq 0\}$  as an LP problem of the form

$$\max\{c^T x : Hx \leq h, x \geq 0\}$$

for suitable matrix  $H$  and vector  $h$ .

**Exercise 12.10: The set of convex combinations is convex**

Let  $x_1, \dots, x_t \in \mathbb{R}^n$  and let  $C$  be the set of vectors of the form

$$\sum_{j=1}^t \lambda_j x_j$$

where  $\lambda_j \geq 0$  for each  $j = 1, \dots, t$ , and  $\sum_{j=1}^t \lambda_j = 1$ . Show that  $C$  is convex. Make a sketch of such a set in  $\mathbb{R}^3$ .

**Exercise 12.11: Solve**

Assume that  $f$  and  $g$  are convex functions defined on an interval  $I$ . Which of the following functions are convex or concave?

- a)  $\lambda f$  where  $\lambda \in \mathbb{R}$ ,
- b)  $\min\{f, g\}$ ,
- c)  $|f|$ .

**Exercise 12.12: A convex function defined on a closed real interval attains its maximum in one of the end points.**

Let  $f : [a, b] \rightarrow \mathbb{R}$  be a convex function. Show that

$$\max\{f(x) : x \in [a, b]\} = \max\{f(a), f(b)\}.$$

**Exercise 12.13: The maximum of convex functions is convex.**

Show that  $\max\{f, g\}$  is a convex function when  $f$  and  $g$  are convex (we define  $\max\{f, g\}$  by  $\max\{f, g\}(\mathbf{x}) = \max(f(\mathbf{x}), g(\mathbf{x}))$ ).

**Exercise 12.14: Solve**

Let  $f : (0, \infty) \rightarrow \mathbb{R}$  and define the function  $g : (0, \infty) \rightarrow \mathbb{R}$  by  $g(x) = xf(1/x)$ . Why is the function  $x \rightarrow xe^{1/x}$  convex?

**Exercise 12.15: The distance to a convex set is a convex function.**

Let  $C \subseteq \mathbb{R}^n$  be a convex set and consider the distance function  $d_C$  defined by  $d_C(x) = \inf\{\|x - y\| : y \in C\}$ . Show that  $d_C$  is a convex function.



# Chapter 13

## Nonlinear equations

A basic mathematical problem is to solve a system of equations in several unknowns (variables). There are numerical methods that can solve such equations, at least within a small error tolerance. We shall briefly discuss such methods here; for further details, see [24, 32].

### 13.1 Equations and fixed points

In linear algebra one works a lot with linear equations in several variables, and Gaussian elimination is a central method for solving such equations. There are also other faster methods, so-called iterative methods, for linear equations. But what about *nonlinear equations*? For instance, consider the system in two variables  $x_1$  and  $x_2$ :

$$\begin{aligned}x_1^2 - x_1x_2^{-3} + \cos x_1 &= 1 \\5x_1^4 + 2x_1^3 - \tan(x_1x_2^8) &= 3\end{aligned}$$

Clearly, such equations can be very hard to solve. The general problem is to solve the equation

$$\mathbf{F}(\mathbf{x}) = \mathbf{0} \tag{13.1}$$

for a given function  $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ . If  $\mathbf{F}(\mathbf{x}) = \mathbf{0}$  we call  $\mathbf{x}$  a *root* of  $\mathbf{F}$  (or of the equation). The example above is equivalent to finding roots in  $\mathbf{F}(\mathbf{x}) = (F_1(\mathbf{x}), F_2(\mathbf{x}))$  where

$$\begin{aligned}F_1(\mathbf{x}) &= x_1^2 - x_1x_2^{-3} + \cos x_1 - 1 \\F_2(\mathbf{x}) &= 5x_1^4 + 2x_1^3 - \tan(x_1x_2^8) - 3\end{aligned}$$

In particular, if  $\mathbf{F}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$  where  $\mathbf{A}$  is an  $n \times n$  matrix and  $\mathbf{b} \in \mathbb{R}^n$ , then we are back to linear equations (a square system). More generally one may consider equations  $\mathbf{G}(\mathbf{x}) = \mathbf{0}$  where  $\mathbf{G} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , but we here only discuss the case  $m = n$ .

Often the problem  $\mathbf{F}(\mathbf{x}) = \mathbf{0}$  has the following form, or may be rewritten to it:

$$\mathbf{K}(\mathbf{x}) = \mathbf{x}. \quad (13.2)$$

for some function  $\mathbf{K} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ . This corresponds to the special choice  $\mathbf{F}(\mathbf{x}) = \mathbf{K}(\mathbf{x}) - \mathbf{x}$ . A point  $\mathbf{x} \in \mathbb{R}^n$  such that  $\mathbf{x} = \mathbf{K}(\mathbf{x})$  is called a *fixed point* of the function  $\mathbf{K}$ . In finding such a fixed point it is tempting to use the following iterative method: choose a starting point  $\mathbf{x}_0$  and repeat the following iteration

$$\mathbf{x}_{k+1} = \mathbf{K}(\mathbf{x}_k) \quad \text{for } k = 1, 2, \dots \quad (13.3)$$

This is called a *fixed-point iteration*. We note that if  $\mathbf{K}$  is continuous and this procedure converges to some point  $\mathbf{x}^*$ , then  $\mathbf{x}^*$  must be a fixed point. The fixed-point iteration is an extremely simple algorithm, and very easy to implement. Perhaps surprisingly, it also works very well for many such problems.

When does the fixed-point iteration work? Let  $\|\cdot\|$  be a fixed norm, e.g. the Euclidean norm, on  $\mathbb{R}^n$ . We say that the function  $\mathbf{K} : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is a *contraction* if there is a constant  $0 \leq c < 1$  such that

$$\|\mathbf{K}(\mathbf{x}) - \mathbf{K}(\mathbf{y})\| \leq c\|\mathbf{x} - \mathbf{y}\| \quad (\mathbf{x}, \mathbf{y} \in \mathbb{R}^n).$$

We also say that  $\mathbf{K}$  is *c-Lipschitz* in this case. The following theorem is called the *Banach contraction principle*. It also holds in Banach spaces, i.e., complete normed vector spaces (possibly infinite-dimensional).

**Theorem 13.1.** *Banach contraction principle.*

Assume that  $\mathbf{K}$  is *c-Lipschitz* with  $0 < c < 1$ . Then  $\mathbf{K}$  has a unique fixed point  $\mathbf{x}^*$ . For any starting point  $\mathbf{x}_0$  the fixed-point iteration (13.3) generates a sequence  $\{\mathbf{x}_k\}_{k=0}^{\infty}$  that converges to  $\mathbf{x}^*$ . Moreover

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq c\|\mathbf{x}_k - \mathbf{x}^*\| \quad \text{for } k = 0, 1, \dots \quad (13.4)$$

so that

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leq c^k \|\mathbf{x}_0 - \mathbf{x}^*\|.$$

*Proof.* First, note that if both  $\mathbf{x}$  and  $\mathbf{y}$  are fixed points of  $\mathbf{K}$ , then

$$\|\mathbf{x} - \mathbf{y}\| = \|\mathbf{K}(\mathbf{x}) - \mathbf{K}(\mathbf{y})\| \leq c\|\mathbf{x} - \mathbf{y}\|$$

which means that  $\mathbf{x} = \mathbf{y}$  (as  $c < 1$ ); therefore  $\mathbf{K}$  has at most one fixed point. Next, we compute

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\| = \|\mathbf{K}(\mathbf{x}_k) - \mathbf{K}(\mathbf{x}_{k-1})\| \leq c\|\mathbf{x}_k - \mathbf{x}_{k-1}\| = \dots \leq c^k \|\mathbf{x}_1 - \mathbf{x}_0\|$$

so

$$\begin{aligned}\|\mathbf{x}_m - \mathbf{x}_0\| &= \left\| \sum_{k=0}^{m-1} (\mathbf{x}_{k+1} - \mathbf{x}_k) \right\| \leq \sum_{k=0}^{m-1} \|\mathbf{x}_{k+1} - \mathbf{x}_k\| \\ &\leq \left( \sum_{k=0}^{m-1} c^k \right) \|\mathbf{x}_1 - \mathbf{x}_0\| \leq (1/(1-c)) \|\mathbf{x}_1 - \mathbf{x}_0\|\end{aligned}$$

From this we derive that  $\{\mathbf{x}_k\}$  is a Cauchy sequence; as we have

$$\begin{aligned}\|\mathbf{x}_{s+m} - \mathbf{x}_s\| &= \|\mathbf{K}(\mathbf{x}_{s+m-1}) - \mathbf{K}(\mathbf{x}_{s-1})\| \leq c \|\mathbf{x}_{s+m-1} - \mathbf{x}_{s-1}\| = \cdots \\ &\leq c^s \|\mathbf{x}_m - \mathbf{x}_0\| \leq (c^s/(1-c)) \|\mathbf{x}_1 - \mathbf{x}_0\|.\end{aligned}$$

and  $0 < c < 1$ . Any Cauchy sequence in  $\mathbb{R}^n$  has a limit point, so  $\mathbf{x}_m \rightarrow \mathbf{x}^*$  for some  $\mathbf{x}^* \in \mathbb{R}^n$ . We now prove that the limit point  $\mathbf{x}^*$  is a (actually, the) fixed point:

$$\begin{aligned}\|\mathbf{x}^* - \mathbf{K}(\mathbf{x}^*)\| &\leq \|\mathbf{x}^* - \mathbf{x}_m\| + \|\mathbf{x}_m - \mathbf{K}(\mathbf{x}^*)\| \\ &= \|\mathbf{x}^* - \mathbf{x}_m\| + \|\mathbf{K}(\mathbf{x}_{m-1}) - \mathbf{K}(\mathbf{x}^*)\| \\ &\leq \|\mathbf{x}^* - \mathbf{x}_m\| + c \|\mathbf{x}_{m-1} - \mathbf{x}^*\|\end{aligned}$$

and letting  $m \rightarrow \infty$  here gives  $\|\mathbf{x}^* - \mathbf{K}(\mathbf{x}^*)\| \leq 0$  so  $\mathbf{x}^* = \mathbf{K}(\mathbf{x}^*)$  as desired.

Finally,

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| = \|\mathbf{K}(\mathbf{x}_k) - \mathbf{K}(\mathbf{x}^*)\| \leq c \|\mathbf{x}_k - \mathbf{x}^*\| \leq c^{k+1} \|\mathbf{x}_0 - \mathbf{x}^*\|.$$

which completes the proof.  $\square$

We see that  $\mathbf{x}_k \rightarrow \mathbf{x}^*$  *linearly*, and that Equation (13.4) gives an estimate on the convergence speed.

## 13.2 Newton's method

We return to the main problem (13.1). Our goal is to present Newton's method, a highly efficient iterative method for solving this equation. The method constructs a sequence

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$$

in  $\mathbb{R}^n$  which, hopefully, converges to a root  $\mathbf{x}^*$  of  $\mathbf{F}$ , so  $\mathbf{F}(\mathbf{x}^*) = \mathbf{0}$ . The idea is to linearize  $\mathbf{F}$  at the current iterate  $\mathbf{x}_k$  and choose the next iterate  $\mathbf{x}_{k+1}$  as a zero of this linearized function. The first order Taylor approximation of  $\mathbf{F}$  at  $\mathbf{x}_k$  is

$$T_{\mathbf{F}}^1(\mathbf{x}_k; \mathbf{x}) = \mathbf{F}(\mathbf{x}_k) + \mathbf{F}'(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k).$$

We solve  $T_{\mathbf{F}}^1(\mathbf{x}_k; \mathbf{x}) = \mathbf{0}$  for  $\mathbf{x}$  and define the next iterate as  $\mathbf{x}_{k+1} = \mathbf{x}$ . This gives

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{F}'(\mathbf{x}_k)^{-1} \mathbf{F}(\mathbf{x}_k) \tag{13.5}$$

which leads to Newton's method. One here assumes that the derivative  $\mathbf{F}'$  is known analytically. Note that we do not (and hardly ever do!) compute the inverse of the matrix  $\mathbf{F}'$ . In the main step, which is to compute  $\mathbf{p}$ , one needs to solve an  $n \times n$  linear system of equations where the coefficient matrix is the Jacobi matrix of  $\mathbf{F}$ , evaluated at  $\mathbf{x}_k$ . In MAT1110 [26] we implemented the following code for Newton's method for nonlinear equations:

```
function x=newtonmult(x0,F,J)
% Performs Newtons method in many variables
% x: column vector which contains the start point
% F: computes the values of F
% J: computes the Jacobi matrix
epsilon=0.0000001; N=30; n=0;
x=x0;
while norm(F(x)) > epsilon && n<=N
    x=x-J(x)\F(x);
    fval = F(x);
    %fprintf('itnr=%2d x=[%13.10f,%13.10f] F(x)=[%13.10f,%13.10f]\n',...
    %      n,x(1),x(2),fval(1),fval(2))
    n = n + 1;
end
```

This code also terminates after a given number of iterations, and when a given accuracy is obtained. Note that this function should work for any function  $\mathbf{F}$ , since it is a parameter to the function.

The convergence of Newton's method may be analyzed using fixed point theory since one may view Newton's method as a fixed point iteration. Observe that the Newton iteration (13.5) may be written

$$\mathbf{x}_{k+1} = \mathbf{G}(\mathbf{x}_k)$$

where  $\mathbf{G}$  is the function

$$\mathbf{G}(\mathbf{x}) = \mathbf{x} - \mathbf{F}'(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x})$$

From this it is possible to show that if the starting point is sufficiently close to the root, then Newton's method will converge to this root at a linear convergence rate. With more clever arguments one may show that the convergence rate of Newton's method is even faster: it has superlinear convergence. Actually, for many functions one even has quadratic convergence rate. The proof of the following convergence theorem relies purely on Taylor's theorem.

**Theorem 13.2.** *Convergence of the Newton method is superlinear.*

Assume that Newton's method with initial point  $\mathbf{x}_0$  produces a sequence  $\{\mathbf{x}_k\}_{k=0}^{\infty}$  which converges to a solution  $\mathbf{x}^*$  of (13.1). Then the convergence rate is superlinear.

*Proof.* From Taylor's theorem for vector-valued functions, Theorem 11.5, in the point  $\mathbf{x}_k$  we have

$$\mathbf{0} = \mathbf{F}(\mathbf{x}^*) = \mathbf{F}(\mathbf{x}_k + (\mathbf{x}^* - \mathbf{x}_k)) = \mathbf{F}(\mathbf{x}_k) + \mathbf{F}'(\mathbf{x}_k)(\mathbf{x}^* - \mathbf{x}_k) + O(\|\mathbf{x}_k - \mathbf{x}^*\|)$$

Multiplying this equation by  $\mathbf{F}'(\mathbf{x}_k)^{-1}$  (which is assumed to exist!) gives

$$\mathbf{x}_k - \mathbf{x}^* - \mathbf{F}'(\mathbf{x}_k)^{-1}\mathbf{F}(\mathbf{x}_k) = O(\|\mathbf{x}_k - \mathbf{x}^*\|)$$

Combining this with the Newton iteration  $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{F}'(\mathbf{x}_k)^{-1}\mathbf{F}(\mathbf{x}_k)$  we get

$$\mathbf{x}_{k+1} - \mathbf{x}^* = O(\|\mathbf{x}_k - \mathbf{x}^*\|).$$

So

$$\lim_{k \rightarrow \infty} \|\mathbf{x}_{k+1} - \mathbf{x}^*\| / \|\mathbf{x}_k - \mathbf{x}^*\| = 0$$

This proves the superlinear convergence.  $\square$

The previous result is interesting, but it does not say *how near* to the root the starting point need to be in order to get convergence. This is the next topic. Let  $\mathbf{F} : U \rightarrow \mathbb{R}^n$  where  $U$  is an open, convex set in  $\mathbb{R}^n$ . Consider the conditions on the derivative  $\mathbf{F}'$

$$\begin{aligned} (i) \quad & \|\mathbf{F}'(\mathbf{x}) - \mathbf{F}'(\mathbf{y})\|_2 \leq L\|\mathbf{x} - \mathbf{y}\| && \text{for all } \mathbf{x}, \mathbf{y} \in U \\ (ii) \quad & \|\mathbf{F}'(\mathbf{x}_0)\|_2 \leq K && \text{for some } \mathbf{x}_0 \in U \end{aligned} \quad (13.6)$$

where  $K$  and  $L$  are some constants. Here  $\|\mathbf{F}'(\mathbf{x}_0)\|_2$  denotes the *spectral norm* of the square matrix  $\mathbf{F}'(\mathbf{x}_0)$ . For a square matrix  $A$  this is defined by

$$\|A\|_2 = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|.$$

It is a fact that  $\|A\|_2$  is equal to the largest singular value of  $A$ , and that it measures how much the operator  $\mathbf{F}'(\mathbf{x}_0)$  may increase the size of vectors. The following convergence result for Newton's method is known as *Kantorovich' theorem*.

**Theorem 13.3.** *Kantorovich' theorem.*

Let  $\mathbf{F} : U \rightarrow \mathbb{R}^n$  be a differentiable function satisfying (13.6). Assume that  $\bar{B}(\mathbf{x}_0; 1/(KL)) \subseteq U$  and that

$$\|\mathbf{F}'(\mathbf{x}_0)^{-1}\mathbf{F}(\mathbf{x}_0)\| \leq 1/(2KL).$$

Then  $\mathbf{F}'(\mathbf{x})$  is invertible for all  $\mathbf{x} \in B(\mathbf{x}_0; 1/(KL))$  and Newton's method with initial point  $\mathbf{x}_0$  will produce a sequence  $\{\mathbf{x}_k\}_{k=0}^{\infty}$  contained in  $B(\mathbf{x}_0; 1/(KL))$  and  $\lim_{k \rightarrow \infty} \mathbf{x}_k = \mathbf{x}^*$  for some limit point  $\mathbf{x}^* \in \bar{B}(\mathbf{x}_0; 1/(KL))$  with

$$\mathbf{F}(\mathbf{x}^*) = \mathbf{0}.$$

A proof of this theorem is quite long (but not very difficult to understand) [26].

One disadvantage with Newton's method is that one needs to know the Jacobi matrix  $\mathbf{F}'$  explicitly. For complicated functions, or functions being the

output of a simulation, the derivative may be hard or impossible to find. The *quasi-Newton method*, also called the *secant-method*, is then a good alternative. The idea is to approximate  $\mathbf{F}'(\mathbf{x}_k)$  by some matrix  $B_k$  and to compute the new search direction from

$$B_k \mathbf{p} = -\mathbf{F}(\mathbf{x}_k)$$

The method we define will make the following assumption:

**Definition 13.4.** *Broyden's method.*

Assume that we have chosen the next iterate  $\mathbf{x}_{k+1}$ . *Broyden's method* updates  $B_k$  to  $B_{k+1}$  in such a way that

$$B_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \mathbf{F}(\mathbf{x}_{k+1}) - \mathbf{F}(\mathbf{x}_k), \quad (13.7)$$

and so that  $B_{k+1}\mathbf{u} = B_k\mathbf{u}$  for all  $\mathbf{u}$  orthogonal to  $\mathbf{x}_{k+1} - \mathbf{x}_k$ .

Equation (13.7) is close to true if we replace  $B_{k+1}$  with  $\mathbf{F}'(\mathbf{x}_k)$  (due to the Taylor series of first order), so this is a reasonable assumption to make. The assumption that  $B_{k+1}$  acts as  $B_k$  on vectors orthogonal to  $\mathbf{s}_k$  comes from that the only new information we need to encapsulate in  $B_{k+1}$  is given by Equation (13.7).

It is straightforward to find an expression for  $B_{k+1}$ . Define  $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$  and  $\mathbf{y}_k = \mathbf{F}(\mathbf{x}_{k+1}) - \mathbf{F}(\mathbf{x}_k)$ . We require that  $B_{k+1}\mathbf{s}_k = \mathbf{y}_k$ . The projection onto the space spanned by  $\mathbf{s}_k$  is given by the matrix  $\mathbf{s}_k \mathbf{s}_k^T / \mathbf{s}_k^T \mathbf{s}_k$ , and the projection onto the orthogonal complement of this space is  $I - \mathbf{s}_k \mathbf{s}_k^T / \mathbf{s}_k^T \mathbf{s}_k$ . Since  $B = \mathbf{y}_k \mathbf{s}_k^T / \mathbf{s}_k^T \mathbf{s}_k$  satisfies  $B\mathbf{s}_k = \mathbf{y}_k$  and  $B\mathbf{u} = \mathbf{0}$  for all vectors in the orthogonal complement, we have that

$$B_{k+1} = B_k \left( I - \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{s}_k} \right) + \frac{\mathbf{y}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{s}_k} = B_k + \frac{(\mathbf{y}_k - B_k \mathbf{s}_k) \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{s}_k}. \quad (13.8)$$

Note that the matrix in Equation (13.8) is a rank one update of  $B_k$ , so that it can be computed efficiently. In an algorithm for Broyden's method  $B_{k+1}$  is computed from Equation (13.8), then  $\mathbf{x}_{k+2}$  is computed by following the search direction  $\mathbf{p}$  obtained by solving  $B_{k+1}\mathbf{p} = -\mathbf{F}(\mathbf{x}_{k+1})$ , and so on. Finally  $\mathbf{s}_{k+1}$  and  $\mathbf{y}_{k+1}$  are updated. An algorithm also computes an  $\alpha$  through what we call a *line search*, to attempt to find the optimal distance to follow the search direction. We do not here specify how this line search can be performed. Also, we do not specify how the initial values can be chosen. For  $B_0$ , any approximation of the Jacobian of  $\mathbf{F}$  at  $\mathbf{x}_0$  can be used, using a numerical differentiation method of your own choosing. One can show that Broyden's method, under certain assumptions, also converges superlinearly, see [32].

### Exercise 13.1: Solve

Show that the problem of solving nonlinear equations (13.1) may be transformed into a nonlinear optimization problem.

**Hint.** Square each component function and sum these up!

### Exercise 13.2: Solve

Let  $T : \mathbb{R} \rightarrow \mathbb{R}$  be given by  $T(x) = (3/2)(x - x^3)$ . Draw the graph of this function, and determine its fixed points. Let  $x^*$  denote the largest fixed point. Find, using your graph, an interval  $I$  containing  $x^*$  such that the fixed point algorithm with an initial point in  $I$  will guaranteed converge towards  $x^*$ . Then try the fixed point algorithm with starting point  $x_0 = \sqrt{5/3}$ .

### Exercise 13.3: Solve

Let  $\alpha \in \mathbb{R}_+$  be fixed, and consider  $f(x) = x^2 - \alpha$ . Then the zeros are  $\pm\sqrt{\alpha}$ . Write down the Newton's iteration for this problem. Let  $\alpha = 2$  and compute the first three iterates in Newton's method when  $x_0 = 1$ .

### Exercise 13.4: Solve

For any vector norm  $\|\cdot\|$  on  $\mathbb{R}^n$ , we can more generally define a corresponding *operator norm* for  $n \times n$  matrices by

$$\|A\| = \sup_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|.$$

a) Explain why this supremum is attained.

In the rest of this exercise we will use the vector norm  $\|\mathbf{x}\| = \|\mathbf{x}\|_1 = \sum_{j=1}^n |x_j|$  on  $\mathbb{R}^n$ .

b) For  $n = 2$ , draw the sub-level set  $\{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\|_1 \leq 1\}$ .

c) Show that  $f(\mathbf{x}) = \|A\mathbf{x}\|_1$  is convex for any  $n$ . It follows from Lemma 12.6 that the maximum of  $f$  on the set  $\{\mathbf{x} : \|\mathbf{x}\|_1 = 1\}$  is attained in a point on the form  $\pm\mathbf{e}_k$ .

d) Show that, for any  $n \times n$ -matrix  $A$ ,  $\|A\| = \sup_k \sum_{i=1}^n |a_{ik}|$ , where  $a_{ij}$  are the entries of  $A$  (i.e. the biggest sum of absolute values in a column).

### Exercise 13.5: Solve

Consider a linear map  $T : \mathbb{R}^n \rightarrow \mathbb{R}^m$  given by  $T(\mathbf{x}) = A\mathbf{x}$  where  $A$  is an  $n \times n$  matrix. When is  $T$  a contraction w.r.t. the vector norm  $\|\cdot\|_1$ ?

### Exercise 13.6: Solve

Test the function `newtonmult` on the equations given initially in Section 13.1.

### Exercise 13.7: Broyden's method

In this exercise we will implement Broyden's method.

- a) Given a value  $\mathbf{x}_0$ , implement a function which computes an estimate of  $\mathbf{F}'(\mathbf{x}_0)$  by estimating the partial derivatives of  $\mathbf{F}$ , using a numerical differentiation method and step size of your own choosing.
- b) Implement a function

```
function x=broyden(x0,F)
```

which returns an estimate of a zero of  $\mathbf{F}$  using Broyden's method. Your method should set  $B_0$  to be the matrix obtained from the function in a. Just indicate where line search along the search direction should be performed in your function, without implementing it. The function should work as `newtonmult` in that it terminates after a given number of iterations, or after precision of a given accuracy has been obtained.

### Exercise 13.8: Solve

The *Frobenius norm* is defined by

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}$$

for any matrix  $A$ . Show that  $\|AB\|_F \leq \|A\|_F \|B\|_2$  whenever the matrix product  $AB$  is well-defined.

### Exercise 13.9: Solve

Show that  $\|\mathbf{v}\mathbf{w}^T\|_2 = \|\mathbf{v}\| \|\mathbf{w}\|$  for any column vectors  $\mathbf{v}$  and  $\mathbf{w}$ .

### Exercise 13.10: Solve

Show that the matrix  $B_{k+1}$  obtained by Broyden's method (Equation (13.8)) minimizes  $f(B) = \|B - B_k\|_F$  subject to the constraint  $B\mathbf{s}_k = \mathbf{y}_k$ .

Note that the Frobenius norm is not the only norm for which this result holds. It is chosen since it is sensitive to changes in all components in the same way.



# Chapter 14

## Unconstrained optimization

How can we know whether a given point  $\mathbf{x}^*$  is a minimum, local or global, of some given function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ? And how can we find such a point  $\mathbf{x}^*$ ?

These are, of course, some main questions in optimization. In order to give good answers to these questions we need *optimality conditions*. They provide tests for optimality, and serve as the basis for algorithms. We here focus on differentiable functions; the corresponding results for the nondifferentiable case are more difficult (but they exist, and are based on convexity, see [19, 41]).

For unconstrained problems it is not difficult to find powerful optimality conditions from Taylor's theorem for functions in several variables.

### 14.1 Optimality conditions

In order to establish optimality conditions in unconstrained optimization, Taylor's theorem is the starting point, see Section 11.3. We only consider minimization problems, as maximization problems are turned into minimization problems by multiplying the function  $f$  by  $-1$ .

First we look at some necessary optimality conditions.

**Theorem 14.1.** *Minima are stationary points.*

Assume that  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  has continuous partial derivatives, and assume that  $\mathbf{x}^*$  is a local minimum of  $f$ . Then

$$\nabla f(\mathbf{x}^*) = \mathbf{0}. \quad (14.1)$$

If, moreover,  $f$  has continuous second order partial derivatives, then  $\nabla^2 f(\mathbf{x}^*)$  is positive semidefinite.

*Proof.* Assume that  $\mathbf{x}^*$  is a local minimum of  $f$  and that  $\nabla f(\mathbf{x}^*) \neq \mathbf{0}$ . Let  $\mathbf{h} = -\alpha \nabla f(\mathbf{x}^*)$  where  $\alpha > 0$ . Then  $\nabla f(\mathbf{x}^*)^T \mathbf{h} = -\alpha \|\nabla f(\mathbf{x}^*)\|^2 < 0$  and by continuity of the partial derivatives of  $f$ ,  $\nabla f(\mathbf{x})^T \mathbf{h} < 0$  for all  $\mathbf{x}$  in some neighborhood of  $\mathbf{x}^*$ . From Theorem 11.2 (first order Taylor) we obtain

$$f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*) = \nabla f(\mathbf{x}^* + t\mathbf{h})^T \mathbf{h} \quad (14.2)$$

for some  $t \in (0, 1)$  (depending on  $\alpha$ ). By choosing  $\alpha$  small enough, the right-hand side of (14.2) is negative (as just said), and so  $f(\mathbf{x}^* + \mathbf{h}) < f(\mathbf{x}^*)$ , contradicting that  $\mathbf{x}^*$  is a local minimum. This proves that  $\nabla f(\mathbf{x}^*) = \mathbf{0}$ .

To prove the second statement, we get from Theorem 11.3 (second order Taylor)

$$\begin{aligned} f(\mathbf{x}^* + \mathbf{h}) &= f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}^* + t\mathbf{h}) \mathbf{h} \\ &= f(\mathbf{x}^*) + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}^* + t\mathbf{h}) \mathbf{h} \end{aligned} \quad (14.3)$$

If  $\nabla^2 f(\mathbf{x}^*)$  is not positive semidefinite, there is an  $\mathbf{h}$  such that  $\mathbf{h}^T \nabla^2 f(\mathbf{x}^*) \mathbf{h} < 0$  and, by continuity of the second order partial derivatives,  $\mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h} < 0$  for all  $\mathbf{x}$  in some neighborhood of  $\mathbf{x}^*$ . But then (14.3) gives  $f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*) < 0$ ; a contradiction. This proves that  $\nabla^2 f(\mathbf{x})$  is positive semidefinite.  $\square$

The two necessary optimality conditions in Theorem 14.1 are called the *first-order* and the *second-order* conditions, respectively. The first-order condition says that the gradient must be zero at  $\mathbf{x}^*$ , and such a point is often called a *stationary* point. The second-order condition may be interpreted by  $f$  being "convex locally" at  $\mathbf{x}^*$ , although this is not a precise term. A stationary point which is neither a local minimum or a local maximum is called a *saddle point*. So, every neighborhood of a saddle point contains points with larger and points with smaller  $f$ -value.

Theorem 14.1 gives a connection to nonlinear equations. In order to find a stationary point we may solve  $\nabla f(\mathbf{x}) = \mathbf{0}$ , which is a  $n \times n$  (usually nonlinear) system of equations. (The system is linear whenever  $f$  is a quadratic function.) One may solve this equation, for instance, by Newton's method and thereby get a candidate for a local minimum. Sometimes this approach works well, in particular if  $f$  has a unique local minimum and we have an initial point "sufficiently close". However, there are other better methods which we discuss later.

It is important to point out that any algorithm for finding a minimum of  $f$  has to be able to find a stationary point. Therefore algorithms in this area are typically iterative and move to gradually better points where the norm of the gradient becomes smaller, and eventually almost equal to zero.

**A simple example.** Consider a convex quadratic function

$$f(\mathbf{x}) = (1/2) \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$$

where  $A$  is the (symmetric) Hessian matrix is (constant equal to)  $A$  and this matrix is positive semidefinite. Then  $\nabla f(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$  so the first-order necessary optimality condition is

$$A\mathbf{x} = \mathbf{b}$$

which is a linear system of equations. If  $f$  is *strictly convex*, which happens when  $A$  is positive definite, then  $A$  is invertible and the unique solution is  $\mathbf{x}^* = A^{-1}\mathbf{b}$ . Thus, there is only one candidate for a local (and global) minimum, namely  $\mathbf{x}^* = A^{-1}\mathbf{b}$ . Actually, this is indeed a unique global minimum, but to verify this we need a suitable argument. One way is to use convexity (with results presented later) or an alternative is to use *sufficient* optimality conditions which we discuss next. The linear system  $A\mathbf{x} = \mathbf{b}$ , when  $A$  is positive definite, may be solved by several methods. A popular, and very fast, method is the *conjugate gradient method*. This method, and related methods, are discussed in detail in the course INF-MAT4360 *Numerical linear algebra* [28].

In order to present a sufficient optimality condition we need a result from linear algebra. Recall from linear algebra that a symmetric positive definite matrix has only real eigenvalues and all these are positive.

**Lemma 14.2.** *Smallest eigenvalue.*

Let  $A$  be an  $n \times n$  symmetric positive definite matrix, and let  $\lambda_n > 0$  denote its smallest eigenvalue. Then

$$\mathbf{h}^T A \mathbf{h} \geq \lambda_n \|\mathbf{h}\|^2 \quad (\mathbf{h} \in \mathbb{R}^n).$$

*Proof.* By the spectral theorem there is an orthogonal matrix  $P$  (containing the orthonormal eigenvectors as its columns) such that

$$A = P D P^T$$

where  $D$  is the diagonal matrix with the eigenvalues  $\lambda_1, \dots, \lambda_n$  on the diagonal. Let  $\mathbf{h} \in \mathbb{R}^n$  and define  $\mathbf{y} = P^T \mathbf{h}$ . Then  $\|\mathbf{y}\| = \|\mathbf{h}\|$  and

$$\mathbf{h}^T A \mathbf{h} = \mathbf{h}^T P D P^T \mathbf{h} = \mathbf{y}^T D \mathbf{y} = \sum_{j=1}^n \lambda_j y_j^2 \geq \lambda_n \sum_{i=1}^n y_i^2 = \lambda_n \|\mathbf{y}\|^2 = \lambda_n \|\mathbf{h}\|^2.$$

□

Next we consider *sufficient* optimality conditions in the general differentiable case. These conditions are used to prove that a candidate point (say, found by an algorithm) is really a local minimum.

**Theorem 14.3.** *Sufficient conditions for a minimum.*

Assume that  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  has continuous second order partial derivatives in some neighborhood of a point  $\mathbf{x}^*$ . Assume that  $\nabla f(\mathbf{x}^*) = \mathbf{0}$  and  $\nabla^2 f(\mathbf{x}^*)$  is positive definite. Then  $\mathbf{x}^*$  is a local minimum of  $f$ .

*Proof.* From Theorem 11.4 (second order Taylor) and Lemma 14.2 we get

$$\begin{aligned} f(\mathbf{x}^* + \mathbf{h}) &= f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x}^*) \mathbf{h} + \epsilon(\mathbf{h}) \|\mathbf{h}\|^2 \\ &\geq f(\mathbf{x}^*) + \frac{1}{2} \lambda_n \|\mathbf{h}\|^2 + \epsilon(\mathbf{h}) \|\mathbf{h}\|^2 \end{aligned}$$

where  $\lambda_n > 0$  is the smallest eigenvalue of  $\nabla^2 f(\mathbf{x}^*)$ . Dividing here by  $\|\mathbf{h}\|^2$  gives

$$(f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*)) / \|\mathbf{h}\|^2 = \frac{1}{2} \lambda_n + \epsilon(\mathbf{h})$$

Since  $\lim_{\mathbf{h} \rightarrow \mathbf{0}} \epsilon(\mathbf{h}) = 0$ , there is an  $r$  such that for  $\|\mathbf{h}\| < r$ ,  $|\epsilon(\mathbf{h})| < \lambda_n/4$ . This implies that

$$(f(\mathbf{x}^* + \mathbf{h}) - f(\mathbf{x}^*)) / \|\mathbf{h}\|^2 \geq \lambda_n/4$$

for all  $\mathbf{h}$  with  $\|\mathbf{h}\| < r$ . This proves that  $\mathbf{x}^*$  is a local minimum of  $f$ .  $\square$

We remark that the proof of the previous theorem actually shows that  $\mathbf{x}^*$  is a *strict* local minimum of  $f$  meaning that  $f(\mathbf{x}^*)$  is strictly smaller than  $f(\mathbf{x})$  for all other points  $\mathbf{x}$  in some neighborhood of  $\mathbf{x}^*$ . Note the difference between the necessary and the sufficient optimality conditions: a necessary condition is that  $\nabla^2 f(\mathbf{x})$  is positive semidefinite, while a part of the sufficient condition is the stronger property that  $\nabla^2 f(\mathbf{x})$  is positive definite.

Let us see what happens when we work with a convex function.

**Theorem 14.4.** *Minima for convex functions.*

Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a convex function. Then a local minimum is also a global minimum. If, in addition,  $f$  is differentiable, then a point  $\mathbf{x}^*$  is a local (and then global) minimum of  $f$  if and only if

$$\nabla f(\mathbf{x}^*) = \mathbf{0}.$$

*Proof.* Let  $\mathbf{x}_1$  be a local minimum. If  $\mathbf{x}_1$  is not a global minimum, there is an  $\mathbf{x}_2 \neq \mathbf{x}_1$  with  $f(\mathbf{x}_2) < f(\mathbf{x}_1)$ . Then for  $0 < \lambda < 1$

$$f((1 - \lambda)\mathbf{x}_1 + \lambda\mathbf{x}_2) \leq (1 - \lambda)f(\mathbf{x}_1) + \lambda f(\mathbf{x}_2) < f(\mathbf{x}_1)$$

and this contradicts that  $f(\mathbf{x}) \geq f(\mathbf{x}_1)$  for all  $\mathbf{x}$  in a neighborhood of  $\mathbf{x}^*$ . Therefore  $\mathbf{x}_1$  must be a global minimum.

Assume  $f$  is convex and differentiable. Due to Theorem 14.1 we only need to show that if  $\nabla f(\mathbf{x}^*) = \mathbf{0}$ , then  $\mathbf{x}^*$  is a local and global minimum. So assume that  $\nabla f(\mathbf{x}^*) = \mathbf{0}$ . Then, from Theorem 12.10 we have

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*)$$

for all  $\mathbf{x} \in \mathbb{R}^n$ . If  $\nabla f(\mathbf{x}^*) = \mathbf{0}$ , this directly shows that  $\mathbf{x}^*$  is a global minimum.  $\square$

## 14.2 Methods

Algorithms for unconstrained optimization are iterative methods that generate a sequence of points with gradually smaller values on the function  $f$  which is to be minimized. There are two main types of algorithms in unconstrained optimization:

- *Line search methods*: Here one first chooses a *search direction*  $\mathbf{d}_k$  from the current point  $\mathbf{x}_k$ , using information about the function  $f$ . Then one chooses a *step length*  $\alpha_k$  so that the new point  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$  has a small, perhaps smallest possible, value on the half-line  $\{\mathbf{x}_k + \alpha \mathbf{d}_k : \alpha \geq 0\}$ .  $\alpha_k$  describes how far one should go along the search direction. The problem of choosing  $\alpha_k$  is a one-dimensional optimization problem. Sometimes we can find  $\alpha_k$  exactly, and in such cases we refer to the method as *exact line search*. In cases where  $\alpha_k$  can not be found analytically, algorithms can be used to approximate how we can get close to the minimum on the half-line. Such a method is also referred to as *inexact line search*.
- *Trust region methods*: In these methods one chooses an approximation  $\hat{f}_k$  to the function in some neighborhood of the current point  $\mathbf{x}_k$ . The function  $\hat{f}_k$  is simpler than  $f$  and one minimizes  $\hat{f}_k$  (in the mentioned neighborhood) and let the next iterate  $\mathbf{x}_{k+1}$  be this minimizer.

These types are typically both based on quadratic approximation of  $f$ , but they differ in the order in which one chooses search direction and step size. In the following we only discuss the first type, the line search methods.

A very natural choice for search direction at a point  $\mathbf{x}_k$  is the negative gradient,  $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$ . Recall that the direction of maximum increase of a (differentiable) function  $f$  at a point  $\mathbf{x}$  is  $\nabla f(\mathbf{x})$ , and the direction of maximum decrease is  $-\nabla f(\mathbf{x})$ . To verify this, Taylor's theorem gives

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2} \mathbf{h}^T \nabla^2 f(\mathbf{x} + t\mathbf{h}) \mathbf{h}.$$

So, for small  $\mathbf{h}$ , the first order term dominates and we would like to make this term small. By the Cauchy-Schwarz inequality<sup>1</sup>.

$$\nabla f(\mathbf{x})^T \mathbf{h} \geq -\|\nabla f(\mathbf{x})\| \|\mathbf{h}\|$$

and equality holds for  $\mathbf{h} = -\alpha \nabla f(\mathbf{x})$  for some  $\alpha \geq 0$ . In general, we call  $\mathbf{h}$  a *descent direction* at  $\mathbf{x}$  if  $\nabla f(\mathbf{x})^T \mathbf{h} < 0$ . Thus, if we move in a descent direction from  $\mathbf{x}$  and make a sufficiently small step, the new point has a smaller  $f$ -value. With this background we shall in the following focus on *gradient methods* given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \tag{14.4}$$

<sup>1</sup>The Cauchy-Schwarz' inequality says:  $|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\| \|\mathbf{v}\|$  for  $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ .

where the direction  $\mathbf{d}_k$  satisfies

$$\nabla f(\mathbf{x}_k)^T \mathbf{d}_k < 0 \quad (14.5)$$

There are two gradient methods we shall discuss:

**The steepest descent method.** Here we choose the search direction  $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$ , we get

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k).$$

In each step it moves in the direction of the negative gradient. Sometimes this gives slow convergence, so other methods have been developed where other choices of direction  $\mathbf{d}_k$  are made.

**Newton's method.** Here we choose

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k). \quad (14.6)$$

This is the gradient method with  $\mathbf{d}_k = -(\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$ ; this vector  $\mathbf{d}_k$  is called the *Newton step*. The so-called *pure Newton method* is when one simply chooses step size  $\alpha_k = 1$  for each  $k$ . We then also say that we take a *full Newton step*. To interpret this method consider the second order Taylor approximation of  $f$  in  $\mathbf{x}_k$

$$f(\mathbf{x}_k + \mathbf{h}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{h} + (1/2) \mathbf{h}^T \nabla^2 f(\mathbf{x}_k) \mathbf{h}$$

If we minimize this quadratic function w.r.t.  $\mathbf{h}$ , assuming  $\nabla^2 f(\mathbf{x}_k)$  is positive definite, we get (see Exercise 14.8)

$$\mathbf{h} = -(\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$$

which explains the Newton step.

In the following we follow the presentation in [2]. In a gradient method we need to *choose the step length*. This is the one-dimensional optimization problem

$$\min\{f(\mathbf{x} + \alpha \mathbf{d}) : \alpha \geq 0\}.$$

Sometimes (maybe not too often) we may solve this problem exactly. Most practical methods try some candidate  $\alpha$ 's and pick the one with smallest  $f$ -value. Note that it is not necessary to compute the exact minimum (this may take too much time). The main thing is to assure that we get a sufficiently large decrease in  $f$  without making a too small step.

A popular method for choosing the step size is *backtracking line search*:

**Definition 14.5.** *Backtracking line search.*

The method of backtracking line search for choosing a step size is defined as follows: We assume that (in advance) we have chosen parameters  $s \leq 1$ , a

reduction factor  $\beta$  satisfying  $0 < \beta < 1$ , and  $0 < \sigma < 1$  (typically this is chosen very small, e.g.  $\sigma = 10^{-3}$ ). We define the integer

$$m_k = \min\{m : m \geq 0, f(\mathbf{x}_k) - f(\mathbf{x}_k + \beta^m s \mathbf{d}_k) \geq -\sigma \beta^m s \nabla f(\mathbf{x}_k)^T \mathbf{d}_k\}. \quad (14.7)$$

The step size is then defined to be  $\alpha_k = \beta^{m_k} s$ . The inequality

$$f(\mathbf{x}_k) - f(\mathbf{x}_k + \beta^m s \mathbf{d}_k) \geq -\sigma \beta^m s \nabla f(\mathbf{x}_k)^T \mathbf{d}_k \quad (14.8)$$

is also called the *stopping condition of backtracking line search*.

The parameter  $s$  fixes the search for step size to lie within the interval  $[0, s]$ . This can be important: for instance, we can set  $s$  so small that the initial step size we try is within the domain of definition for  $f$ . The natural thing would be to choose  $s = 1$ : if the stopping condition then applies immediately, then  $\alpha_k = 1$ . If Newton's method is used this corresponds to using the pure Newton step, i.e. a full Newton step is chosen.

According to [2]  $\beta$  is usually chosen in  $[1/10, 1/2]$ . In the literature one may find a lot more information about step size rules and how they may be adjusted to the methods for finding search direction, see [2], [32].

Now, we return to the *choice of search direction* in the gradient method (14.4). A main question is whether it generates a sequence  $\{\mathbf{x}_k\}_{k=1}^{\infty}$  which converges to a stationary point  $\mathbf{x}^*$ , i.e., where  $\nabla f(\mathbf{x}^*) = \mathbf{0}$ . It turns out that this may not be the case; one needs to be careful about the choice of  $\mathbf{d}_k$  to assure this convergence. The problem is that if  $\mathbf{d}_k$  tends to be nearly orthogonal to  $\nabla f(\mathbf{x}_k)$  one may get into trouble. For this reason one introduces the following notion:

**Definition 14.6.** *Gradient related.*

$\{\mathbf{d}_k\}$  is called *gradient related* to  $\{\mathbf{x}_k\}$  if for any subsequence  $\{\mathbf{x}_{k_p}\}_{p=1}^{\infty}$  of  $\{\mathbf{x}_k\}$  converging to a nonstationary point, then the corresponding subsequence  $\{\mathbf{d}_{k_p}\}_{p=1}^{\infty}$  of  $\{\mathbf{d}_k\}$  is bounded and  $\limsup_{p \rightarrow \infty} \nabla f(\mathbf{x}_{k_p})^T \mathbf{d}_{k_p} < 0$ .

What this condition assures is that  $\|\mathbf{d}_k\|$  is not too small or large compared to  $\|\nabla f(\mathbf{x}_k)\|$  and that the angle between the vectors  $\mathbf{d}_k$  and  $\nabla f(\mathbf{x}_k)$  is not too close to  $90^\circ$ . The proof of the following theorem may be found in [2].

**Theorem 14.7.** *Backtracking line search and gradient related.*

Let  $\{\mathbf{x}_k\}_{k=0}^{\infty}$  be generated by the gradient method (14.4), where  $\{\mathbf{d}_k\}_{k=0}^{\infty}$  is gradient related to  $\{\mathbf{x}_k\}_{k=0}^{\infty}$  and the step size  $\alpha_k$  is chosen using backtracking line search. Then every limit point of  $\{\mathbf{x}_k\}_{k=0}^{\infty}$  is a stationary point.

We remark that in Theorem 14.7 the same conclusion holds if we use exact minimization as step size rule, i.e.,  $f(\mathbf{x}_k + \alpha \mathbf{d}_k)$  is minimized exactly with respect to  $\alpha$ .

A very important property of a numerical algorithm is its convergence speed. Let us consider the steepest descent method first. It turns out that the convergence speed for this algorithm is very well explained by its performance on minimizing a quadratic function, so therefore the following result is important.

In this theorem  $A$  is a symmetric positive definite matrix with eigenvalues  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n > 0$ .

**Theorem 14.8.** *Minima and teh smallest eigenvalue.*

If the steepest descent method  $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$  using exact line search is applied to the quadratic function  $f(\mathbf{x}) = \mathbf{x}^T A \mathbf{x}$  where  $A$  is positive definite, then (the minimum value is 0 and)

$$f(\mathbf{x}_{k+1}) \leq m_A f(\mathbf{x}_k)$$

where  $m_A = ((\lambda_1 - \lambda_n)/(\lambda_1 + \lambda_n))^2$ .

The proof may be found in [2]. Thus, if the largest eigenvalue is much larger than the smallest one,  $m_A$  will be nearly 1 and one typically have slow convergence. In this case we have  $m_A \approx \text{cond}(A)$  where  $\text{cond}(A) = \lambda_1/\lambda_n$  is the *condition number* of the matrix  $A$ . So the rule is: if the condition number of  $A$  is small we get fast convergence, but if  $\text{cond}(A)$  is large, there will be slow convergence. A similar behavior holds for most functions  $f$  because locally near a minimum point the function is very close to its second order Taylor approximation in  $\mathbf{x}^*$  which is a quadratic function with  $A = \nabla^2 f(\mathbf{x}^*)$ .

Thus, Theorem 14.8 says that the sequence obtained in the steepest descent method converges linearly to a stationary point (at least for quadratic functions).

We now turn to Newton's method.

Recall that the pure Newton step minimizes the second order Taylor approximation of  $f$  at the current iterate  $\mathbf{x}_k$ . Thus, if the function we minimize is quadratic, we are done in one step. Similarly, if the function can be well approximated by a quadratic function, then one would expect fast convergence.

We shall give a result on the convergence of Newton's method (see [3] for further details). When  $A$  is symmetric, we let  $\lambda_{\min}(A)$  denote that smallest eigenvalue of  $A$ .

For the convergence result we need a lemma on strictly convex functions. Assume that  $\mathbf{x}_0$  is a starting point for Newton's method and let  $S = \{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$ . We shall assume that  $f$  is continuous and convex, and this implies that  $S$  is a closed convex set. We also assume that  $f$  has a minimum point  $\mathbf{x}^*$  which then must be a global minimum. Moreover the minimum point will be unique due to a strict convexity assumption on  $f$ . Let  $f^* = f(\mathbf{x}^*)$  be the optimal value.

The following lemma says that for a convex function as just described, a point is nearly a minimum point (in terms of the  $f$ -value) whenever the gradient is small in that point.

**Lemma 14.9.** *Norm of teh gradient.*

Assume that  $f$  is convex as above and that  $\lambda_{\min}(\nabla^2 f(\mathbf{x})) \geq m$  for all  $\mathbf{x} \in S$ . Then

$$f(\mathbf{x}) - f^* \leq \frac{1}{2m} \|\nabla f(\mathbf{x})\|^2. \quad (14.9)$$



*Proof.* From Theorem 11.3, the second order Taylor's theorem, we have for each  $\mathbf{x}, \mathbf{y} \in S$

$$f(\mathbf{y}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + (1/2)(\mathbf{y} - \mathbf{x})^T \nabla^2 f(\mathbf{z})(\mathbf{y} - \mathbf{x})$$

for suitable  $\mathbf{z}$  on the line segment between  $\mathbf{x}$  and  $\mathbf{y}$ . Here a lower bound for the quadratic term is  $(m/2)\|\mathbf{y} - \mathbf{x}\|^2$ , due to Lemma 14.2. Therefore

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + (m/2)\|\mathbf{y} - \mathbf{x}\|^2.$$

Now, fix  $\mathbf{x}$  and view the expression on the right-hand side as a quadratic function of  $\mathbf{y}$ . This function is minimized for  $\mathbf{y}^* = \mathbf{x} - (1/m)\nabla f(\mathbf{x})$ . So, by inserting  $\mathbf{y} = \mathbf{y}^*$  above we get

$$\begin{aligned} f(\mathbf{y}) &\geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}^* - \mathbf{x}) + (m/2)\|\mathbf{y}^* - \mathbf{x}\|^2 \\ &= f(\mathbf{x}) - \frac{1}{2m}\|\nabla f(\mathbf{x})\|^2 \end{aligned}$$

This holds for every  $\mathbf{y} \in S$  so letting  $\mathbf{y} = \mathbf{x}^*$  gives

$$f^* = f(\mathbf{x}^*) \geq f(\mathbf{x}) - \frac{1}{2m}\|\nabla f(\mathbf{x})\|^2$$

which proves the desired inequality.  $\square$

In the following convergence result we consider a function  $f$  as in Lemma 14.9. Moreover, we assume that the Hessian matrix is Lipschitz continuous over  $S$ ; this is essentially a bound on the third derivatives of  $f$ . We do not give the complete proof (it is quite long), but consider some of the main ideas. Recall the definition of the set  $S$  from above.

**Theorem 14.10.** *Quadratic convergence of Newton's method for convex functions.*

Let  $f$  be convex and twice continuously differentiable and assume that

1.  $\lambda_{\min}(\nabla^2 f(\mathbf{x})) \geq m$  for all  $\mathbf{x} \in S$ .
2.  $\|\nabla^2 f(\mathbf{x}) - \nabla^2 f(\mathbf{y})\|_2 \leq L\|\mathbf{x} - \mathbf{y}\|$  for all  $\mathbf{x} \in S$ .

Moreover, assume that  $f$  has a minimum point  $\mathbf{x}^*$ . Then Newton's method generates a sequence  $\{\mathbf{x}_k\}_{k=0}^{\infty}$  that converges to  $\mathbf{x}^*$ . From a certain  $k'$  the convergence speed is quadratic.

*Proof.* The proof is based on [3]). Define  $f^* = f(\mathbf{x}^*)$ . We will prove the result by establishing two lemmas. The proofs of these lemmas are rather technical, so they are put in their own sections which are not part of the curriculum, and are only included for the sake of completeness.

The first lemma applies to the first iterations of Newton's method. In this phase the convergence of the method may be slow, and we will see that backtracking line search may choose a step size which is very small. This phase of Newton's method is therefore called the *damped Newton phase*:

**Lemma 14.11.** *First lemma.*

For any  $\eta$ , there exists  $\gamma > 0$  so that, for each  $k$ , if  $\|\nabla f(\mathbf{x}_k)\| \geq \eta$ , then

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \gamma. \quad (14.10)$$

The proof can be found in Section 14.2. After the damped Newton phase, the Newton method will enter a phase where the convergence is much quicker, as the following result shows. It is in this phase that we have a quadratical convergence rate, so that this phase also is called the *quadratically convergent phase*. It turns out that backtracking line search always chooses a step size equal to 1 in this phase:

**Lemma 14.12.** *Second lemma.*

There exists  $\eta$  with  $0 < \eta \leq m^2/L$  so that, for each  $k$ , if  $\|\nabla f(\mathbf{x}_k)\| < \eta$ , then  $\alpha_k = 1$  satisfies the stopping criterion of backtracking line search in Newton's method, and Newton's method with backtracking line search gives

$$\frac{L}{2m^2} \|\nabla f(\mathbf{x}_{k+1})\| \leq \left( \frac{L}{2m^2} \|\nabla f(\mathbf{x}_k)\| \right)^2. \quad (14.11)$$

The proof can be found in Section 14.2. Now, let us combine these two lemmas to prove the theorem. In each iteration where (14.10) occurs  $f$  is decreased by at least  $\gamma$ , so the number of such iterations must be bounded by

$$(f(\mathbf{x}_0) - f^*)/\gamma$$

which is a finite number. For some  $k$  we must thus have by Lemma 14.11 that  $\|\nabla f(\mathbf{x}_k)\| < \eta$ , and we can then use (14.11) and Lemma 14.12 to obtain

$$\begin{aligned} \|\nabla f(\mathbf{x}_{k+1})\| &\leq \frac{2m^2}{L} \left( \frac{L}{2m^2} \|\nabla f(\mathbf{x}_k)\| \right)^2 \\ &= \frac{L}{2m^2} \|\nabla f(\mathbf{x}_k)\|^2 \leq \frac{L}{2m^2} \eta^2 = \frac{1}{2} \frac{L}{m^2} \eta \leq \frac{1}{2} \eta \leq \eta. \end{aligned}$$

Therefore, as soon as (14.11) occurs in the iterative process, in all the remaining iterations (14.11) will occur. Actually, let us show that as soon as (14.11) “kicks in”, quadratic convergence starts:

Define  $\mu_l = \frac{L}{2m^2} \|\nabla f(\mathbf{x}_l)\|$  for each  $l \geq k$ . Then  $0 \leq \mu_k < 1/2$  as  $\eta \leq m^2/L$ . From (14.11) it follows that

$$\mu_{l+1} \leq \mu_l^2 \quad (l \geq k).$$

So (by induction)

$$\mu_l \leq \mu_k^{2^{l-k}} \leq (1/2)^{2^{l-k}} \quad (l = k+1, k+2, \dots).$$

Next, from Lemma 14.9

$$\begin{aligned} f(\mathbf{x}_l) - f^* &\leq \frac{1}{2m} \|\nabla f(\mathbf{x}_l)\|^2 = \frac{1}{2m} \frac{4m^4}{L^2} \left( \frac{L}{2m^2} \|\nabla f(\mathbf{x}_l)\| \right)^2 \\ &= \frac{2m^3}{L^2} \mu_l^2 \leq \frac{2m^3}{L^2} (1/2)^{2^{l-k+1}}, \end{aligned}$$

for  $l \geq k$ . This inequality shows that  $f(\mathbf{x}_l) \rightarrow f^*$ , and since the minimum point is unique due to convexity, we must have  $\mathbf{x}_l \rightarrow \mathbf{x}^*$ . It follows that the convergence is quadratic.  $\square$

From the proof it is also possible to say something about how many iterations that are needed to reach a certain accuracy. In fact, if  $\epsilon > 0$  a bound on the number of iterations until  $f(\mathbf{x}_k) \leq f^* + \epsilon$  is

$$(f(\mathbf{x}_0) - f^*)/\gamma + \log_2 \log_2 \frac{2m^3}{\epsilon L^2}.$$

Here  $\gamma$  is the parameter introduced in the proof above. The second term in this expression (the logarithmic term) grows very slowly as  $\epsilon$  is decreased, and it may roughly be replaced by the constant 6. So, whenever the second stage (14.11) occurs, the convergence is extremely fast, it takes about 6 more Newton iterations. Note that quadratic convergence means, roughly, that the number of correct digits in the answer doubles for every iteration.

**\*The proof for Lemma 14.11.** We have that

$$\|\mathbf{d}_k\|^2 = (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-2} \nabla f(\mathbf{x}_k) \leq \frac{1}{m} (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k),$$

since the largest eigenvalue of  $(\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-2} \nabla f(\mathbf{x}_k)$  is less than  $1/m$ . Since there also is an upper bound  $M$  on the highest eigenvalue of  $\nabla^2 f(\mathbf{x})$ , the second order Taylor approximation gives

$$\begin{aligned} f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) &= f(\mathbf{x}_k) + \alpha_k \nabla f(\mathbf{x}_k)^T \mathbf{d}_k + \frac{1}{2} (\alpha_k)^2 (\mathbf{d}_k)^T \nabla^2 f(\mathbf{z}) \mathbf{d}_k \\ &\leq f(\mathbf{x}_k) + \alpha_k \nabla f(\mathbf{x}_k)^T \mathbf{d}_k + \frac{M \|\mathbf{d}_k\|^2}{2} \alpha_k^2 \\ &\leq f(\mathbf{x}_k) - \alpha_k (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k) + \frac{M (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)}{2m} \alpha_k^2 \end{aligned}$$

If we try the value  $\hat{\alpha}_k = m/M$  we get

$$f(\mathbf{x}_k + \hat{\alpha}_k \mathbf{d}_k) \leq f(\mathbf{x}_k) - \frac{1}{2} \hat{\alpha}_k (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k),$$

which can be written as

$$\begin{aligned}
f(\mathbf{x}_k) - f(\mathbf{x}_k + \hat{\alpha}_k \mathbf{d}_k) &\geq \frac{1}{2} \hat{\alpha}_k (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k) \\
&\geq \sigma \hat{\alpha}_k (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k) \\
&= -\sigma \hat{\alpha}_k (\nabla f(\mathbf{x}_k))^T \mathbf{d}_k,
\end{aligned}$$

which shows that  $\hat{\alpha}_k = m/M$  satisfies the stopping criterion of backtracking line search. Since we may not have exactly  $m/M = \beta^n s$  for some  $n$ , we may still conclude that backtracking line search stops at  $\alpha_k \geq \beta m/M$ , so that

$$\begin{aligned}
f(\mathbf{x}_{k+1}) &\leq f(\mathbf{x}_k) + \sigma \hat{\alpha}_k (\nabla f(\mathbf{x}_k))^T \mathbf{d}_k \\
&\leq f(\mathbf{x}_k) - \sigma \alpha_k (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k) \\
&\leq f(\mathbf{x}_k) - \sigma \beta \frac{m}{M} \frac{1}{M} \|\nabla f(\mathbf{x}_k)\|^2 \\
&\leq f(\mathbf{x}_k) - \sigma \beta \eta^2 \frac{m}{M^2}.
\end{aligned}$$

This shows that we can choose  $\gamma = \sigma \beta \eta^2 \frac{m}{M^2}$ .

**\*The proof for Lemma 14.12.** We will first show that backtracking line search chooses unit steps provided that  $\eta \leq 3(1 - 2\sigma)m^2/L$ . By condition (ii),

$$\|\nabla^2 f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) - \nabla^2 f(\mathbf{x}_k)\|_2 \leq \alpha_k L \|\mathbf{d}_k\|,$$

so that

$$|(\mathbf{d}_k)^T (\nabla^2 f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) - \nabla^2 f(\mathbf{x}_k)) \mathbf{d}_k| \leq \alpha_k L \|\mathbf{d}_k\|^3.$$

Now we define the function  $g(t) = f(\mathbf{x} + t\mathbf{d}_k)$ . The chain rule gives that

$$g'(t) = \nabla f(\mathbf{x} + t\mathbf{d}_k)^T \mathbf{d}_k \quad g''(t) = (\mathbf{d}_k)^T \nabla^2 f(\mathbf{x}_k + t\mathbf{d}_k) \mathbf{d}_k.$$

In particular, note that  $g''(0) = (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$ . The inequality above can therefore be written as

$$|g''(t) - g''(0)| \leq tL \|\mathbf{d}_k\|^3,$$

so that

$$\begin{aligned}
g''(t) &\leq g''(0) + tL \|\mathbf{d}_k\|^3 \\
&\leq (\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k) + t \frac{L}{m^{3/2}} ((\nabla f(\mathbf{x}_k))^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k))^{3/2}
\end{aligned}$$

where we have used that

$$m\|\mathbf{d}_k\|^2 = m(\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-2}\nabla f(\mathbf{x}_k) \leq (\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k).$$

We integrate this inequality to get

$$\begin{aligned} g'(t) &\leq g'(0) + t(\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k) \\ &\quad + t^2 \frac{L}{2m^{3/2}}((\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k))^{3/2} \\ &= -(\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k) + t(\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k) \\ &\quad + t^2 \frac{L}{2m^{3/2}}((\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k))^{3/2}. \end{aligned}$$

We integrate this once more and get

$$\begin{aligned} g(t) &\leq g(0) - t(\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k) \\ &\quad + t^2 \frac{1}{2}(\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k) \\ &\quad + t^3 \frac{L}{6m^{3/2}}((\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k))^{3/2}. \end{aligned}$$

If we here set  $t = 1$  we get

$$\begin{aligned} f(\mathbf{x}_k + \mathbf{d}_k) &\leq f(\mathbf{x}) - \frac{1}{2}(\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k) \\ &\quad + \frac{L}{6m^{3/2}}((\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k))^{3/2}. \end{aligned}$$

Assume now that also  $\|\nabla f(\mathbf{x}_k)\| \leq 3(1 - 2\sigma)m^2/L$ . Since the biggest eigenvalue of  $(\nabla^2 f(\mathbf{x}_k))^{-1}$  is less than  $1/m$ , we have that

$$(\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k) \leq \frac{1}{m}(3(1 - 2\sigma)m^2/L)^2 = (3(1 - 2\sigma)m^{3/2}/L)^2.$$

This implies that

$$1/2 - \frac{L((\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k))^{1/2}}{6m^{3/2}} \geq \sigma.$$

We therefore have that

$$\begin{aligned} f(\mathbf{x}_k + \mathbf{d}_k) &\leq f(\mathbf{x}) - \nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k) \left( \frac{1}{2} - \frac{L}{6m^{3/2}}((\nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k))^{1/2} \right) \\ &\leq f(\mathbf{x}) - \sigma \nabla f(\mathbf{x}_k))^T(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k) = f(\mathbf{x}) + \sigma \nabla f(\mathbf{x}_k))^T \mathbf{d}_k, \end{aligned}$$

which proves that  $\alpha_k = 1$  is accepted by the stopping criterion of backtracking line search. We also have that

$$\begin{aligned} \|\nabla f(\mathbf{x}_k + \mathbf{d}_k)\| &= \|\nabla f(\mathbf{x}_k + \mathbf{d}_k) - \nabla f(\mathbf{x}_k) - \nabla^2 f(\mathbf{x}_k)\mathbf{d}_k\| \\ &= \left\| \int_0^1 (\nabla^2 f(\mathbf{x}_k + t\mathbf{d}_k) - \nabla^2 f(\mathbf{x}_k))\mathbf{d}_k dt \right\| \\ &\leq \int_0^1 \|(\nabla^2 f(\mathbf{x}_k + t\mathbf{d}_k) - \nabla^2 f(\mathbf{x}_k))\mathbf{d}_k\|_2 dt \\ &\leq \int_0^1 t \|\mathbf{d}_k\|^2 dt = \frac{L}{2} \|\mathbf{d}_k\|^2 = \frac{L}{2} \|(\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)\|^2 \\ &\leq \frac{L}{2m^2} \|\nabla f(\mathbf{x}_k)\|^2. \end{aligned}$$

This proves the lemma.

### Exercise 14.1: Solve

Consider the function  $f(x_1, x_2) = x_1^2 + ax_2^2$  where  $a > 0$  is a parameter. Draw some of the level sets of  $f$  (for different levels) for each  $a$  in the set  $\{1, 4, 100\}$ . Also draw the gradient in a few points on these level sets.

### Exercise 14.2: Solve

State and prove a theorem similar to Theorem 14.1 for maximization problems.

### Exercise 14.3: Solve

Let  $f(\mathbf{x}) = \mathbf{x}^T A \mathbf{x}$  where  $A$  is a symmetric  $n \times n$  matrix. Assume that  $A$  is indefinite, so it has both positive and negative eigenvalues. Show that  $\mathbf{x} = \mathbf{0}$  is a saddlepoint of  $f$ .

### Exercise 14.4: Solve

Let  $f(x_1, x_2) = 4x_1 + 6x_2 + x_1^2 + 2x_2^2$ . Find all stationary points and determine if they are minimum, maximum or saddlepoints. Do the same for the function  $g(x_1, x_2) = 4x_1 + 6x_2 + x_1^2 - 2x_2^2$ .

### Exercise 14.5: Solve

Let the function  $f$  be given by  $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 2)^2 + 1$ .

- Compute the search direction  $\mathbf{d}_k$  which is chosen by the steepest descent method in the point  $\mathbf{x}_k = (2, 3)$ .
- Compute in the same way the search direction  $\mathbf{d}_k$  which is chosen when we instead use Newton's method in the point  $\mathbf{x}_k = (2, 3)$ .

**Exercise 14.6: Solve**

The function  $f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$  is called the *Rosenbrock function*. Compute the gradient and the Hessian matrix at every point  $\mathbf{x}$ . Find every local minimum. Also draw some of the level sets (contour lines) of  $f$ .

**Exercise 14.7: When steepest descent finds the minimum in one step**

Let  $f(\mathbf{x}) = (1/2)\mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$  where  $A$  is a positive definite  $n \times n$  matrix. Consider the steepest descent method applied to the minimization of  $f$ , where we assume exact line search is used. Assume that the search direction happens to be equal to an eigenvector of  $A$ . Show that then the minimum is reached in just one step.

**Hint.** Start by writing out one step with Newton's method when the search direction happens to be equal to an eigenvector of  $A$ , and establish a connection with the steepest descent method.

**Exercise 14.8: Solve**

Consider the second order Taylor approximation

$$T_f^2(\mathbf{x}; \mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + (1/2)\mathbf{h}^T \nabla^2 f(\mathbf{x}) \mathbf{h}.$$

- a) Show that  $\nabla_{\mathbf{h}} T_f^2 = \nabla f(\mathbf{x}) + \nabla^2 f(\mathbf{x}) \mathbf{h}$ .
- b) Minimizing  $T_f^2$  with respect to  $\mathbf{h}$  implies solving  $\nabla_{\mathbf{h}} T_f^2 = \mathbf{0}$ , i.e.  $\nabla f(\mathbf{x}) + \nabla^2 f(\mathbf{x}) \mathbf{h} = \mathbf{0}$  from a.. If  $\nabla^2 f(\mathbf{x})$  is positive definite, explain that it also is invertible, so that this equation has the unique solution  $\mathbf{h} = -(\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$ , as previously noted for the Newton step.

**Exercise 14.9: Solve**

We want to find the minimum of  $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$ , defined on  $\mathbb{R}^n$ . Formulate one step with Newton's method, and one step with the steepest descent method, where you set the step size to  $\alpha_k = 1$ . Which of these methods works best for finding the minimum for functions on this form?

**Exercise 14.10: Solve**

Implement the steepest descent method. Test the algorithm on the functions in exercises 14.4 and 14.6. Use different starting points.

**Exercise 14.11: Solve**

What can go wrong when you apply backtracking line search (Equation (14.7)) to a function  $f$  where  $\nabla^2 f$  is not negative definite (i.e. all eigenvalues of  $\nabla^2 f$  are negative)?

**Hint.** Substitute the Taylor approximation

$$f(\mathbf{x}_k + \beta^m s \mathbf{d}_k) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T (\beta^m s \mathbf{d}_k)$$

in Equation (14.7), and remember that  $\sigma$  there is chosen so that  $\sigma < 1$ .

**Exercise 14.12: Solve**

Write a function `newtonbacktrack` which performs Newton's method for unconstrained optimization. The input parameters are the function, its gradient, its Hesse matrix, and the initial point. The function should also return the number of iterations, and at each iteration write the corresponding function value. Use backtracking line search to compute the step size, i.e. compute  $m_k$  from Equation (14.7) with  $\beta = 0.2$ ,  $s = 0.5$ ,  $\sigma = 10^{-3}$ , and use  $\alpha = \beta^{m_k} s$  as the step size. Test the algorithm on the functions in exercises 14.4 and 14.6. Use different starting points.

**Exercise 14.13: Solve**

Let us return to the maximum likelihood example on the desintegration of muons.

a) Run the function `newtonbacktrack` with parameters being the function  $f$  and its and derivatives defined as in the example with  $n = 10$  and

$$\mathbf{x} = (0.4992, -0.8661, 0.7916, 0.9107, 0.5357, 0.6574, 0.6353, 0.0342, 0.4988, -0.4607)$$

Use the start value  $\alpha_0 = 0$  for Newton's method. What estimate for the minimum of  $f$  (and thereby  $\alpha$ ) did you obtain?

b) The ten measurements from a) were generated from a probability distribution where  $\alpha = 0.5$ . The answer you obtained was quite far from this. Let us therefore take a look at how many measurements we should use in order to get quite precise estimates for  $\alpha$ . You can use the function

```
function ret=randmuon(alpha,m,n)
```

to generate an  $m \times n$ -matrix with measurements generated with a probability distribution with a given parameter  $\alpha$ . This function can be found at the homepage of the book.

With  $\alpha = 0.5$ , generate  $n = 10$  measurements with the help of the function `randmuon`, and find the maximum likelihood estimate as above. Repeat this



10 times, and plot the ten estimates you obtain. Repeat for  $n = 1000$ , and for  $n = 100000$  (in all cases you are supposed to plot 10 maximum likelihood estimates). How many measurements do we need in order to obtain maximum likelihood estimates which are reliable?

Note that it is possible for the maximum likelihood estimates you obtain here to be outside the domain of definition  $[-1, 1]$ . You need not take this into account.

## Chapter 15

# Constrained optimization - theory

In this chapter we consider constrained optimization problems. A general optimization problem is

$$\text{minimize } f(\mathbf{x}) \text{ subject to } \mathbf{x} \in S.$$

where  $S \subseteq \mathbb{R}^n$  is a given set and  $f : S \rightarrow \mathbb{R}$ . We here focus on a very general optimization problem which often occurs in applications. Consider the *nonlinear optimization problem* with equality/inequality constraints

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && \\ & && h_i(\mathbf{x}) = 0 \quad (i \leq m) \\ & && g_j(\mathbf{x}) \leq 0 \quad (j \leq r) \end{aligned} \tag{15.1}$$

where  $f, h_1, h_2, \dots, h_m$  and  $g_1, g_2, \dots, g_r$  are continuously differentiable functions from  $\mathbb{R}^n$  into  $\mathbb{R}$ . A point  $\mathbf{x}$  satisfying all the  $m + r$  constraints will be called *feasible*. Thus, we look for a feasible point with smallest  $f$ -value.

Our goal is to establish optimality conditions for this problem, starting with the special case with only equality constraints. Then we discuss algorithms for solving this problem. Our presentation is strongly influenced by [3] and [2].

### 15.1 Equality constraints and the Lagrangian

Consider the nonlinear optimization problem with equality constraints

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && \\ & && h_i(\mathbf{x}) = 0 \quad (i \leq m) \end{aligned} \tag{15.2}$$

where  $f$  and  $h_1, h_2, \dots, h_m$  are continuously differentiable functions from  $\mathbb{R}^n$  into  $\mathbb{R}$ . We introduce the vector field  $\mathbf{H} = (h_1, h_2, \dots, h_m)$ , so  $\mathbf{H} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  and  $\mathbf{H}(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_m(\mathbf{x}))$ .

We first establish necessary optimality conditions for this problem. A point  $\mathbf{x}^* \in \mathbb{R}^n$  is called *regular* if the gradient vectors  $\nabla h_i(\mathbf{x}^*)$  ( $i \leq m$ ) are linearly independent.

**Theorem 15.1.** *Lagrange.*

Let  $\mathbf{x}^*$  be a local minimum in problem (15.1) and assume that  $\mathbf{x}^*$  is a regular point. Then there is a unique vector  $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \dots, \lambda_m^*) \in \mathbb{R}^m$  such that

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0}. \quad (15.3)$$

If  $f$  and each  $h_i$  are twice continuously differentiable, then the following also holds

$$\mathbf{h}^T (\nabla^2 f(\mathbf{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla^2 h_i(\mathbf{x}^*)) \mathbf{h} \geq 0 \quad \text{for all } \mathbf{h} \in T(\mathbf{x}^*) \quad (15.4)$$

where  $T(\mathbf{x}^*)$  is the subspace  $T(\mathbf{x}^*) = \{\mathbf{h} \in \mathbb{R}^n : \nabla h_i(\mathbf{x}^*) \cdot \mathbf{h} = 0 \ (i \leq m)\}$ .

The numbers  $\lambda_i^*$  in this theorem are called the *Lagrangian multipliers*. Note that the Lagrangian multiplier vector  $\boldsymbol{\lambda}^*$  is unique; this follows directly from the linear independence assumption as  $\mathbf{x}^*$  is assumed regular. The theorem may also be stated in terms of the *Lagrangian function*  $L : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$  given by

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^m \lambda_i h_i(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{H}(\mathbf{x}) \quad (\mathbf{x} \in \mathbb{R}^n, \boldsymbol{\lambda} \in \mathbb{R}^m).$$

Then

$$\begin{aligned} \nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}) &= \nabla f(\mathbf{x}) + \sum_i \lambda_i \nabla h_i \\ \nabla_{\boldsymbol{\lambda}} L(\mathbf{x}, \boldsymbol{\lambda}) &= \mathbf{H}(\mathbf{x}). \end{aligned}$$

Therefore, the first order conditions in Theorem 15.1 may be rewritten as follows

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}, \quad \nabla_{\boldsymbol{\lambda}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}.$$

Here the second equation simply means that  $\mathbf{H}(\mathbf{x}) = \mathbf{0}$ . These two equations say that  $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$  is a *stationary point for the Lagrangian*, and it is a system of  $n + m$  (possibly nonlinear) equations in  $n + m$  variables.

Let us interpret Theorem 15.1. First of all,  $T(\mathbf{x}^*)$  can be interpreted as a linear subspace consisting of the “first order feasible directions” at  $\mathbf{x}^*$ , i.e. search

directions we can choose which do not violate the constraints (so that  $h_i(\mathbf{x}^* + \mathbf{h}) = 0$  whenever  $h_i(\mathbf{x}^*) = 0$ ,  $i \leq m$ ). To see this, note that  $\nabla h_i(\mathbf{x}^*) \cdot \mathbf{h}$  is what is called the directional derivative of  $h_i$  in the direction  $\mathbf{h}$ . This quantity measures the change of  $h_i$  in direction  $\mathbf{h}$ , and if this is zero,  $h_i$  remains zero when we move in direction  $\mathbf{h}$ , so that the constraints are kept. Actually, if each  $h_i$  is linear, then  $T(\mathbf{x}^*)$  consists of those  $\mathbf{h}$  such that  $\mathbf{x}^* + \mathbf{h}$  is also feasible, i.e.,  $h_i(\mathbf{x}^* + \mathbf{h}) = 0$  for each  $i \leq m$ . Thus, Equation (15.3) says that in a local minimum  $\mathbf{x}^*$  the gradient  $\nabla f(\mathbf{x}^*)$  is orthogonal to the subspace  $T(\mathbf{x}^*)$  of the first order feasible variations. This is reasonable since otherwise there would be a feasible direction in which  $f$  would decrease. In Figure 15.1 we have plotted a curve where two constraints are fulfilled. In Figure 15.2 we have then shown an interpretation of Theorem 15.1. Note that this necessary optimality condition corresponds to the condition  $\nabla f(\mathbf{x}^*) = \mathbf{0}$  in the unconstrained case. The second condition (15.4) is a similar generalization of the second order condition in Theorem 14.1 (saying that  $\nabla^2 f(\mathbf{x}^*)$  is positive semidefinite).

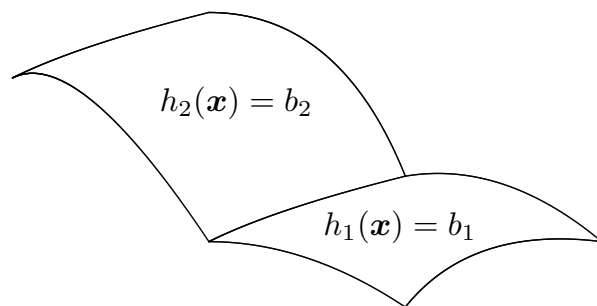


Figure 15.1: The two surfaces  $h_1(\mathbf{x}) = b_1$  og  $h_2(\mathbf{x}) = b_2$  intersect each other in a curve. Along this curve the constraints are fulfilled.

It is possible to prove the theorem by eliminating variables based on the equations and thereby reducing the problem to an unconstrained one. Another proof, which we shall present below is based on the *penalty approach*. This approach is also interesting as it leads to algorithms for actually solving the problem.

*Proof.* (Theorem 15.1) For  $k = 1, 2, \dots$  consider the modified objective function

$$F^k(\mathbf{x}) = f(\mathbf{x}) + (k/2)\|\mathbf{H}(\mathbf{x})\|^2 + (\alpha/2)\|\mathbf{x} - \mathbf{x}^*\|^2$$

where  $\mathbf{x}^*$  is the local minimum under consideration, and  $\alpha$  is a positive constant. The second term is a penalty term for violating the constraints and the last term is there for proof technical reasons. As  $\mathbf{x}^*$  is a local minimum there is an  $\epsilon > 0$  such that  $f(\mathbf{x}^*) \leq f(\mathbf{x})$  for all  $\mathbf{x} \in \bar{B}(\mathbf{x}^*; \epsilon)$ . Choose now an optimal solution  $\mathbf{x}^k$

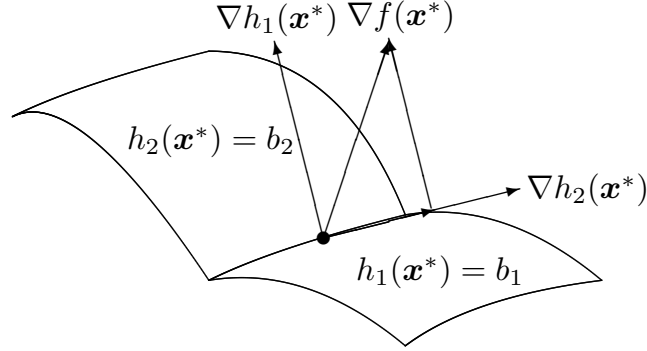


Figure 15.2:  $\nabla f(\mathbf{x}^*)$  as a linear combination of  $\nabla h_1(\mathbf{x}^*)$  and  $\nabla h_2(\mathbf{x}^*)$ .

of the problem  $\min\{F^k(\mathbf{x}) : \mathbf{x} \in \bar{B}(\mathbf{x}^*; \epsilon)\}$ ; the existence here follows from the extreme value theorem ( $F^k$  is continuous and the ball is compact). For every  $k$

$$F^k(\mathbf{x}^k) = f(\mathbf{x}^k) + (k/2)\|\mathbf{H}(\mathbf{x}^k)\|^2 + (\alpha/2)\|\mathbf{x}^k - \mathbf{x}^*\|^2 \leq F^k(\mathbf{x}^*) = f(\mathbf{x}^*).$$

By letting  $k \rightarrow \infty$  in this inequality we conclude that  $\lim_{k \rightarrow \infty} \|\mathbf{H}(\mathbf{x}^k)\| = 0$ . So every limit point  $\bar{\mathbf{x}}$  of the sequence  $\{\mathbf{x}^k\}$  satisfies  $\mathbf{H}(\bar{\mathbf{x}}) = \mathbf{0}$ . The inequality above also implies (by dropping a term on the left-hand side) that  $f(\mathbf{x}^k) + (\alpha/2)\|\mathbf{x}^k - \mathbf{x}^*\|^2 \leq f(\mathbf{x}^*)$  for all  $k$ , so by passing to the limit we get

$$f(\bar{\mathbf{x}}) + (\alpha/2)\|\bar{\mathbf{x}} - \mathbf{x}^*\|^2 \leq f(\mathbf{x}^*) \leq f(\bar{\mathbf{x}})$$

where the last inequality follows from the facts that  $\bar{\mathbf{x}} \in \bar{B}(\mathbf{x}^*; \epsilon)$  and  $\mathbf{H}(\bar{\mathbf{x}}) = \mathbf{0}$ . Clearly, this gives  $\bar{\mathbf{x}} = \mathbf{x}^*$ . We have therefore shown that the sequence  $\{\mathbf{x}^k\}$  converges to the local minimum  $\mathbf{x}^*$ . Since  $\mathbf{x}^*$  is the center of the ball  $\bar{B}(\mathbf{x}^*; \epsilon)$ , the points  $\mathbf{x}^k$  lie in the interior of  $S$  for suitably large  $k$ . The conclusion is then that  $\mathbf{x}^k$  is the **unconstrained** minimum of  $F^k$  when  $k$  is sufficiently large. We may therefore apply Theorem 14.1 so  $\nabla F^k(\mathbf{x}^k) = \mathbf{0}$ . Note first that the Jacobi matrix of  $(k/2)\|\mathbf{H}(\mathbf{x})\|^2$  is the row vector  $k\mathbf{H}(\mathbf{x}^k)^T \mathbf{H}'(\mathbf{x}^k)$ , so that the gradient is  $k\mathbf{H}'(\mathbf{x}^k)^T \mathbf{H}(\mathbf{x}^k)$ . We now obtain

$$\mathbf{0} = \nabla F^k(\mathbf{x}^k) = \nabla f(\mathbf{x}^k) + k\mathbf{H}'(\mathbf{x}^k)^T \mathbf{H}(\mathbf{x}^k) + \alpha(\mathbf{x}^k - \mathbf{x}^*). \quad (15.5)$$

For suitably large  $k$  the matrix  $\mathbf{H}'(\mathbf{x}^k)\mathbf{H}'(\mathbf{x}^k)^T$  is invertible (as the rows of  $\mathbf{H}'(\mathbf{x}^k)$  are linearly independent due to  $\text{rank}(\mathbf{H}'(\mathbf{x}^*)) = m$  and a continuity argument). Multiply equation (15.5) by  $(\mathbf{H}'(\mathbf{x}^k)\mathbf{H}'(\mathbf{x}^k)^T)^{-1}\mathbf{H}'(\mathbf{x}^k)$  on the left to obtain

$$k\mathbf{H}(\mathbf{x}^k) = -(\mathbf{H}'(\mathbf{x}^k)\mathbf{H}'(\mathbf{x}^k)^T)^{-1}\mathbf{H}'(\mathbf{x}^k)(\nabla f(\mathbf{x}^k) + \alpha(\mathbf{x}^k - \mathbf{x}^*)).$$

Letting  $k \rightarrow \infty$  we see that the sequence  $\{k\mathbf{H}(\mathbf{x}^k)\}$  is convergent and its limit point  $\boldsymbol{\lambda}^*$  is given by

$$\boldsymbol{\lambda}^* = -(\mathbf{H}'(\mathbf{x}^*)\mathbf{H}'(\mathbf{x}^*)^T)^{-1}\mathbf{H}'(\mathbf{x}^*)\nabla f(\mathbf{x}^*).$$

Finally, by passing to the limit in (15.5) we get

$$\mathbf{0} = \nabla f(\mathbf{x}^*) + \mathbf{H}'(\mathbf{x}^*)^T \boldsymbol{\lambda}^*$$

This proves the first part of the theorem; we omit proving the second part which may be found in [2].  $\square$

The first order necessary condition (15.3) along with the constraints  $\mathbf{H}(\mathbf{x}) = \mathbf{0}$  is a system of  $n + m$  equations in the  $n + m$  variables  $x_1, x_2, \dots, x_n$  and  $\lambda_1, \lambda_2, \dots, \lambda_m$ . One may use e.g. Newton's method for solving these equations and find a candidate for an optimal solution. But usually there are better numerical methods for solving the optimization (15.1), as we shall see soon.

Necessary optimality conditions are used for finding a candidate solution for being optimal. In order to verify optimality we need *sufficient* optimality conditions.

**Theorem 15.2.** *Lagrange, necessary condition.*

Assume that  $f$  and  $\mathbf{H}$  are twice continuously differentiable functions. Moreover, let  $\mathbf{x}^*$  be a point satisfying the first order necessary optimality condition (15.3) and the following condition

$$\mathbf{y}^T \nabla^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*) \mathbf{y} > 0 \quad \text{for all } \mathbf{y} \neq \mathbf{0} \text{ with } \mathbf{H}'(\mathbf{x}^*) \mathbf{y} = 0 \quad (15.6)$$

where  $\nabla^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*)$  is the Hessian of the Lagrangian function with second order partial derivatives with respect to  $\mathbf{x}$ . Then  $\mathbf{x}^*$  is a (strict) local minimum of  $f$  subject to  $\mathbf{H}(\mathbf{x}) = \mathbf{0}$ .

This theorem may be proved (see [2] for details) by considering the *augmented Lagrangian function*

$$L_c(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{H}(\mathbf{x}) + (c/2) \|\mathbf{H}(\mathbf{x})\|^2 \quad (15.7)$$

where  $c$  is a positive scalar. This is in fact the Lagrangian function in the modified problem

$$\text{minimize } f(\mathbf{x}) + (c/2) \|\mathbf{H}(\mathbf{x})\|^2 \quad \text{subject to } \mathbf{H}(\mathbf{x}) = \mathbf{0} \quad (15.8)$$

and this problem must have the same local minima as the problem of minimizing  $f(\mathbf{x})$  subject to  $\mathbf{H}(\mathbf{x}) = \mathbf{0}$ . The objective function in (15.8) contains the *penalty term*  $(c/2) \|\mathbf{H}(\mathbf{x})\|^2$  which may be interpreted as a penalty (increased function

value) for violating the constraint  $\mathbf{H}(\mathbf{x}) = \mathbf{0}$ . In connection with the proof of Theorem 15.2 based on the augmented Lagrangian one also obtains the following interesting and useful fact:

if  $\mathbf{x}^*$  and  $\boldsymbol{\lambda}^*$  satisfy the sufficient conditions in Theorem 15.2 then there exists a positive  $\bar{c}$  such that for all  $c \geq \bar{c}$  the point  $\mathbf{x}^*$  is also a local minimum of the augmented Lagrangian  $L_c(\cdot, \boldsymbol{\lambda}^*)$ .

Thus, the original constrained problem has been converted to an unconstrained one involving the augmented Lagrangian. And, as we know, unconstrained problems are easier to solve (solve the equations saying that the gradient is equal to zero).

## 15.2 Inequality constraints and KKT

We now consider the general nonlinear optimization problem where there are both equality and inequality constraints. The problem is then

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) \\ & \text{subject to} && \\ & && h_i(\mathbf{x}) = 0 \quad (i \leq m) \\ & && g_j(\mathbf{x}) \leq 0 \quad (j \leq r) \end{aligned} \tag{15.9}$$

We assume, as usual, that all these functions are continuously differentiable real-valued functions defined on  $\mathbb{R}^n$ . In short form we write the constraints as  $\mathbf{H}(\mathbf{x}) = \mathbf{0}$  and  $\mathbf{G}(\mathbf{x}) \leq \mathbf{0}$  where we let  $\mathbf{H} = (h_1, h_2, \dots, h_m)$  and  $\mathbf{G} = (g_1, g_2, \dots, g_r)$ .

A main difficulty in problems with inequality constraints is to determine which of the inequalities that are active in an optimal solution. If we knew the active inequalities, we would essentially have a problem with only equality constraints,  $\mathbf{H}(\mathbf{x}) = \mathbf{0}$  plus the active equalities, i.e., a problem of the form discussed in the previous section. For very small problems (solvable by hand-calculation) a direct method is to consider all possible choices of active inequalities and solve the corresponding equality-constrained problem by looking at the Lagrangian function.

Interestingly, one may also transform the problem (15.9) into the following equality-constrained problem

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) \\ & \text{subject to} && \\ & && h_i(\mathbf{x}) = 0 \quad (i \leq m) \\ & && g_j(\mathbf{x}) + z_j^2 = 0 \quad (j \leq r). \end{aligned} \tag{15.10}$$

We have introduced extra variables  $z_j$ , one for each inequality. The square of these variables represent slack in each of the original inequalities. Note that there is no sign constraint on  $z_j$ . Clearly, the problems (15.9) and (15.10) are equivalent. This transformation can also be useful computationally. Moreover, it is useful theoretically as one may apply the optimality conditions from the previous section to problem (15.10) to derive the theorem below (see [2]).

We now present a main result in nonlinear optimization. It gives optimality conditions for this problem, and these conditions are called the *Karush-Kuhn-Tucker conditions*, or simply the *KKT conditions*. In order to present the KKT conditions we introduce the *Lagrangian* function  $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r \rightarrow \mathbb{R}$  given by

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \sum_{i=1}^m \lambda_i h_i(\mathbf{x}) + \sum_{j=1}^r \mu_j g_j(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{H}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{G}(\mathbf{x}). \quad (15.11)$$

The gradient of  $L$  with respect to  $\mathbf{x}$  is given by

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \nabla f(\mathbf{x}) + \sum_{i=1}^m \lambda_i \nabla h_i(\mathbf{x}) + \sum_{j=1}^r \mu_j \nabla g_j(\mathbf{x}).$$

The Hessian matrix of  $L$  at  $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$  containing second order partial derivatives of  $L$  with respect to  $\mathbf{x}$  will be denoted by  $\nabla_{\mathbf{x}\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ . Finally, the indices of the active inequalities at  $\mathbf{x}$  is denoted by  $A(\mathbf{x})$ , so  $A(\mathbf{x}) = \{j \leq r : g_j(\mathbf{x}) = 0\}$ . A point  $\mathbf{x}$  is called *regular* if  $\{\nabla h_1(\mathbf{x}), \dots, \nabla h_m(\mathbf{x})\} \cup \{\nabla g_i(\mathbf{x}) : i \in A(\mathbf{x})\}$  is linearly independent.

In the following theorem the first part contains necessary conditions while the second part contains sufficient conditions for optimality.

**Theorem 15.3.** *KKT.*

Consider problem (15.9) with the usual differentiability assumptions.

1. Let  $\mathbf{x}^*$  be a local minimum of this problem and assume that  $\mathbf{x}^*$  is a regular point. Then there are unique Lagrange multiplier vectors  $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \dots, \lambda_m^*)$  and  $\boldsymbol{\mu}^* = (\mu_1^*, \mu_2^*, \dots, \mu_r^*)$  such that

$$\begin{aligned} \nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \mathbf{0} \\ \mu_j^* &\geq 0 && (j \leq r) \\ \mu_j^* &= 0 && (j \notin A(\mathbf{x}^*)). \end{aligned} \quad (15.12)$$

If  $f$ ,  $g$  and  $h$  are twice continuously differentiable, then the following also holds

$$\mathbf{y}^T \nabla_{\mathbf{x}\mathbf{x}}^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \mathbf{y} \geq 0 \quad (15.13)$$

for all  $\mathbf{y}$  with  $\nabla h_i(\mathbf{x}^*)^T \mathbf{y} = 0$  ( $i \leq m$ ) and  $\nabla g_j(\mathbf{x}^*)^T \mathbf{y} = 0$  ( $j \in A(\mathbf{x}^*)$ ).

2. Assume that  $\mathbf{x}^*$ ,  $\boldsymbol{\lambda}^*$  and  $\boldsymbol{\mu}^*$  are such that  $\mathbf{x}^*$  is a feasible point and (15.12) holds. Assume, moreover, that (15.13) holds with **strict** inequality for each  $\mathbf{y}$ . Then  $\mathbf{x}^*$  is a (strict) local minimum in problem (15.9).

*Proof.* We shall derive this result from Theorem 15.1.

1. By assumption  $\mathbf{x}^*$  is a local minimum of problem (15.9), and  $\mathbf{x}^*$  is a regular point. Consider the *constrained* problem



$$\begin{aligned}
& \text{minimize} && f(\mathbf{x}) \\
& \text{subject to} && \\
& && h_i(\mathbf{x}) = 0 \quad (i \leq m) \\
& && g_j(\mathbf{x}) = 0 \quad (j \in A(\mathbf{x}^*))
\end{aligned} \tag{15.14}$$

which is obtained by removing all inactive constraints in  $\mathbf{x}^*$ . Then  $\mathbf{x}^*$  must be a local minimum in (15.14); otherwise there would be a point  $\mathbf{x}'$  in the neighborhood of  $\mathbf{x}^*$  which is feasible in (15.14) and satisfying  $f(\mathbf{x}') < f(\mathbf{x}^*)$ . By choosing  $\mathbf{x}'$  sufficiently near  $\mathbf{x}^*$  we would get  $g_j(\mathbf{x}') < 0$  for all  $j \notin A(\mathbf{x}^*)$ , contradicting that  $\mathbf{x}^*$  is a local minimum in (15.9). Therefore we may apply Theorem 15.1 to problem (15.14) and by regularity of  $\mathbf{x}^*$  there must be unique Lagrange multiplier vectors  $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \dots, \lambda_m^*)$  and  $\mu_j^*$  ( $j \in A(\mathbf{x}^*)$ ) such that

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^m \lambda_i^* \nabla h_i(\mathbf{x}^*) + \sum_{j \in A(\mathbf{x}^*)} \mu_j^* \nabla g_j(\mathbf{x}^*) = \mathbf{0}$$

By defining  $\mu_j = 0$  for  $j \notin A(\mathbf{x}^*)$  we get (15.12), except for the nonnegativity of  $\mu$ .

The remaining part of the theorem may be proved, after some work, by studying the equality-constrained reformulation (15.10) of (15.9) and applying Theorem 15.1 to (15.10). The details may be found in [2].  $\square$

The KKT conditions have an interesting geometrical interpretation. They say that  $-\nabla f(\mathbf{x}^*)$  may be written as linear combination of the gradients of the  $h_i$ 's plus a nonnegative linear combination of the gradients of the  $g_j$ 's that are active at  $\mathbf{x}^*$ .

### Example 15.1: A simple optimization problem

Let us consider the following optimization problem:

$$\min\{x_1 : x_2 \geq 0, 1 - (x_1 - 1)^2 - x_2^2 \geq 0\}.$$

Here there are two inequality constraints:

$$\begin{aligned}
g_1(x_1, x_2) &= -x_2 \leq 0 \\
g_2(x_1, x_2) &= (x_1 - 1)^2 + x_2^2 - 1 \leq 0.
\end{aligned}$$

If we compute the gradients we see that the KKT conditions take the form

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \mu_1 \begin{pmatrix} 0 \\ -1 \end{pmatrix} + \mu_2 \begin{pmatrix} 2(x_1 - 1) \\ 2x_2 \end{pmatrix} = \mathbf{0},$$

where the two last terms on the left hand side only are included if the corresponding inequalities are active. It is clear that we find no solutions if no inequalities are active. If only the first inequality is active we find no solution either. If only the second inequality is active we get the equations

$$\begin{aligned}(x_1 - 1)^2 + x_2^2 &= 1 \\ 1 + 2\mu_2(x_1 - 1) &= 0 \\ 2\mu_2x_2 &= 0.\end{aligned}$$

From the last equation we see that either  $x_2 = 0$  or  $\mu_2 = 0$ . But here  $x_2 > 0$  since only the second inequality is active, so that  $\mu_2 = 0$ .  $\mu_2 = 0$  is in conflict with the second equation, however. Finally, let us consider the case where both equalities are active. This occurs only in the points  $(0, 0)$  and  $(2, 0)$ . These two points give the gradients  $\nabla g_2 = (\mp 2, 0)$ , so that the gradient equation can be written as

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \mu_1 \begin{pmatrix} 0 \\ -1 \end{pmatrix} + \mu_2 \begin{pmatrix} \mp 2 \\ 0 \end{pmatrix} = \mathbf{0},$$

These give  $\mu_1 = 0$  and  $\mu_1 = \pm 1/2$ . Since we require  $\mu_2 \geq 0$ , the only candidate we obtain is  $(0, 0)$ .

Finally we should comment on any points which are not regular. If only the first inequality is active it is impossible to have that  $\nabla g_1 = \mathbf{0}$ . If only the second inequality is active it is impossible to have  $\nabla g_2 = \mathbf{0}$  since this would require  $x_1 = 1$ ,  $x_2 = 0$  which contradicts an active point. If both inequalities are active, we saw that  $(0, 0)$  and  $(2, 0)$  are the only possible points. This gave the gradients  $(0, -1)$  and  $(\mp 2, 0)$ , which clearly are linearly independent. We therefore have that all points are regular.

We remark that the assumption that  $\mathbf{x}^*$  is a regular point may be too restrictive in some situations, for instance there may be more than  $n$  active inequalities in  $\mathbf{x}^*$ . There exist several other weaker assumptions that assure the existence of Lagrangian multipliers (and similar necessary conditions).

In the proof of Theorem 15.3 we did not prove the nonnegativity of  $\mu$ . To show this is actually quite hard, but let us comment on the main lines. We first need the concept of a tangent vector.

**Definition 15.4.** *Tangent vector.*

Let  $C \subseteq \mathbb{R}^n$  and let  $\mathbf{x} \in C$ . A vector  $\mathbf{d} \in \mathbb{R}^n$  is called a *tangent* (vector) to  $C$  at  $\mathbf{x}$  if there is a sequence  $\{\mathbf{x}^k\}$  in  $C$  and a sequence  $\{\alpha_k\}$  in  $\mathbb{R}_+$  such that

$$\lim_{k \rightarrow \infty} (\mathbf{x}^k - \mathbf{x})/\alpha_k = \mathbf{d}.$$

The set of tangent vectors at  $\mathbf{x}$  is denoted by  $T_C(\mathbf{x})$ .

$T_C(\mathbf{x})$  always contains the zero vector and it is a cone, meaning that it contains each positive multiple of its vectors. We now restrict to the problem (15.9) and let  $C$  be the set of feasible solutions (those  $\mathbf{x}$  satisfying all the equality and inequality constraints). One first shows that (see [32])  $\mathbf{x}^*$  satisfies  $\nabla f(\mathbf{x}^*)^T \mathbf{d} \geq 0$  for all  $\mathbf{d} \in T_C(\mathbf{x}^*)$ . After this, the following concept is needed.

**Definition 15.5.** *Linearized feasible directions.*

A linearized feasible direction at  $\mathbf{x} \in C$  is a vector  $\mathbf{d}$  such that

$$\begin{aligned} \mathbf{d} \cdot \nabla h_i(\mathbf{x}) &= 0 & (i \leq m) \\ \mathbf{d} \cdot \nabla g_j(\mathbf{x}) &\leq 0 & (j \in A(\mathbf{x})) \end{aligned}$$

(since  $\mathbf{H}'(\mathbf{x})$  is the matrix with rows  $\nabla h_i(\mathbf{x})$ , the first condition is the same as  $\mathbf{H}'(\mathbf{x})\mathbf{d} = 0$ . Similarly, when all constraints are active the second condition is the same as  $\mathbf{G}'(\mathbf{x})\mathbf{d} \leq 0$ ). We denote by  $LF_C(\mathbf{x})$  the set of all linearized feasible directions at  $\mathbf{x}$ .

So, if we move from  $\mathbf{x}$  along a linearized feasible direction with a suitably small step, then the new point is feasible if we only care about the *linearized* constraints at  $\mathbf{x}^*$  (the first order Taylor approximations) of each  $h_i$  and each  $g_j$  for active constraints at  $\mathbf{x}^*$ , i.e., those inequality constraints that hold with equality. With this notation we have the following lemma. The proof may be found in [32] and it involves the implicit function theorem from multivariate calculus [26].

**Lemma 15.6.** *Tangent cone and feasible directions.*

Let  $\mathbf{x}^* \in C$ . Then  $T_C(\mathbf{x}^*) \subseteq LF_C(\mathbf{x}^*)$ . If  $\mathbf{x}^*$  is a regular point, then  $T_C(\mathbf{x}^*) = LF_C(\mathbf{x}^*)$ .

Putting these things together, when  $\mathbf{x}^*$  is regular,  $\nabla f(\mathbf{x}^*)^T \mathbf{d} \geq 0$  for all  $\mathbf{d} \in LF_C(\mathbf{x}^*)$ . Now we need a lemma called *Farkas' lemma*.

**Lemma 15.7.** *Farkas lemma.*

If  $B$  and  $C$  are matrices with  $n$  rows, and  $K$  is the cone defined by  $K = \{B\mathbf{y} + C\mathbf{w}, \text{ with } \mathbf{y} \geq \mathbf{0}\}$ , then exactly one of the following two alternatives are true:

1.  $\mathbf{g} \in K$
2. There exists a  $\mathbf{d} \in \mathbb{R}^n$  so that  $\mathbf{g}^T \mathbf{d} < 0$ ,  $B^T \mathbf{d} \geq 0$ , and  $C^T \mathbf{d} = 0$ .

If we apply this lemma with  $\mathbf{g} = \nabla f(\mathbf{x}^*)$ ,  $B = -\mathbf{G}'(\mathbf{x}^*)^T$ , and  $C = -\mathbf{H}'(\mathbf{x}^*)^T$ , the conditions  $B^T \mathbf{d} \geq 0$ , and  $C^T \mathbf{d} = 0$  simply says that  $\mathbf{d} \in LF_C(\mathbf{x}^*) = T_C(\mathbf{x}^*)$ . But for all such  $\mathbf{d}$  we have proved that  $\mathbf{g}^T \mathbf{d} = \nabla f(\mathbf{x}^*)^T \mathbf{d} \geq 0$ , so that point 2 of Farkas lemma does not hold for  $\mathbf{g} = \nabla f(\mathbf{x}^*)$ . We conclude that  $\mathbf{g} = \nabla f(\mathbf{x}^*) \in K$ , so that we can find  $\mathbf{y} \geq 0$  and  $\mathbf{w}$  so that

$$\mathbf{g} = \nabla f(\mathbf{x}^*) = -\mathbf{H}'(\mathbf{x}^*)^T \mathbf{w} - \mathbf{G}'(\mathbf{x}^*)^T \mathbf{y} = B\mathbf{y} + C\mathbf{w}.$$

But this states exactly what we want to prove, namely that  $\nabla f(\mathbf{x}^*) + \mathbf{H}'(\mathbf{x}^*)^T \mathbf{w} + \mathbf{G}'(\mathbf{x}^*)^T \mathbf{y} = \mathbf{0}$ , and that  $\mathbf{w}$  contains the Lagrange multipliers  $\lambda_i$ , and  $\mathbf{y}$  contains the  $\mu_i$ , which must be non-negative. For a more thorough discussion of these matters, see e.g. [32, 2].

In the remaining part of this section we discuss some examples; the main tool is to establish the KKT conditions.

**Example 15.2: a one-variable problem**

Consider the one-variable problem: minimize  $f(x)$  subject to  $x \geq 0$ , where  $f : \mathbb{R} \rightarrow \mathbb{R}$  is a differentiable convex function. We here let  $g_1(x) = -x$  and  $m = 0$ . The KKT conditions then become: there is a number  $\mu$  such that  $f'(x) - \mu = 0$ ,  $\mu \geq 0$  and  $\mu = 0$  if  $x > 0$ . This is one of the (rare) occasions where we can eliminate the Lagrangian variable  $\mu$  via the equation  $\mu = f'(x)$ . So the optimality conditions are:  $x \geq 0$  (feasibility),  $f'(x) \geq 0$ , and  $f'(x) = 0$  if  $x > 0$  ( $x$  is an interior point of the domain so the derivative must be zero), and if  $x = 0$  we must have  $f'(0) \geq 0$ .

**Example 15.3: a multi-variable problem**

More generally, consider the problem to minimize  $f(\mathbf{x})$  subject to  $\mathbf{x} \geq \mathbf{0}$ , where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ . So here  $C = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \geq \mathbf{0}\}$  is the nonnegative orthant. We have that  $g_i(x) = -x_i$ , so that  $\nabla g_i = -\mathbf{e}_i$ . The KKT conditions say that  $-\nabla f(\mathbf{x}^*)$  is a nonnegative combination of  $-\mathbf{e}_i$  for  $i$  so that  $x_i = 0$ . In other words,  $\nabla f(\mathbf{x}^*)$  is a nonnegative combination of  $\mathbf{e}_i$  for  $i$  so that  $x_i = 0$ . This means that

$$\begin{aligned} \partial f(\mathbf{x}^*)/\partial x_i &= 0 && \text{for all } i \leq n \text{ with } \mathbf{x}_i^* > 0, \quad \text{and} \\ \partial f(\mathbf{x}^*)/\partial x_i &\geq 0 && \text{for all } i \leq n \text{ with } \mathbf{x}_i^* = 0. \end{aligned}$$

If we interpret this for  $n = 3$  we get the following cases:

- No active constraints: This means that  $x, y, z > 0$ . The KKT-conditions say that all partial derivatives are 0, so that  $\nabla f(\mathbf{x}^*) = \mathbf{0}$ . This is reasonable, since these points are internal points.
- One active constraint, such as  $x = 0, y, z > 0$ . The KKT-conditions say that  $\partial f(\mathbf{x}^*)/\partial y = \partial f(\mathbf{x}^*)/\partial z = 0$ , so that  $\nabla f(\mathbf{x}^*)$  points in the positive direction of  $\mathbf{e}_1$ , as shown in Figure 15.3(a).
- Two active constraints, such as  $x = y = 0, z > 0$ . The KKT-conditions say that  $\partial f(\mathbf{x}^*)/\partial z = 0$ , so that  $\nabla f(\mathbf{x}^*)$  lies in the cone spanned by  $\mathbf{e}_1, \mathbf{e}_2$ , i.e.  $\nabla f(\mathbf{x}^*)$  lies in the first quadrant of the  $xy$ -plane, as shown in Figure 15.3(b).
- Three active constraints: This means that  $x = y = z = 0$ . The KKT conditions say that  $\nabla f(\mathbf{x}^*)$  is in the cone spanned by  $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ , as shown in Figure 15.3(c).

In all cases  $\nabla f(\mathbf{x}^*)$  points into a cone spanned by gradients corresponding to the active inequalities (in general, by a cone we mean the set of all linear combinations of a set of vectors, with positive coefficients). Note that for the third case above, we are used to finding minimum values from before: if we restrict  $f$  to values where  $x = y = 0$ , we have a one-dimensional problem where we want to minimize  $g(z) = f(x, y, z)$ , which is equivalent to finding  $z$  so that  $g'(z) = \partial f(\mathbf{x}^*)/\partial z = 0$ , as stated by the KKT-conditions.

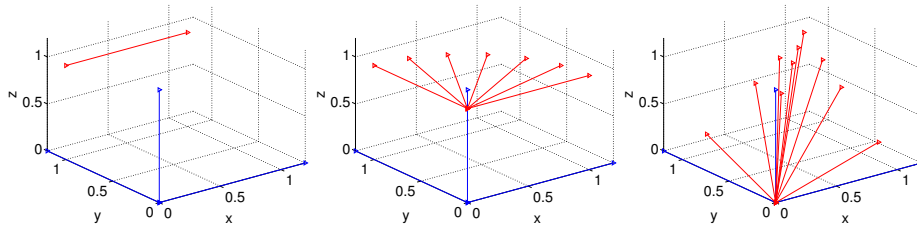


Figure 15.3: The different possibilities (one, two, and three active constraints) for  $\nabla f$  in a minimum of  $f$ , under the constraints  $\mathbf{x} \geq \mathbf{0}$ .

### Example 15.4: Quadratic optimization problem with linear equality constraints

Consider the problem

$$\begin{aligned} & \text{minimize} && (1/2) \mathbf{x}^T D \mathbf{x} - \mathbf{q}^T \mathbf{x} \\ & \text{subject to} && \\ & && A \mathbf{x} = \mathbf{b} \end{aligned}$$

where  $D$  is positive semidefinite and  $A \in \mathbb{R}^{m \times n}$ ,  $\mathbf{b} \in \mathbb{R}^m$ . This is a special case of (15.16) where  $f(\mathbf{x}) = (1/2) \mathbf{x}^T D \mathbf{x} - \mathbf{q}^T \mathbf{x}$ . Then  $\nabla f(\mathbf{x}) = D \mathbf{x} - \mathbf{q}$  (see Exercise 11.9 in Chapter 11). Thus, the KKT conditions are: there is some  $\boldsymbol{\lambda} \in \mathbb{R}^m$  such that  $D \mathbf{x} - \mathbf{q} + A^T \boldsymbol{\lambda} = \mathbf{0}$ . In addition, the vector  $\mathbf{x}$  is feasible so we have  $A \mathbf{x} = \mathbf{b}$ . Thus, solving the quadratic optimization problem amounts to solving the linear system of equations

$$D \mathbf{x} + A^T \boldsymbol{\lambda} = \mathbf{q}, \quad A \mathbf{x} = \mathbf{b}$$

which may be written as

$$\begin{bmatrix} D & A^T \\ A & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{q} \\ \mathbf{b} \end{bmatrix}. \quad (15.15)$$

Under the additional assumption that  $D$  is positive definite and  $A$  has full row rank, one can show that the coefficient matrix in (15.15) is invertible so this system has a unique solution  $\mathbf{x}, \boldsymbol{\lambda}$ . Thus, for this problem, we may write down an explicit solution (in terms of the inverse of the block matrix). Numerically, one finds  $\mathbf{x}$  (and the Lagrangian multiplier  $\boldsymbol{\lambda}$ ) by solving the linear system (15.15) by e.g. Gaussian elimination or some faster (direct or iterative) method.

### Example 15.5: Extension

Consider an extension of the previous example by allowing linear inequality constraints as well:

$$\begin{aligned} & \text{minimize} && (1/2) \mathbf{x}^T D \mathbf{x} - \mathbf{q}^T \mathbf{x} \\ & \text{subject to} && \\ & && A \mathbf{x} = \mathbf{b} \\ & && \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Here  $D$ ,  $A$  and  $\mathbf{b}$  are as above. Then  $\nabla f(\mathbf{x}) = D\mathbf{x} - \mathbf{q}$  and  $\nabla g_k(\mathbf{x}) = -\mathbf{e}_k$ . Thus, the KKT conditions for this problem are: there are  $\boldsymbol{\lambda} \in \mathbb{R}^m$  and  $\boldsymbol{\mu} \in \mathbb{R}^n$  such that  $D\mathbf{x} - \mathbf{q} + A^T \boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{0}$ ,  $\boldsymbol{\mu} \geq \mathbf{0}$  and  $\mu_k = 0$  if  $x_k > 0$  ( $k \leq n$ ). We eliminate  $\boldsymbol{\mu}$  from the first equation and obtain the equivalent condition: there is a  $\boldsymbol{\lambda} \in \mathbb{R}^m$  such that  $D\mathbf{x} + A^T \boldsymbol{\lambda} \geq \mathbf{q}$  and  $(D\mathbf{x} + A^T \boldsymbol{\lambda} - \mathbf{q})_k \cdot x_k = 0$  ( $k \leq n$ ). In addition, we have  $A\mathbf{x} = \mathbf{b}$ ,  $\mathbf{x} \geq \mathbf{0}$ . This problem may be solved numerically, for instance, by a so-called active set method, see [27].

### Example 15.6: Linear optimization

Linear optimization is a problem of the form

$$\text{minimize } \mathbf{c}^T \mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}$$

This is a special case of the convex programming problem (15.16) where  $g_j(\mathbf{x}) = -x_j$  ( $j \leq n$ ). Here  $\nabla f(\mathbf{x}) = \mathbf{c}$  and  $\nabla g_k(\mathbf{x}) = -\mathbf{e}_k$ . Let  $\mathbf{x}$  be a feasible solution. The KKT conditions state that there are vectors  $\boldsymbol{\lambda} \in \mathbb{R}^m$  and  $\boldsymbol{\mu} \in \mathbb{R}^n$  such that  $\mathbf{c} + A^T \boldsymbol{\lambda} - \boldsymbol{\mu} = \mathbf{0}$ ,  $\boldsymbol{\mu} \geq \mathbf{0}$  and  $\mu_k = 0$  if  $x_k > 0$  ( $k \leq n$ ). Here we eliminate  $\boldsymbol{\mu}$  and obtain the equivalent set of KKT conditions: there is a vector  $\boldsymbol{\lambda} \in \mathbb{R}^m$  such that  $\mathbf{c} + A^T \boldsymbol{\lambda} \geq \mathbf{0}$ ,  $(\mathbf{c} + A^T \boldsymbol{\lambda})_k \cdot x_k = 0$  ( $k \leq n$ ). These conditions are the familiar optimality conditions in linear optimization theory. The vector  $\boldsymbol{\lambda}$  is feasible in the so-called dual problem and complementary slack holds. We do not go into details on this here, but refer to the course INF-MAT3370 *Linear optimization* where these matters are treated in detail.

## 15.3 Convex optimization

A *convex optimization* problem is to minimize a convex function  $f$  over a convex set  $C$  in  $\mathbb{R}^n$ . These problems are especially attractive, both from a theoretic and algorithmic perspective.

First, let us consider some general results.

**Theorem 15.8.** *Optimizing convex functions.*

Let  $f : C \rightarrow \mathbb{R}$  be a convex function defined on a convex set  $C \subseteq \mathbb{R}^n$ .

1. Then every local minimum of  $f$  over  $C$  is also a global minimum.
2. If  $f$  is continuous and  $C$  is closed, then the set of local (and therefore global) minimum points of  $f$  over  $C$  is a closed convex set.
3. Assume, furthermore, that  $f : C \rightarrow \mathbb{R}$  is differentiable and  $C$  is open. Let  $\mathbf{x}^* \in C$ . Then  $\mathbf{x}^* \in C$  is a local (global) minimum if and only if  $\nabla f(\mathbf{x}^*) = \mathbf{0}$ .

*Proof.* 1.) The proof of property 1 is exactly as the proof of the first part of Theorem 14.4, except that we work with local and global minimum of  $f$  over  $C$ .

2.) Assume the set  $C^*$  of minimum points is nonempty and let  $\alpha = \min_{\mathbf{x} \in C} f(\mathbf{x})$ . Then  $C^* = \{\mathbf{x} \in C : f(\mathbf{x}) \leq \alpha\}$  is a convex set, see Proposition 12.5. Moreover, this set is closed as  $f$  is continuous.

3.) This follows directly from Theorem 12.10.  $\square$

Next, we consider a quite general convex optimization problem which is of the form (15.9):

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) \\ & \text{subject to} && \\ & && A\mathbf{x} = \mathbf{b} \\ & && g_j(\mathbf{x}) \leq 0 \quad (j \leq r) \end{aligned} \tag{15.16}$$

where all the functions  $f$  and  $g_j$  are differentiable convex functions, and  $A \in \mathbb{R}^{m \times n}$  and  $\mathbf{b} \in \mathbb{R}^m$ . Let  $C$  denote the feasible set of problem (15.16). Then  $C$  is a convex set, see Proposition 12.5. A special case of (15.16) is linear optimization.

An important concept in convex optimization is *duality*. To briefly explain this introduce again the Lagrangian function  $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}_+^r \rightarrow \mathbb{R}$  given by

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T(A\mathbf{x} - \mathbf{b}) + \boldsymbol{\nu}^T \mathbf{G}(\mathbf{x}) \quad (\mathbf{x} \in \mathbb{R}^n, \boldsymbol{\lambda} \in \mathbb{R}^m, \boldsymbol{\nu} \in \mathbb{R}_+^r)$$

Remark: we use the variable name  $\boldsymbol{\nu}$  here in stead of the  $\boldsymbol{\mu}$  used before because of another parameter  $\boldsymbol{\mu}$  to be used soon. Note that we require  $\boldsymbol{\nu} \geq \mathbf{0}$ .

Define the new function  $g : \mathbb{R}^m \times \mathbb{R}_+^r \rightarrow \bar{\mathbb{R}}$  by

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

Note that this infimum may sometimes be equal to  $-\infty$  (meaning that the function  $\mathbf{x} \rightarrow L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$  is unbounded below). The function  $g$  is the pointwise infimum of a family of affine functions in  $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ , one function for each  $\mathbf{x}$ , and this implies that  $g$  is a concave function. We are interested in  $g$  due to the following fact, which is easy to prove. It is usually referred to as *weak duality*.

**Lemma 15.9.** *Weak duality.*

Let  $\mathbf{x}$  be feasible in problem (15.16) and let  $\boldsymbol{\lambda} \in \mathbb{R}^m, \boldsymbol{\nu} \in \mathbb{R}^r$  where  $\boldsymbol{\nu} \geq \mathbf{0}$ . Then

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f(\mathbf{x}).$$

*Proof.* For  $\boldsymbol{\lambda} \in \mathbb{R}^m, \boldsymbol{\nu} \in \mathbb{R}^r$  with  $\boldsymbol{\nu} \geq \mathbf{0}$  and  $\mathbf{x}$  feasible in problem (15.16) we have

$$\begin{aligned} g(\boldsymbol{\lambda}, \boldsymbol{\nu}) & \leq L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \\ & = f(\mathbf{x}) + \boldsymbol{\lambda}^T(A\mathbf{x} - \mathbf{b}) + \boldsymbol{\nu}^T \mathbf{G}(\mathbf{x}) \\ & \leq f(\mathbf{x}) \end{aligned}$$

as  $A\mathbf{x} = \mathbf{b}, \boldsymbol{\nu} \geq \mathbf{0}$  and  $\mathbf{G}(\mathbf{x}) \leq \mathbf{0}$ .  $\square$

Thus,  $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$  provides a lower bound on the optimal value in (15.16). It is natural to look for a best possible such lower bound and this is precisely the so-called *dual problem*, which is

$$\begin{aligned} & \text{maximize} && g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \\ & \text{subject to} && \\ & && \boldsymbol{\nu} \geq \mathbf{0}. \end{aligned} \tag{15.17}$$

Actually, in this dual problem, we may further restrict the attention to those  $(\boldsymbol{\lambda}, \boldsymbol{\nu})$  for which  $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$  is finite.  $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$  is also called the *dual objective function*.

The original problem (15.16) will be called the *primal problem*. It follows from Lemma 15.9 that

$$g^* \leq f^*$$

where  $f^*$  denotes the optimal value in the primal problem and  $g^*$  the optimal value in the dual problem. If  $g^* < f^*$ , we say that there is a *duality gap*. Note that the derivation above, and weak duality, holds for *arbitrary* functions  $f$  and  $g_j$  ( $j \leq r$ ). The concavity of  $g$  also holds generally.

The dual problem is useful when the dual objective function  $g$  may be computed efficiently, either analytically or numerically. Duality provides a powerful method for proving that a solution is optimal or, possibly, near-optimal. If we have a feasible  $\boldsymbol{x}$  in (15.16) and we have found a dual solution  $(\boldsymbol{\lambda}, \boldsymbol{\nu})$  with  $\boldsymbol{\nu} \geq \mathbf{0}$  such that

$$f(\boldsymbol{x}) = g(\boldsymbol{\lambda}, \boldsymbol{\nu}) + \epsilon$$

for some  $\epsilon$  (which then has to be nonnegative), then we can conclude that  $\boldsymbol{x}$  is “nearly optimal”, it is not possible to improve  $f$  by more than  $\epsilon$ . Such a point  $\boldsymbol{x}$  is sometimes called  *$\epsilon$ -optimal*, where the case  $\epsilon = 0$  means optimal.

So, how good is this duality approach? For *convex problems* it is often perfect as the next theorem says. We omit most of the proof, see [19, 2, 49]). For non-convex problems one should expect a duality gap. Recall that  $\mathbf{G}'(\boldsymbol{x})$  denotes the Jacobi matrix of  $\mathbf{G} = (g_1, g_2, \dots, g_r)$  at  $\boldsymbol{x}$ .

**Theorem 15.10.** *Convex optimization.*

Consider convex optimization problem (15.16) and assume this problem has a feasible point satisfying

$$g_j(\boldsymbol{x}') < 0 \quad (j \leq r).$$

Then  $f^* = g^*$ , so there is no duality gap. Moreover,  $\boldsymbol{x}$  is a (local and global) minimum in (15.16) if and only if there are  $\boldsymbol{\lambda} \in \mathbb{R}^m$  and  $\boldsymbol{\nu} \in \mathbb{R}^r$  with  $\boldsymbol{\nu} \geq \mathbf{0}$  and

$$\nabla f(\boldsymbol{x}) + A^T \boldsymbol{\lambda} + \mathbf{G}'(\boldsymbol{x})^T \boldsymbol{\nu} = \mathbf{0}$$

and

$$\nu_j g_j(\boldsymbol{x}) = 0 \quad (j \leq r).$$



*Proof.* We only prove the second part (see the references above). So assume that  $f^* = g^*$  and the infimum and supremum are attained in the primal and dual problems, respectively. Let  $\mathbf{x}$  be a feasible point in the primal problem. Then  $\mathbf{x}$  is a minimum in the primal problem if and only if there are  $\boldsymbol{\lambda} \in \mathbb{R}^m$  and  $\boldsymbol{\nu} \in \mathbb{R}^r$  such that all the inequalities in the proof of Lemma 15.9 hold with equality. This means that  $g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$  and  $\boldsymbol{\nu}^T \mathbf{G}(\mathbf{x}) = 0$ . But  $L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$  is convex in  $\mathbf{x}$  so it is minimized by  $\mathbf{x}$  if and only if its gradient is the zero vector, i.e.,  $\nabla f(\mathbf{x}) + \boldsymbol{\lambda}^T A + \mathbf{G}'(\mathbf{x})^T \boldsymbol{\nu} = \mathbf{0}$ . This leads to the desired characterization.  $\square$

The assumption stated in the theorem, that  $g_j(\mathbf{x}') < 0$  for each  $j$ , is called the *weak Slater condition*.

### Example 15.7: Comparing the primal and the dual problem

Consider the convex optimization problem where we want to minimize the function  $f(x) = x^2 + 1$  subject to the inequality constraint  $g(x) = (x - 3)^2 - 1 \leq 0$ . From Figure 15.4(a) it is quite clear that the minimum is attained for  $x = 2$ , and is  $f(2) = 5$ . Since both the constraint and the objective function are convex, and since here the weak Slater condition holds, Theorem 15.10 guarantees that the dual problem has the same solution as the primal problem. Let us verify this by considering the dual problem as well. The Lagrangian function is given by

$$L(x, \nu) = f(x) + \nu g(x) = x^2 + 1 + \nu((x - 3)^2 - 1).$$

It is easy to see that this function attains its minimum for  $x = \frac{3\nu}{1+\nu}$ . This means that the dual objective function is given by

$$g(\nu) = L\left(\frac{3\nu}{1+\nu}, \nu\right) = \left(\frac{3\nu}{1+\nu}\right)^2 + 1 + \nu\left(\left(\frac{3\nu}{1+\nu} - 3\right)^2 - 1\right).$$

This is shown in Figure 15.4(b).

It is quite clear from this figure that the maximum is 5, which we already found by solving the primal problem. To prove this requires some more work, by setting the derivative of the dual objective function to zero. Therefore, the primal and the dual problem are two very different problems, where we in practice choose the one which is simplest to solve.

#### 15.3.1 A useful theorem on convex optimization

Finally, we mention a theorem on convex optimization which is used in several applications.

**Theorem 15.11.** *Characterization of a convex function.*

Let  $f : C \rightarrow \mathbb{R}$  be a convex function defined on a convex set  $C \subseteq \mathbb{R}^n$ , and  $\mathbf{x}^* \in C$ . Then  $\mathbf{x}^*$  is a (local and therefore global) minimum of  $f$  over  $C$  if and only if

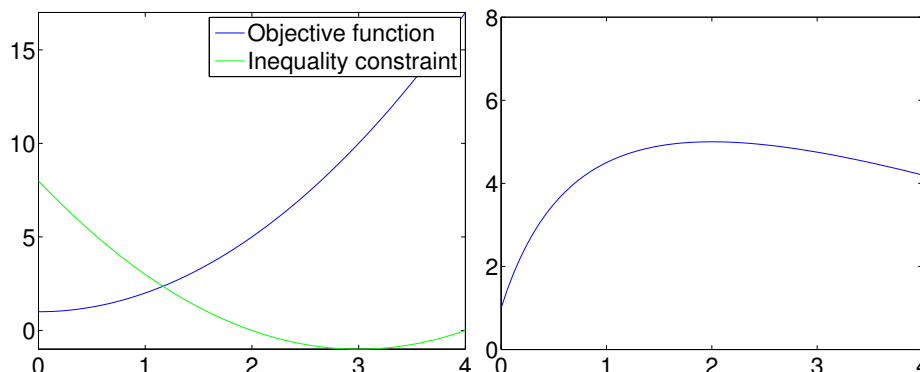


Figure 15.4: The objective function and the dual objective function of Example 15.7.

$$\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) \geq 0 \quad \text{for all } \mathbf{x} \in C. \quad (15.18)$$

*Proof.* Assume first that  $\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) < 0$  for some  $\mathbf{x} \in C$ . Consider the function  $g(\epsilon) = f(\mathbf{x}^* + \epsilon(\mathbf{x} - \mathbf{x}^*))$  and apply the first order Taylor theorem to this function. Thus, for every  $\epsilon > 0$  there exists an  $t \in [0, 1]$  with

$$f(\mathbf{x}^* + \epsilon(\mathbf{x} - \mathbf{x}^*)) = f(\mathbf{x}^*) + \epsilon \nabla f(\mathbf{x}^* + t\epsilon(\mathbf{x} - \mathbf{x}^*))^T(\mathbf{x} - \mathbf{x}^*).$$

Since  $\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) < 0$  and the gradient function is continuous (our standard assumption!) we have for sufficiently small  $\epsilon > 0$  that  $\nabla f(\mathbf{x}^* + t\epsilon(\mathbf{x} - \mathbf{x}^*))^T(\mathbf{x} - \mathbf{x}^*) < 0$ . This implies that  $f(\mathbf{x}^* + \epsilon(\mathbf{x} - \mathbf{x}^*)) < f(\mathbf{x}^*)$ . But, as  $C$  is convex, the point  $\mathbf{x}^* + \epsilon(\mathbf{x} - \mathbf{x}^*) = \epsilon\mathbf{x} + (1 - \epsilon)\mathbf{x}^*$  also lies in  $C$  and so we conclude that  $\mathbf{x}^*$  is not a local minimum. This proves that (15.18) is necessary for  $\mathbf{x}^*$  to be a local minimum of  $f$  over  $C$ .

Next, assume that (15.18) holds. Using Theorem 12.10 we then get

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) \geq f(\mathbf{x}^*) \quad \text{for every } \mathbf{x} \in C$$

so  $\mathbf{x}^*$  is a (global) minimum.  $\square$

### Exercise 15.8: Find min

In the plane consider a rectangle  $R$  with sides of length  $x$  and  $y$  and with perimeter equal to  $\alpha$  (so  $2x + 2y = \alpha$ ). Determine  $x$  and  $y$  so that the area of  $R$  is largest possible.

### Exercise 15.9: Find min

Consider the optimization problem

minimize  $f(x_1, x_2)$  subject to  $(x_1, x_2) \in C$

where  $C = \{(x_1, x_2) \in \mathbb{R}^2 : x_1, x_2 \geq 0, 4x_1 + x_2 \geq 8, 2x_1 + 3x_2 \leq 12\}$ . Draw the feasible set  $C$  in the plane. Find the set of optimal solutions in each of the cases given below.

- a)  $f(x_1, x_2) = 1$ .
- b)  $f(x_1, x_2) = x_1$ .
- c)  $f(x_1, x_2) = 3x_1 + x_2$ .
- d)  $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2$ .
- e)  $f(x_1, x_2) = (x_1 - 10)^2 + (x_2 - 8)^2$ .

### Exercise 15.10: Find min

Solve

$$\max\{x_1 x_2 \cdots x_n : \sum_{j=1}^n x_j = 1, x_j \geq 0\}.$$

### Exercise 15.11: Find min

Let  $S = \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\| = 1\}$  be the unit circle in the plane. Let  $\mathbf{a} \in \mathbb{R}^2$  be a given point. Formulate the problem of finding a nearest point in  $S$  to  $\mathbf{a}$  as a nonlinear optimization problem. How can you solve this problem directly using a geometrical argument?

### Exercise 15.12: Find min

Let  $S$  be the unit circle from the previous exercise. Let  $\mathbf{a}_1, \mathbf{a}_2$  be two given points in the plane. Let  $f(\mathbf{x}) = \sum_{i=1}^2 \|\mathbf{x} - \mathbf{a}_i\|^2$ . Formulate this as an optimization problem and find its Lagrangian function  $L$ . Find the stationary points of  $L$ , and use this to solve the optimization problem.

### Exercise 15.13: Find min

Solve

$$\text{minimize } x_1 + x_2 \text{ subject to } x_1^2 + x_2^2 = 1.$$

using the Lagrangian, see Theorem 15.1. Next, solve the problem by eliminating  $x_2$  (using the constraint).

**Exercise 15.14: Find min**

Let  $g(x_1, x_2) = 3x_1^2 + 10x_1x_2 + 3x_2^2 - 2$ . Solve

$$\min\{\|(x_1, x_2)\| : g(x_1, x_2) = 0\}.$$

**Exercise 15.15: Find min**

Same question as in previous exercise, but with  $g(x_1, x_2) = 5x_1^2 - 4x_1x_2 + 4x_2^2 - 6$ .

**Exercise 15.16: Find min**

Let  $f$  be a two times differentiable function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ . Consider the optimization problem

$$\text{minimize } f(\mathbf{x}) \text{ subject to } x_1 + x_2 + \cdots + x_n = 1.$$

Characterize the stationary points (find the equation they satisfy).

**Exercise 15.17: Find min**

Consider the previous exercise. Explain how to convert this into an unconstrained problem by eliminating  $x_n$ .

**Exercise 15.18: Find min**

Let  $A$  be a real symmetric  $n \times n$  matrix. Consider the optimization problem

$$\max \left\{ \frac{1}{2} \mathbf{x}^T A \mathbf{x} : \|\mathbf{x}\| = 1 \right\}$$

Rewrite the constraint as  $\|\mathbf{x}\| - 1 = 0$  and show that an optimal solution of this problem must be an eigenvector of  $A$ . What can you say about the Lagrangian multiplier?

**Exercise 15.19: Find min**

Solve

$$\min\{(1/2)(x_1^2 + x_2^2 + x_3^2) : x_1 + x_2 + x_3 \leq -6\}.$$

**Exercise 15.20: Find min**

Solve

$$\min\{(x_1 - 3)^2 + (x_2 - 5)^2 + x_1x_2 : 0 \leq x_1, x_2 \leq 1\}.$$

**Exercise 15.21: Find min**

Solve

$$\min\{x_1 + x_2 : x_1^2 + x_2^2 \leq 2\}.$$

**Exercise 15.22: Find min**

Write down the KKT conditions for the portfolio optimization problem of Section 11.2.

**Exercise 15.23: Find min**

Write down the KKT conditions for the optimization problem

$$\min\{f(x_1, x_2, \dots, x_n) : x_j \geq 0 \ (j \leq n), \sum_{j=1}^n x_j \leq 1\}$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is a differentiable function.**Exercise 15.24: Find min**

Consider the following optimization problem

$$\min\left\{\left(x_1 - \frac{3}{2}\right)^2 + x_2^2 : x_1 + x_2 \leq 1, x_1 - x_2 \leq 1, -x_1 + x_2 \leq 1, -x_1 - x_2 \leq 1\right\}.$$

a) Draw the region which we minimize over, and find the minimum of  $f(\mathbf{x}) = \left(x_1 - \frac{3}{2}\right)^2 + x_2^2$  by a direct geometric argument.

b) Write down the KKT conditions for this problem. From a., decide which two conditions  $g_1$  and  $g_2$  are active at the minimum, and verify that you can find  $\mu_1 \geq 0, \mu_2 \geq 0$  so that  $\nabla f + \mu_1 \nabla g_1 + \mu_2 \nabla g_2 = \mathbf{0}$  (as the KKT conditions guarantee in a minimum) (it is not the meaning here that you should go through all possibilities for active inequalities, only those you see must be fulfilled from a.).

**Exercise 15.25: Find min**

Consider the following optimization problem

$$\min\{-x_1 x_2 : x_1^2 + x_2^2 \leq 1\}$$

Write down the KKT conditions for this problem, and find the minimum.

## Chapter 16

# Constrained optimization - methods

In this final chapter we present numerical methods for solving nonlinear optimization problems. This is a huge area, so we can here only give a small taste of it! The algorithms we present are known good methods.

### 16.1 Equality constraints

We here consider the nonlinear optimization problem with linear equality constraints

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) \\ & \text{subject to} && \\ & && A\mathbf{x} = \mathbf{b} \end{aligned} \tag{16.1}$$

Newton's method may be applied to this problem. The method is very similar to the unconstrained case, but with two modifications. First, the initial point  $\mathbf{x}_0$  must be chosen so that it is feasible, i.e.,  $A\mathbf{x}_0 = \mathbf{b}$ . Next, the search direction  $\mathbf{d}$  must be such that the new iterate is feasible as well. This means that  $A\mathbf{d} = \mathbf{0}$ , so the search direction lies in the nullspace of  $A$ .

The second order Taylor approximation of  $f$  at an iterate  $\mathbf{x}_k$  is

$$T_f^1(\mathbf{x}_k; \mathbf{x}_k + \mathbf{h}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{h} + (1/2)\mathbf{h}^T \nabla^2 f(\mathbf{x}_k) \mathbf{h}$$

and we want to minimize this under the constraint  $A\mathbf{x}_{k+1} = A(\mathbf{x}_k + \mathbf{h}) = A\mathbf{x}_k = \mathbf{b}$ , i.e.

$$A\mathbf{h} = \mathbf{0} \tag{16.2}$$

Since the gradient of  $T_f^1$  w.r.t.  $\mathbf{h}$  is  $\nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k)\mathbf{h}$ , setting the gradient of the Lagrangian w.r.t.  $\mathbf{h}$  equal to zero gives

$$\nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k)\mathbf{h} + A^T\boldsymbol{\lambda} = \mathbf{0}, \quad (16.3)$$

where  $\boldsymbol{\lambda}$  is the Lagrange multiplier. Equations (16.2)-(16.3) together give

$$\begin{bmatrix} \nabla^2 f(\mathbf{x}_k) & A^T \\ A & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{h} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\nabla f(\mathbf{x}_k) \\ \mathbf{0} \end{bmatrix}.$$

The Newton step is only defined when the coefficient matrix in the KKT problem is invertible. In that case, the problem has a unique solution  $(\mathbf{h}, \boldsymbol{\lambda})$  and we define  $\mathbf{d}_{Nt} = \mathbf{h}$  and call this the *Newton step*. Newton's method for solving Equation (16.1) can now be extended from the previous code.

TODO: Explain why the stop criterion  $\eta^2/2 < \epsilon$  with  $\eta := \mathbf{d}_{Nt}^T \nabla^2 f(\mathbf{x}) \mathbf{d}_{Nt}$  is used

This leads to an algorithm for Newton's method for linear equality constrained optimization which is very similar to the function `newtonbacktrack` from Exercise 14.12. We do not state a formal convergence theorem for this method, but it behaves very much like Newton's method for unconstrained optimization. Actually, it can be seen that the method just described corresponds to eliminating variables based on the equations  $A\mathbf{x} = \mathbf{b}$  and using the unconstrained Newton method for the resulting (smaller) problem. So as soon as the solution is "sufficiently near" an optimal solution, the convergence rate is quadratic, so extremely few iterations are needed in this final stage.

## 16.2 Inequality constraints

We here briefly discuss an algorithm for inequality constrained nonlinear optimization problems. The presentation is mainly based on [3, 32]. We restrict the attention to convex optimization problems, but many of the ideas are used for nonconvex problems as well.

The method we present is an *interior-point method*, more precisely, an *interior-point barrier method*. This is an iterative method which produces a sequence of points lying in the relative interior of the feasible set. The barrier idea is to approximate the problem by a simpler one in which constraints are replaced by a penalty term. The purpose of this penalty term is to give large objective function values to points near the (relative) boundary of the feasible set, which effectively becomes a barrier against leaving the feasible set.

Consider again the convex optimization problem

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && \\ & && A\mathbf{x} = \mathbf{b} \\ & && g_j(\mathbf{x}) \leq 0 \quad (j \leq r) \end{aligned} \quad (16.4)$$

where  $A$  is an  $m \times n$  matrix and  $\mathbf{b} \in \mathbb{R}^m$ . The feasible set here is  $F = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, g_j(\mathbf{x}) \leq 0 \ (j \leq r)\}$ . We assume that the weak Slater condition holds, and therefore by Theorem 15.10 the KKT conditions for problem (16.4) are

$$\begin{aligned} A\mathbf{x} &= \mathbf{b}, & g_j(\mathbf{x}) &\leq 0 \quad (j \leq r) \\ \boldsymbol{\nu} &\geq \mathbf{0}, & \nabla f(\mathbf{x}) + A^T \boldsymbol{\lambda} + \mathbf{G}'(\mathbf{x})^T \boldsymbol{\nu} &= \mathbf{0} \\ & & \nu_j g_j(\mathbf{x}) &= 0 \quad (j \leq r). \end{aligned} \quad (16.5)$$

So,  $\mathbf{x}$  is a minimum in (16.4) if and only if there are  $\boldsymbol{\lambda} \in \mathbb{R}^m$  and  $\boldsymbol{\nu} \in \mathbb{R}^r$  such that (16.5) holds.

Let us state an algorithm for Newton's method for linear equality constrained optimization with inequality constraints. Before we do this there is one final problem we need to address: The  $\alpha$  we get from backtracking line search may be so that  $\mathbf{x} + \alpha \mathbf{d}_{Nt}$  do not satisfy the inequality constraints (in the exercises you will be asked to verify that this is the case for a certain function). The problem comes from that the iterates  $\mathbf{x}_k + \beta^m s \mathbf{d}_k$  from Armijo's rule do not necessarily satisfy the inequality constraints. However, we can choose  $m$  large enough so that all succeeding iterates satisfy these constraints. We can modify the function `newtonbacktrack` from Exercise 14.12 to a function `newtonbacktrackg1g2` in an obvious way so that, in addition to applying Armijos rule, we also choose a step size so small that the inequality constraints are satisfied:

```
function [x,numit]=newtonbacktrackg1g2LEC(f,df,d2f,A,b,x0,g1,g2)
    epsilon=10^(-3);
    x=x0;
    maxit=100;
    for numit=1:maxit
        matr=[d2f(x) A'; A zeros(size(A,1))];
        vect=[-df(x); zeros(size(A,1),1)];
        solvedvals=matr\vect;
        d=solvedvals(1:size(A,2));
        eta=d'*d2f(x)*d;
        if eta^2/2<epsilon
            break;
        end
        % Armijos rule with two inequalities
        beta=0.2; s=0.5; sigma=10^(-3);
        m=0;
        while (f(x)-f(x+beta^m*s*d) < -sigma *beta^m*s *(df(x))'*d) || (g1(x+beta^m*s*d)>0) || (g2(x
            m=m+1;
        end
        alpha = beta^m*s;
        x=x+alpha*d;
    end
```

Here `g1` and `g2` are function handles which represent the inequality constraints. The new function works only in the case when there are exactly two inequality constraints.

The interior-point barrier method is based on an approximation of problem (16.4) by the *barrier problem*

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) + \mu\phi(\mathbf{x}) \\ &\text{subject to} && A\mathbf{x} = \mathbf{b} \end{aligned} \quad (16.6)$$

where



$$\phi(\mathbf{x}) = -\sum_{j=1}^r \ln(-g_j(\mathbf{x}))$$

and  $\mu > 0$  is a parameter (in  $\mathbb{R}$ ). The function  $\phi$  is called the (*logarithmic barrier function*) and its domain is the relative interior of the feasible set

$$F^\circ = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, g_j(\mathbf{x}) < 0 \ (j \leq r)\}.$$

The same set  $F^\circ$  is the feasible set of the barrier problem. The key properties of the barrier function are:

1.  $\phi$  is twice differentiable and

$$\nabla\phi(\mathbf{x}) = \sum_{j=1}^r \frac{1}{(-g_j(\mathbf{x}))} \nabla g_j(\mathbf{x}) \quad (16.7)$$

$$\nabla^2\phi(\mathbf{x}) = \sum_{j=1}^r \frac{1}{g_j^2(\mathbf{x})} \nabla g_j(\mathbf{x}) \nabla g_j(\mathbf{x})^T + \sum_{j=1}^r \frac{1}{(-g_j(\mathbf{x}))} \nabla^2 g_j(\mathbf{x}) \quad (16.8)$$

2.  $\phi$  is convex. For this it is enough to show that  $\nabla^2\phi$  is positive semidefinite at all points, which can be shown from Equation (16.8) as follows:

$$\begin{aligned} \mathbf{h}^T \nabla^2\phi(\mathbf{x}) \mathbf{h} &= \sum_{j=1}^r \left( \frac{1}{g_j^2(\mathbf{x})} \mathbf{h}^T \nabla g_j(\mathbf{x}) \nabla g_j(\mathbf{x})^T \mathbf{h} + \frac{1}{(-g_j(\mathbf{x}))} \mathbf{h}^T \nabla^2 g_j(\mathbf{x}) \mathbf{h} \right) \\ &= \sum_{j=1}^r \left( \frac{1}{g_j^2(\mathbf{x})} \|\nabla g_j(\mathbf{x})^T \mathbf{h}\|^2 + \frac{1}{(-g_j(\mathbf{x}))} \mathbf{h}^T \nabla^2 g_j(\mathbf{x}) \mathbf{h} \right) \geq 0 \end{aligned}$$

since  $\frac{1}{(-g_j(\mathbf{x}))} > 0$  and  $\mathbf{h}^T \nabla^2 g_j(\mathbf{x}) \mathbf{h} \geq 0$  (since all  $g_j$  are convex,  $\nabla^2 g_j(\mathbf{x})$  is positive semidefinite).

3. If  $\{\mathbf{x}_k\}$  is a sequence in  $F^\circ$  such that  $g_j(\mathbf{x}_k) \rightarrow 0$  for some  $j \leq r$ , then  $\phi(\mathbf{x}_k) \rightarrow \infty$ . This is the barrier property.

The idea here is that for points  $\mathbf{x}$  near the boundary of  $F$  the value of  $\phi(\mathbf{x})$  is very large. So, an iterative method which moves around in the interior  $F^\circ$  of  $F$  will typically avoid points near the boundary as the logarithmic penalty term makes the function value  $f(\mathbf{x}) + \mu\phi(\mathbf{x})$  very large.

The interior point method consists in solving the barrier problem, using Newton's method, for a sequence  $\{\mu_k\}$  of (positive) barrier parameters; these are called the *outer iterations*. The solution  $\mathbf{x}_k$  found for  $\mu = \mu_k$  is used as the starting point in Newton's method in the next outer iteration where  $\mu = \mu_{k+1}$ . The sequence  $\{\mu_k\}$  is chosen such that  $\mu_k \rightarrow 0$ . When  $\mu$  is very small, the barrier function approximates the "ideal" penalty function  $\eta(x)$  which is zero in  $F$  and  $-\infty$  when one of the inequalities  $g_j(\mathbf{x}) \leq 0$  is violated.

A natural question is why one bothers to solve the barrier problems for more than one single  $\mu$ , typically a very small value. The reason is that it would be

hard to find a good starting point for Newton's method in that case; the Hessian matrix of  $\mu\phi$  is typically ill-conditioned for small  $\mu$ .

Assume now that the barrier problem has a *unique* optimal solution  $\mathbf{x}(\mu)$ ; this is true under reasonable assumptions that we shall return to. The point  $\mathbf{x}(\mu)$  is called a *central point*. Assume also that Newton's method may be applied to solve the barrier problem. The set of points  $\mathbf{x}(\mu)$  for  $\mu > 0$  is called the *central path*; it is a path (or curve) as we know it from multivariate calculus. In order to investigate the central path we prefer to work with the equivalent problem<sup>1</sup>. to (16.6) obtained by multiplying the objection function by  $1/\mu$ , so

$$\begin{aligned} & \text{minimize} && (1/\mu)f(\mathbf{x}) + \phi(\mathbf{x}) \\ & \text{subject to} && \\ & && A\mathbf{x} = \mathbf{b}. \end{aligned} \tag{16.9}$$

A central point  $\mathbf{x}(\mu)$  is characterized by

$$\begin{aligned} A\mathbf{x}(\mu) &= \mathbf{b} \\ g_j(\mathbf{x}(\mu)) &< 0 \quad (j \leq r) \end{aligned}$$

and the existence of  $\boldsymbol{\lambda} \in \mathbb{R}^m$  (the Lagrange multiplier vector) such that

$$(1/\mu)\nabla f(\mathbf{x}(\mu)) + \nabla\phi(\mathbf{x}(\mu)) + A^T\boldsymbol{\lambda} = \mathbf{0}$$

i.e.,

$$(1/\mu)\nabla f(\mathbf{x}(\mu)) + \sum_{j=1}^r \frac{1}{(-g_j(\mathbf{x}(\mu)))} \nabla g_j(\mathbf{x}(\mu)) + A^T\boldsymbol{\lambda} = \mathbf{0}. \tag{16.10}$$

A fundamental question is: how far from being optimal is the central point  $\mathbf{x}(\mu)$ ? We now show that duality provides a very elegant way of answering this question.

**Theorem 16.1.** *Distance from minimum.*

For each  $\mu > 0$  the central point  $\mathbf{x}(\mu)$  satisfies

$$f^* \leq f(\mathbf{x}(\mu)) \leq f^* + r\mu.$$

*Proof.* Define  $\boldsymbol{\nu}(\mu) = (\nu_1(\mu), \dots, \nu_r(\mu)) \in \mathbb{R}^r$  and  $\boldsymbol{\lambda}(\mu) \in \mathbb{R}^m$  as Lagrange parameters for the original problem by

$$\begin{aligned} \nu_j(\mu) &= -\mu/g_j(\mathbf{x}(\mu)), & (j \leq r); \\ \boldsymbol{\lambda}(\mu) &= \mu\boldsymbol{\lambda}. \end{aligned} \tag{16.11}$$

where  $\boldsymbol{\lambda}$  and  $\mathbf{x}(\mu)$  satisfy Equation (16.10), i.e. they are Lagrange parameters for the barrier problem. We need to return to the dual problem (of the original problem), defined in Section 15.3. We first claim that the pair  $(\boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu))$

<sup>1</sup>Equivalent here means the same minimum points.

is feasible in the dual problem to (16.4). We thus need to show that  $\boldsymbol{\nu}(\mu)$  is nonnegative. This is immediate: since  $g_j(\mathbf{x}(\mu)) < 0$  and  $\mu > 0$ , we get  $\nu_j(\mu) = -\mu/g_j(\mathbf{x}(\mu)) > 0$  for each  $j$ . We now also want to show that  $\mathbf{x}(\mu)$  satisfies

$$g(\boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)) = L(\mathbf{x}(\mu), \boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)),$$

where  $g$  is the dual objective function. To see this, note first that the Lagrangian function  $L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T(A\mathbf{x} - \mathbf{b}) + \boldsymbol{\nu}^T\mathbf{G}(\mathbf{x})$  is convex in  $\mathbf{x}$  for given  $\boldsymbol{\lambda}$  and  $\boldsymbol{\nu} \geq 0$ . Thus,  $\mathbf{x}$  minimizes this function if and only if  $\nabla_{\mathbf{x}}L = \mathbf{0}$ . Now,

$$\begin{aligned} & \nabla_{\mathbf{x}}L(\mathbf{x}(\mu), \boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)) \\ &= \nabla f(\mathbf{x}(\mu)) + A^T\boldsymbol{\lambda}(\mu) + \sum_{j=1}^r \nu_j(\mu)\nabla g_j(\mathbf{x}(\mu)) \\ &= \nabla f(\mathbf{x}(\mu)) + \mu A^T\boldsymbol{\lambda} + \mu \sum_{j=1}^r \frac{1}{(-g_j(\mathbf{x}(\mu)))} \nabla g_j(\mathbf{x}(\mu)) \\ &= \mu \left( \frac{1}{\mu} \nabla f(\mathbf{x}(\mu)) + A^T\boldsymbol{\lambda} + \sum_{j=1}^r \frac{1}{(-g_j(\mathbf{x}(\mu)))} \nabla g_j(\mathbf{x}(\mu)) \right) = \mathbf{0}, \end{aligned}$$

by (16.10) and the definition of the dual variables (16.11). This shows that  $g(\boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)) = L(\mathbf{x}(\mu), \boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu))$ .

By weak duality and Lemma 15.9, we now obtain

$$\begin{aligned} f^* &\geq g(\boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)) \\ &= L(\mathbf{x}(\mu), \boldsymbol{\lambda}(\mu), \boldsymbol{\nu}(\mu)) \\ &= f(\mathbf{x}(\mu)) + \boldsymbol{\lambda}(\mu)^T(A\mathbf{x}(\mu) - \mathbf{b}) + \sum_{j=1}^r \nu_j(\mu)g_j(\mathbf{x}(\mu)) \\ &= f(\mathbf{x}(\mu)) - r\mu \end{aligned}$$

which proves the result.  $\square$

This theorem is very useful and shows why letting  $\mu \rightarrow 0$  (more accurately  $\mu \rightarrow 0^+$ ) is a good idea.

**Corollary 16.2.** *Convergence of the central path.*

The central path has the following property

$$\lim_{\mu \rightarrow 0} f(\mathbf{x}(\mu)) = f^*.$$

In particular, if  $f$  is continuous and  $\lim_{\mu \rightarrow 0} \mathbf{x}(\mu) = \mathbf{x}^*$  for some  $\mathbf{x}^*$ , then  $\mathbf{x}^*$  is a global minimum in (16.4).

*Proof.* This follows from Theorem 16.1 by letting  $\mu \rightarrow 0$ . The second part follows from

$$f(\mathbf{x}^*) = f(\lim_{\mu \rightarrow 0} \mathbf{x}(\mu)) = \lim_{\mu \rightarrow 0} f(\mathbf{x}(\mu)) = f^*$$

by the first part and the continuity of  $f$ ; moreover  $\mathbf{x}^*$  must be a feasible point by elementary topology.  $\square$

After these considerations we may now present the interior-point barrier method. The following code uses a tolerance  $\epsilon > 0$  in its stopping criterion, and assumes 2 inequality constraints:

```
function xopt=IPBopt(f,g1,g2,df,dg1,dg2,d2f,d2g1,d2g2,A,b,x0)
    xopt=x0;
    mu=1;
    alpha=0.1;
    r=2;
    epsilon=10^(-3);
    numitouter=0;
    while (r*mu>epsilon)
        [xopt,numit]=newtonbacktrackg1g2LEC(...
            @(x)(f(x)-mu*log(-g1(x))-mu*log(-g2(x))),...
            @(x)(df(x) - mu*dg1(x)/g1(x) - mu*dg2(x)/g2(x)),...
            @(x)(d2f(x) + mu*dg1(x)*dg1(x)'/(g1(x)^2) ...
                + mu*dg2(x)*dg2(x)'/(g2(x)^2) - mu*d2g1(x)/g1(x)...
                - mu*d2g2(x)/g2(x) ),A,b,xopt,g1,g2);
        mu=alpha*mu;
        numitouter=numitouter+1;
        fprintf('Iteration %i:',numitouter);
        fprintf('(%f,%f)\n',xopt,f(xopt));
    end
```

Note that we here have inserted the expressions from Equation (16.7) and Equation (16.8) for the gradient and the Hesse matrix of the barrier function. The input are  $f$ ,  $g_1$ ,  $g_2$ , their gradients and their Hesse matrices, the matrix  $A$ , the vector  $\mathbf{b}$ , and an initial feasible point  $\mathbf{x}_0$ . The function calls `newtonbacktrackg1g2LEC`, and returns the optimal solution  $\mathbf{x}^*$ . It also gives some information on the values of  $f$  during the iterations. The iterations used in Newton's method is called the *inner iterations*. There are different implementation details here that we do not discuss very much. A typical value on  $\alpha$  is 0.1. The choice of the initial  $\mu^0$  can be difficult, if it is chosen too large, one may experience many outer iterations. Another issue is how accurately one solves (16.6). It may be sufficient to find a near-optimal solution here as this saves inner iterations. For this reason the method is also called a *path-following method*; it follows in the neighborhood of the central path.

Finally, it should be mentioned that there exists a variant of the interior-point barrier method which permits an infeasible starting point. For more details on this and various implementation issues one may consult [3] or [32].

### Example 16.1: Numeric test of the internal-point barrier method

Consider the function  $f(x) = x^2 + 1$ ,  $2 \leq x \leq 4$ . Minimizing  $f$  can be considered as the problem of finding a minimum subject to the constraints  $g_1(x) = 2 - x \leq 0$ , and  $g_2(x) = x - 4 \leq 0$ . The barrier problem is to minimize the function

$$f(x) + \mu\phi(x) = x^2 + 1 - \mu \ln(x - 2) - \mu \ln(4 - x).$$

Some of these are drawn in Figure 16.1, where we clearly can see the effect of decreasing  $\mu$  in the barrier function: The function converges to  $f$  pointwise as  $\mu \rightarrow 0^+$ , except at the boundaries  $x = 2$ ,  $x = 4$ .

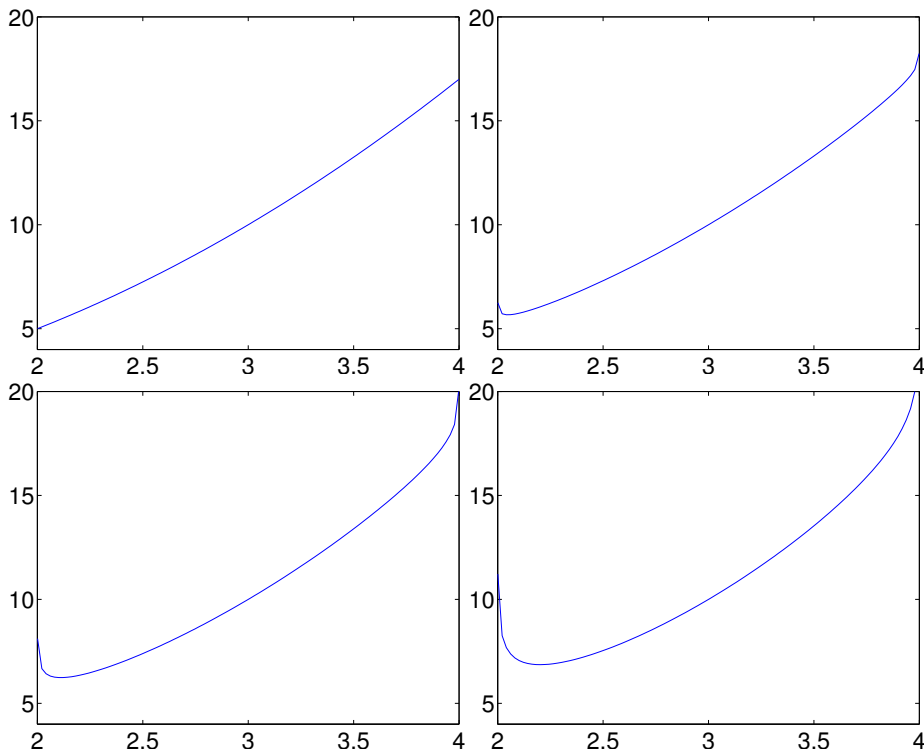


Figure 16.1: The function from Example 16.1 and its barrier functions with  $\mu = 0.2$ ,  $\mu = 0.5$ , and  $\mu = 1$ .

It is easy to see that  $x = 2$  is the minimum of  $f$  under the given constraints, and that  $f(2) = 5$  is the minimum value. There are no equality constraints in this case, so that we can use the barrier method with Newton's method for unconstrained optimization, as this was implemented in Exercise 14.12. We need, however, to make sure also here that the iterates from Armijo's rule satisfy the inequality constraints. In fact, in the exercises you will be asked to verify that,

for the function  $f$  considered here, some of the iterates from Armijo's rule do not satisfy the constraints.

It is straightforward to implement a function `newtonbacktrackg1g2` which implements Newton's method for two inequality constraints and no equality constraints, similarly to how we implemented the function `newtonbacktrackg1g2LEC`. This leads to the following algorithm for the internal point barrier method for the case of no equality constraints, but 2 inequality constraints:

```
function xopt=IPBopt2(f,g1,g2,df,dg1,dg2,d2f,d2g1,d2g2,x0)
    xopt=x0;
    mu=1; alpha=0.1; r=2; epsilon=10^(-3);
    numitouter=0;
    while (r*mu>epsilon)
        [xopt,numit]=newtonbacktrackg1g2(...
            @(x)(f(x)-mu*log(-g1(x))-mu*log(-g2(x))),...
            @(x)(df(x) - mu*dg1(x)/g1(x) - mu*dg2(x)/g2(x)),...
            @(x)(d2f(x) + mu*dg1(x)*dg1(x)'/(g1(x)^2) ...
                + mu*dg2(x)*dg2(x)'/(g2(x)^2) ...
                - mu*d2g1(x)/g1(x) - mu*d2g2(x)/g2(x) ),xopt,g1,g2);
        mu=alpha*mu;
        numitouter=numitouter+1;
        fprintf('Iteration %i:',numitouter);
        fprintf('%f,%f\n',xopt,f(xopt));
    end
```

Note that this function also prints a summary for each of the outer iterations, so that we can see the progress in the barrier method. We can now find the minimum of  $f$  with the following code, where we have substituted functions for  $f$ ,  $g_i$ , their gradients and Hessians.

```
IPBopt2(@(x)(x.^2+1),@(x)(2-x),@(x)(x-4),...
        @(x)(2*x),@(x)(-1),@(x)(1),...
        @(x)(2),@(x)(0),@(x)(0),3)
```

Running this code gives a good approximation to the minimum  $x = 2$  after 4 outer iterations.

### Example 16.2: Analytic test of the internal-point barrier method

Let us consider the problem of finding the minimum of  $x_1^2 + x_2^2$  subject to the constraint  $x_1 + x_2 \geq 2$ . We set  $f(x_1, x_2) = x_1^2 + x_2^2$ , and write the constraint as  $g_1(x_1, x_2) = 2 - x_1 - x_2 \leq 0$ . Here it is not difficult to state the KKT conditions and solve these, so let us do this first. The gradients are  $\nabla f = (2x_1, 2x_2)$ ,  $\nabla g_1 = (-1, -1)$ , so that the KKT conditions take the form

$$(2x_1, 2x_2) + \nu_1(-1, -1) = \mathbf{0}$$

for a  $\nu_1 \geq 0$ , where the last term is included only if  $x_1 + x_2 = 2$  (i.e. when the constraint is active). If the constraint is not active we see that  $x_1 = x_2 = 0$ , which does not satisfy the inequality constraint. If the constraint is active we

see that  $x_1 = x_2 = \nu_1/2$ , so that  $x_1 = x_2 = 1$  and  $\nu_1 = 2 \geq 0$  in order for  $x_1 + x_2 = 2$ . The minimum value is thus  $f(1, 1) = 2$ . It is clear that this must be a minimum: Since  $f$  is bounded below and approaches  $\infty$  when either  $x_1$  or  $x_2$  grows large, it must have a minimum ( $f$  has no global maximum). For this one can also argue that the Hessian of the Lagrangian for the constrained problem becomes positive definit. All points are regular for this problem since  $\nabla g_1 \neq \mathbf{0}$ .

Let us also see if we can come to this same solution by solving the barrier problem. The barrier function is  $\phi(x_1, x_2) = -\ln(x_1 + x_2 - 2)$ , which has gradient  $\nabla\phi = (-1/(x_1 + x_2 - 2), -1/(x_1 + x_2 - 2))$ . We set the gradient of  $f(x_1, x_2) + \mu\phi(x_1, x_2)$  to 0 and get

$$(2x_1, 2x_2) + \mu(-1/(x_1 + x_2 - 2), -1/(x_1 + x_2 - 2)) = \mathbf{0}.$$

From this we see that  $x_1 = x_2$  must fulfill  $2x_1 = \frac{\mu}{2x_1 - 2}$ , so that  $4x_1(x_1 - 1) = \mu$ , so that  $4x_1^2 - 4x_1 - \mu = 0$ . If we solve this problem we find that  $x_1 = \frac{4 \pm \sqrt{16 + 16\mu}}{8} = \frac{1 \pm \sqrt{1 + \mu}}{2}$ . If we choose the negative sign here we find that  $x_1 < 0$ , which does not lie inside the domain of definition for the function we optimize (i.e. points where  $x_1 + x_2 > 2$ ). If we choose the positive sign we find  $x_1 = x_2 = \frac{1 + \sqrt{1 + \mu}}{2}$ . It is clear that, when  $\mu \rightarrow 0$ , this will converge to  $x_1 = x_2 = 1$ , which equals the solution we found when we solved the KKT conditions.

### Exercise 16.3: Solve

Consider problem (16.1) in Section 16.1. Verify that the KKT conditions for this problem are as stated there.

### Exercise 16.4: Solve

Define the function  $f(x, y) = x + y$ . We will attempt to minimize  $f$  under the constraints  $y - x = 1$ , and  $x, y \geq 0$

- Find  $A$ ,  $\mathbf{b}$ , and functions  $g_1, g_2$  so that the problem takes the same form as in Equation (16.4).
- Draw the contours of the barrier function  $f(x, y) + \mu\phi(x, y)$  for  $\mu = 0.1, 0.2, 0.5, 1$ , where  $\phi(x, y) = -\ln(-g_1(x, y)) - \ln(-g_2(x, y))$ .
- Solve the barrier problem analytically using the Lagrange method.
- It is straightforward to find the minimum of  $f$  under the mentioned constraints. State a simple argument for finding this minimum.
- State the KKT conditions for finding the minimum, and solve these.
- Show that the central path converges to the same solution which you found in d. and e..

**Exercise 16.5: Solve**

Use the function `IPBopt` to verify the solution you found in Exercise 16.4. Initially you must compute a feasible starting point  $\mathbf{x}_0$ .

**Exercise 16.6: Solve**

State the KKT conditions for finding the minimum for the constrained problem of Example 16.1, and solve these. Verify that you get the same solution as in Example 16.1.

**Exercise 16.7: Solve**

In the function `IPBopt2`, replace the call to the function `newtonbacktrackg1g2` with a call to the function `newtonbacktrack`, with the obvious modification to the parameters. Verify that the code does not return the expected minimum in this case.

**Exercise 16.8: Solve**

Consider the function  $f(x) = (x - 3)^2$ , with the same constraints  $2 \leq x \leq 4$  as in Example 16.1. Verify in this case that the function `IPBopt2` returns the correct minimum regardless of whether you call `newtonbacktrackg1g2` or `newtonbacktrack`. This shows that, at least in some cases where the minimum is an interior point, the iterates from Newton's method satisfy the inequality constraints as well.

**Exercise 16.9: Solve**

In this exercise we will find the minimum of the function  $f(x, y) = 3x + 2y$  under the constraints  $x + y = 1$  and  $x, y \geq 0$ .

- a) Find a matrix  $A$  and a vector  $\mathbf{b}$  so that the constraint  $x + y = 1$  can be written on the form  $A\mathbf{x} = \mathbf{b}$ .
- b) State the KKT-conditions for this problem, and find the minimum by solving these.
- c) Write down the barrier function  $\phi(x, y) = -\ln(-g_1(x, y)) - \ln(-g_2(x, y))$  for this problem, where  $g_1$  and  $g_2$  represent the two constraints of the problem. Also compute  $\nabla\phi$ .
- d) Solve the barrier problem with parameter  $\mu$ , and denote the solution by  $\mathbf{x}(\mu)$ . Is it the case that the limit  $\lim_{\mu \rightarrow 0} \mathbf{x}(\mu)$  equals the solution you found in b.?



# Appendix A

## Basic Linear Algebra

This book assumes that the student has taken a beginning course in linear algebra at university level. In this appendix we summarize the most important concepts one needs to know from linear algebra. Note that what is listed here should not be considered as a substitute for such a course: It is important for the student to go through a full course in linear algebra, in order to get good intuition for these concepts through extensive exercises. Such exercises are omitted here.

### A.1 Matrices

An  $m \times n$ -matrix is simply a set of  $mn$  numbers, stored in  $m$  rows and  $n$  columns. We write  $a_{kn}$  for the entry in row  $k$  and column  $n$  of the matrix  $A$ . The zero matrix, denoted  $\mathbf{0}$  is the matrix with all zeroes. A square matrix (i.e. where  $m = n$ ) is said to be diagonal if  $a_{kn} = 0$  whenever  $k \neq n$ . The identity matrix, denoted  $I$ , or  $I_n$  to make the dimension of the matrix clear, is the diagonal matrix where the entries on the diagonal are 1, the rest zeroes. If  $A$  is a matrix we will denote the transpose of  $A$  by  $A^T$ . If  $A$  is invertible we denote its inverse by  $A^{-1}$ . We say that a matrix  $A$  is *orthogonal* if  $A^T A = A A^T = I$ . A matrix is called sparse if most of the entries in the matrix are zero.

### A.2 Vector spaces

A set of vectors  $V$  is called a vector space if ... We say that the vectors  $\{\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{n-1}\}$  are *linearly independent* if, whenever  $\sum_{i=0}^{n-1} c_i \mathbf{v}_i = \mathbf{0}$ , we must have that all  $c_i = 0$ . We will say that a set of vectors  $\mathcal{B} = \{\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{n-1}\}$  from  $V$  is a *basis* for  $V$  if the vectors are linearly independent, and span  $V$ .

Subspaces of  $\mathbb{R}^N$ , and function spaces.

### A.3 Inner products and orthogonality

Most vector spaces in this book are inner product spaces. A (real) inner product on a vector space is a binary operation, written as  $(\mathbf{u}, \mathbf{v}) \rightarrow \langle \mathbf{u}, \mathbf{v} \rangle$ , which fulfills the following properties for any vectors  $\mathbf{u}$ ,  $\mathbf{v}$ , and  $\mathbf{w}$ :

- $\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{v}, \mathbf{u} \rangle$
- $\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle + \langle \mathbf{v}, \mathbf{w} \rangle$
- $\langle c\mathbf{u}, \mathbf{v} \rangle = c\langle \mathbf{u}, \mathbf{v} \rangle$  for any scalar  $c$
- $\langle \mathbf{u}, \mathbf{u} \rangle \geq 0$ , and  $\langle \mathbf{u}, \mathbf{u} \rangle = 0$  if and only if  $\mathbf{u} = \mathbf{0}$ .

$\mathbf{u}$  and  $\mathbf{v}$  are said to be *orthogonal* if  $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{0}$ . In this book we have seen two important examples of inner product spaces. First of all the Euclidean inner product, which is defined by

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=0}^{n-1} u_i v_i \quad (\text{A.1})$$

for any  $\mathbf{u}, \mathbf{v}$  in  $\mathbb{R}^n$ . For functions we have seen examples which are variants of the following form:

$$\langle f, g \rangle = \int f(t)g(t)dt. \quad (\text{A.2})$$

Any set of mutually orthogonal elements are also linearly independent. A basis where all basis vectors are mutually orthogonal is called an *orthogonal basis*. If additionally the vectors all have length 1, we say that the basis is *orthonormal*. If  $\mathbf{x}$  is in a vector space with an orthogonal basis  $\mathcal{B} = \{\mathbf{v}_k\}_{k=0}^{n-1}$ , we can express  $\mathbf{x}$  as

$$\frac{\langle \mathbf{x}, \mathbf{v}_0 \rangle}{\langle \mathbf{v}_0, \mathbf{v}_0 \rangle} \mathbf{v}_0 + \frac{\langle \mathbf{x}, \mathbf{v}_1 \rangle}{\langle \mathbf{v}_1, \mathbf{v}_1 \rangle} \mathbf{v}_1 + \cdots + \frac{\langle \mathbf{x}, \mathbf{v}_{n-1} \rangle}{\langle \mathbf{v}_{n-1}, \mathbf{v}_{n-1} \rangle} \mathbf{v}_{n-1}. \quad (\text{A.3})$$

In other words, the weights in linear combinations are easily found when the basis is orthogonal. This is also called the *orthogonal decomposition theorem*.

By the *projection* of a vector  $\mathbf{x}$  onto a subspace  $U$  we mean the vector  $\mathbf{y} = \text{proj}_U \mathbf{x}$  which minimizes the distance  $\|\mathbf{y} - \mathbf{x}\|$ . If  $\mathbf{v}_i$  is an orthogonal basis for  $U$ , we have that  $\text{proj}_U \mathbf{x}$  can be written by Equation (A.3).

### A.4 Coordinates and change of coordinates

If  $\mathcal{B} = \{\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{n-1}\}$  is a basis for a vector space, and  $\mathbf{x} = \sum_{i=0}^{n-1} x_i \mathbf{v}_i$ , we say that  $(x_0, x_1, \dots, x_{n-1})$  is the *coordinate vector* of  $\mathbf{x}$  w.r.t. the basis  $\mathcal{B}$ . We also write  $[\mathbf{x}]_{\mathcal{B}}$  for this coordinate vector.

If  $\mathcal{B}$  and  $\mathcal{C}$  are two different bases for the same vector space, we can write down the two coordinate vectors  $[\mathbf{x}]_{\mathcal{B}}$  and  $[\mathbf{x}]_{\mathcal{C}}$ . A useful operation is to transform the coordinates in  $\mathcal{B}$  to those in  $\mathcal{C}$ , i.e. apply the transformation which sends  $[\mathbf{x}]_{\mathcal{B}}$  to  $[\mathbf{x}]_{\mathcal{C}}$ . This is a linear transformation, and we will denote the  $n \times n$ -matrix of this linear transformation by  $P_{\mathcal{C} \leftarrow \mathcal{B}}$ , and call this the *change of coordinate matrix* from  $\mathcal{B}$  to  $\mathcal{C}$ . In other words, the change of coordinate matrix is defined by requiring that

$$P_{\mathcal{C} \leftarrow \mathcal{B}}[\mathbf{x}]_{\mathcal{B}} = [\mathbf{x}]_{\mathcal{C}}. \quad (\text{A.4})$$

It is straightforward to show that  $P_{\mathcal{C} \leftarrow \mathcal{B}} = (P_{\mathcal{B} \leftarrow \mathcal{C}})^{-1}$ , so that matrix inversion can be used to compute the change of coordinate matrix the opposite way. It is also straightforward to show that the columns in the change of coordinate matrix can be obtained by expressing the old basis in terms of the new basis, i.e. finding the vectors  $[P_{\mathcal{B} \leftarrow \mathcal{C}}(\mathbf{v}_i)]_{\mathcal{C}}$ .

If  $L$  is a linear transformation between the spaces  $V$  and  $W$ , and  $\mathcal{B}$  is a basis for  $V$ ,  $\mathcal{C}$  a basis for  $W$ , we can consider the operation which sends the coordinates of  $\mathbf{v} \in V$  in the basis  $\mathcal{B}$  to the coordinates of  $L\mathbf{v} \in W$  in the basis  $\mathcal{C}$ . This is represented by a matrix, called *the matrix of  $L$  relative to the bases  $\mathcal{B}$  and  $\mathcal{C}$* . Similarly to change of coordinate matrices, the columns of the matrix of  $L$  relative to the bases  $\mathcal{B}$  and  $\mathcal{C}$  are given by  $[L(\mathbf{v}_i)]_{\mathcal{C}}$ .

## A.5 Eigenvectors and eigenvalues

If  $A$  is a linear transformation from a vector space to itself, a vector  $\mathbf{v}$  is called an *eigenvector* if there exists a scalar  $\lambda$  so that  $A\mathbf{v} = \lambda\mathbf{v}$ .  $\lambda$  is called the corresponding eigenvalue.

If the matrix  $A$  is symmetric, the following hold:

- The eigenvalues of  $A$  are real,
- the eigenspaces of  $A$  are orthonormal,
- any vector can be decomposed as a sum of eigenvectors from  $A$ .

For non-symmetric matrices, these results do not hold in general. But for filters, clearly the second and third property always hold, regardless of whether the filter is symmetric or not.

## A.6 Diagonalization

One can show that, for a symmetric matrix,  $A = PDP^T$  where  $D$  is a diagonal matrix and the eigenvalues of  $A$  are the values on the diagonal of  $D$ , and  $P$  is a matrix where the columns are the eigenvectors of  $A$ , with corresponding eigenvalue appearing in the same column in  $D$ .

## Appendix B

# Signal processing and linear algebra: a translation guide

This book should not be considered as a standard signal processing textbook. There are several reasons for this. First of all, much signal processing literature is written for people with an engineering background. This book is written for people with a basic linear algebra background. Secondly, the book does not give a comprehensive treatment of all basic signal processing concepts. Signal processing concepts are introduced whenever they are needed to encompass the mathematical exposition. In order to learn more about the different signal processing concepts, the reader can consult many excellent textbooks, such as [37, 1, 34, 43]. The translation guide of this chapter may be of some help in this respect, when one tries to unify material presented here with material from these signal processing textbooks. The translation guide handles both differences in notation between this book and signal processing literature, and topical differences. Most topical differences are also elaborated further in the summaries of the different chapters. The book has adopted most of its notation and concepts from mathematical literature.

### B.1 Complex numbers

There are several differences between engineering literature and mathematics. In mathematics literature,  $i$  is used for the imaginary complex number which satisfies  $i^2 = -1$ . In engineering literature, the name  $j$  is used instead.

### B.2 Functions

What in signal processing are referred to as continuous-time signals, are here referred to as functions. Usually we refer to a function by the letter  $f$ , according

to the mathematical tradition. The variable is mostly time, represented by the symbol  $t$ .

In signal processing, one often uses capital letters to denote a function which is the Fourier transform of another function, so that the Fourier transform of  $x$  would be denoted by  $X$ . Here we simply denote a periodic function by its Fourier coefficients  $y_n$ , and we avoid the CTFT. We use analog filters, however, which also work in continuous time. Analog filters preserve frequencies, and we have used  $\nu$  to denote frequency (variations per second), and not used angular frequency  $\omega$ . In signal processing literature it is common to jump between the two.

### B.3 Vectors

Discrete-time signals, as they are used in signal processing, are here mostly referred to as vectors. To as big extent as possible, we have attempted to keep vectors finite-dimensional. Vectors are in boldface (i.e.  $\mathbf{x}$ ), but its elements are not in boldface, and with subscripts (i.e.  $x_n$ ). Superscripts are also used to differ between vectors with the same base name (i.e.  $\mathbf{x}^{(1)}$ ,  $\mathbf{x}^{(2)}$  etc.), so that this does not interfere with the vector indices. In signal processing literature the corresponding notation would be  $x$  for the signal, and  $x[n]$  for its elements, and signals with equal base names could be named like  $x_1[n], x_2[n]$ .

We have sometimes denoted the Fourier transform of  $\mathbf{x}$  by  $\widehat{\mathbf{x}}$ , according to the mathematical tradition. More often we have distinguished between a vector and its Discrete Fourier transform by using  $\mathbf{x}$  for the first, and  $\mathbf{y}$  for the latter. This also makes us distinguish between the input and output to a filter, where we instead use  $\mathbf{z}$  for the latter. Much signal processing literature write (capital)  $X$  for the DFT of the vector  $x$ .

### B.4 Inner products and orthogonality

Throughout the book we have defined inner products for functions (for Fourier analysis and wavelets), and we have also used the standard inner product of  $\mathbb{R}^N$ . From this we have deduced the orthogonality of several basis functions used in signal processing theory. That the functions are orthogonal, as well as the inner product itself are, however, often not commented on in signal processing literature. As an unfortunate consequence, one has to explain the expression for the Fourier series using other means than the orthogonal decomposition formula and the least squares method. Also, one does not mention that the DFT is a unitary transformation.

### B.5 Matrices and filters

Boldface notation is not used for matrices, according to the mathematical tradition. In signal processing, it is not common to formulate matrix equations,

such as for the DFT and DCT, or matrix factorizations. Instead one typically writes down each equation, one equation for each row in  $\mathbf{y} = A\mathbf{x}$ , i.e. not recognizing matrix/vector multiplication. We have stuck to the name filtering operations, but made it clear that this is nothing but a linear transformation with a Toeplitz matrix as its matrix. In particular, we alternately use the terms filtering and multiplication with a Toeplitz matrix. The characterization of filters as circulant Toeplitz matrices is usually not done in signal processing literature (but see [16]). In this text we allow for matrices also to be of infinite dimensions, expanding on the common use in linear algebra. When infinite dimensions are assumed, infinite in both directions is assumed. Matrices are scaled if necessary to make them unitary, in particular the DCT and the DFT. This scaling is usually not done in signal processing literature.

Representing a filter in terms of a finite matrix and restriction of a filter to a finite signal. This is usually omitted in signal processing literature.

One of the most important statements in signal processing is that convolution in time is equivalent to multiplication in frequency. We have presented a compelling interpretation of this in linear algebra terms. Since the frequency response simply are eigenvalues of the filter, and convolution simply is matrix factorization, multiplication in frequency simply means to multiply two diagonal matrices to obtain the frequency response of the product. Moreover, the Fourier basis vectors can be interpreted as eigenvectors.

## B.6 Convolution

While we have defined the concept of convolution, readers familiar with signal processing may have noticed that this concept has not been used much. The reason is that we have wanted to present convolution as a matrix multiplication (to adapt to mathematical tradition), and that we have used the concept of filtering often instead. In signal processing literature one defines convolution in terms of vectors of infinite length. We have avoided this, since in practice vectors always need to be truncated to finite lengths. Due to this, we also have analyzed how a finite vector may be turned into a periodic vector (periodic or symmetric extension), and how this affects our analysis. Also we have concentrated on FIR-filters, and this makes us avoid convergence issues. Note that we do not present matrix multiplication as a method of implementing filtering, due to the special structure of this operation. We do not suggest other methods for implementation than applying the convolution formula in a brute-force way, or factoring the filter in simpler components.

## B.7 Polyphase factorizations and lifting

In signal processing literature, it is not common to associate polyphase components with matrices, but rather with Laurent polynomials generated from the corresponding filter. The Laurent polynomial is nothing else than the  $Z$ -

transform of the associated filter. Associating polyphase components with blocks in a block matrix makes this book fit with block matrix methods in linear algebra textbooks.

The polyphase factorization serves two purposes in this book. Firstly, the lifting factorization (as used for wavelets) is derived from it, and put in a linear algebra framework as a factorization into sparse matrices, similarly to the FFT factorization. Thereby it fits together with many of the matrix factorization results from classical linear algebra, where also sparsity is what makes the factorization good for computation.

Secondly, the polyphase factorization of the filter bank transforms in the MP3 standard are derived (also as a sparse matrix factorization), and from this it is apparent what properties to put on the prototype filters in order to obtain useful transforms. In fact, from this factorization it became apparent that the MP3 filter bank transforms could be expressed in terms of alternative QMF filter banks (i.e.  $M = 2$ ).

These two topics (lifting and the MP3 filter bank transform polyphase factorization) are usually not presented in a unified way in textbooks. We see here that there is a big advantage of doing this, since the second can build on theory from the first.

## B.8 Transforms in general

In signal processing, one often refers to the forward and reverse filter bank transforms as analysis and synthesis, respectively, and for obvious reasons. In mathematical literature, one instead often use the term change of coordinates in a wavelet setting. These terms are not normally used in mathematical literature, where the term basis vectors/change of coordinate matrices would be used instead. Also, the output from a forward filter bank transform is often referred to as the *transformed vector*, and the result we get when we apply the reverse filter bank transform to this is called the *reconstructed vector*.

This exposition takes extra care in presenting how the DCT is derived naturally from the DFT. In particular both the DFT and the DCT are derived as matrices of eigenvectors for finite-dimensional filters. The DCT is derived from the DFT in that one restricts to a certain subset of vectors. The orthogonality of these matrices follows from the orthogonality of distinct eigenspaces.

## B.9 Perfect reconstruction systems

The term biorthogonality is not used to describe a mutual property of the filters of wavelets. Borthogonality corresponds simply to two matrices being inverses of one another. For the same reason, the term perfect reconstruction is not used much. Much wavelet theory refer to a property called delay normalization. This terms has been avoided by mostly considering wavelets with symmetric filters,

for which delay-normalization is automatic. There are, however, many examples of wavelets where this term is important.

## B.10 $Z$ -transform and frequency response

The  $Z$ -transform and the frequency response are much used in signal processing literature, and are important concepts for filter design. We have deliberately dropped the  $Z$ -transform. Due to this, much signal processing has of course been left out, since placements of poles and zeroes are not performed outside or inside the unit circle, since the frequency response only captures the values on the unit circle. Placement of poles and circles is perhaps the most-used design feature in filter design. The focus here is on implementing filters, not designing them, however.

In signal processing literature, the DTFT and the  $Z$ -transform is used, assuming that the inputs and outputs are vectors of infinite length. In practice of course, some truncation is needed, since only finite-dimensional arithmetic is performed by the computer. How this truncation is to be done without affecting the computations is thus never mentioned in signal processing, although it is always performed somehow. This exposition shows that this truncation can be taken as part of the theory, without seriously affecting the results.



# Nomenclature

symbol	definition
$f_s$	Sampling frequency
$T_s$	Sampling period
$T$	Period of a function
$\nu$	Frequency
$f_N$	$N$ th order Fourier series of $f$
$V_{N,T}$	$N$ th order Fourier space
$\mathcal{D}_{N,T}$	Order $N$ real Fourier basis for $V_{N,T}$
$\mathcal{F}_{N,T}$	Order $N$ complex Fourier basis for $V_{N,T}$
$\check{f}$	Symmetric extension of the function $f$
$\lambda_s(\nu)$	Frequency response of a filter
$N$	Number of points in a DFT/DCT
$\mathcal{F}_N = \{\phi_0, \phi_1, \dots, \phi_{N-1}\}$	Fourier basis for $\mathbb{R}^N$
$F_N$	<i>NimesN</i> -Fourier matrix
$\hat{\mathbf{x}}$	DFT of the vector $\mathbf{x}$
$\bar{A}$	Conjugate of a matrix
$A^H$	Conjugate transpose of a matrix
$\mathbf{x}^{(e)}$	Vector of even samples
$\mathbf{x}^{(o)}$	Vector of odd samples
$O(N)$	Order of an algorithm
$l(S)$	Length of a filter
$\mathbf{x} * \mathbf{y}$	Convolution of vectors
$\lambda_{S,n}$	Vector frequency response of a digital filter
$E_d$	Filter which delays with $d$ samples
$\omega$	Angular frequency
$\lambda_S(\omega)$	Continuous frequency response of a digital filter
$\check{\mathbf{x}}$	Symmetric extension of a vector
$S_r$	Symmetric restriction of $S$
$S^f$	Matrix with the columns reversed
$\mathcal{D}_N = \{\mathbf{d}_0, \mathbf{d}_1, \dots, \mathbf{d}_{N-1}\}$	$N$ -point DCT basis for $\mathbb{R}^N$
$\text{DCT}_N$	$N \times N$ -DCT matrix

symbol	definition
$\phi$	Scaling function
$V_m$	Resolution space
$\mathbf{r}\phi_m$	Basis for $V_m$
$c_{m,n}$	Coordinates in $\phi_m$
$W_m$	Detail space
$\mathbf{r}U \oplus V$	Direct sum of vector spaces
$\psi_m$	Basis for $W_m$
$w_{m,n}$	Coordinates in $\psi_m$
$C_m$	Reordering of $(\phi_{m-1}, \psi_{m-1})$
$\tilde{\phi}$	Dual scaling function
$\tilde{\psi}$	Dual mother wavelet
$\tilde{V}_m$	Dual resolution space
$\tilde{W}_m$	Dual detail space
$D_m$	Reordering of $\phi_m$
$\mathcal{E}_N = \{\mathbf{e}_0, \mathbf{e}_1, \dots, \mathbf{e}_{N-1}\}$	Standard basis for $\mathbb{R}^N$
$\otimes$	Tensor product
$W_m^{(0,1)}$	Resolution $m$ Complementary wavelet space, LH
$W_m^{(1,0)}$	Resolution $m$ Complementary wavelet space, HL
$W_m^{(1,1)}$	Resolution $m$ Complementary wavelet space, HH
$A^T$	Transpose of a matrix
$A^{-1}$	Inverse of a matrix
$\langle \mathbf{u}, \mathbf{v} \rangle$	Inner product
$[\mathbf{x}]_{\mathcal{B}}$	Coordinate vector of $\mathbf{x}$ relative to the basis $\mathcal{B}$
$P_{\mathcal{C} \leftarrow \mathcal{B}}$	Change of coordinate matrix from $\mathcal{B}$ to $\mathcal{C}$

# Bibliography

- [1] A. Ambardar. *Digital Signal Processing: a Modern Introduction*. Cengage Learning, 2006.
- [2] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.
- [3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [4] C. M. Brislawn. Fingerprints go digital. *Notices of the AMS*, 42(11):1278–1283, 1995.
- [5] B. A. Cipra. The best of the 20th century: Editors name top 10 algorithms. *SIAM News*, 33(4), 2000. <http://www.uta.edu/faculty/rcli/TopTen/topten.pdf>.
- [6] A. Cohen and I. Daubechies. Wavelets on the interval and fast wavelet transforms. *Applied and computational harmonic analysis*, 1:54–81, 1993.
- [7] A. Cohen, I. Daubechies, and J-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Appl. Math.*, 45(5):485–560, June 1992.
- [8] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comp.*, 19:297–301, 1965.
- [9] A. Croisier, D. Esteban, and C. Galand. Perfect channel splitting by use of interpolation/decimation/tree decomposition techniques. *Int. Conf. on Information Sciences and Systems*, pages 443–446, August 1976.
- [10] G. Dahl. A note on diagonally dominant matrices. *Linear Algebra and its Appl.*, 317(1-3):217–224, 2000.
- [11] G. Dahl. *An Introduction to Convexity*. Report, University of Oslo, 2010.
- [12] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Appl. Math.*, 41(7):909–996, October 1988.
- [13] I. Daubechies. *Ten Lectures on Wavelets*. CBMS-NSF conference series in applied mathematics. SIAM Ed., 1992.

- [14] P. Duhamel and H. Hollmann. 'split-radix' FFT-algorithm. *Electronic letters*, 20(1):14–16, 1984.
- [15] FBI. WSQ gray-scale fingerprint image compression specification. Technical report, IAFIS-IC, 1993.
- [16] M. W. Frazier. *An Introduction to Wavelets Through Linear Algebra*. Springer, 1999.
- [17] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.
- [18] R. C. Gonzalez, R. E. Woods, and S. L. Eddins. *Digital Image Processing Using MATLAB*. Gatesmark publishing, 2009.
- [19] J. B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms I*. Springer, 1993.
- [20] ISI/IEC. Information technology - coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s. Technical report, ISO/IEC, 1993.
- [21] ISO/IEC. Jpeg2000 part 1 final draft international standard. iso/iec fdis 15444-1. Technical report, ISO/IEC, 2000.
- [22] S.G Johnson and M. Frigo. A modified split-radix FFT with fewer arithmetic operations. *IEEE Transactions on Signal Processing*, 54, 2006.
- [23] J.D. Johnston. A filter family designed for use in quadrature mirror filter banks. *Proc. Int. Conf. Acoust. Speech and Sig. Proc.*, pages 291–294, 1980.
- [24] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995.
- [25] D. C. Lay. *Linear Algebra and Its Applications (4th Edition)*. Addison-Wesley, 2011.
- [26] T. Lindstrøm and K. Hveberg. *Flervariabel Analyse Med Lineær Algebra*. Pearson, 2011.
- [27] D.G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 1984.
- [28] T. Lyche. *Numerical Linear Algebra*. Report, University of Oslo, 2010.
- [29] S. Mallat. *A Wavelet Tour of Signal Processing*. Tapir Academic Press, 1998.
- [30] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2000.
- [31] Knut Mørken. *Numerical Algorithms and Digital Representation*. UIO, 2013.

- [32] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 2006.
- [33] P. Noll. MPEG digital audio coding. *IEEE Signal processing magazine*, pages 59–81, September 1997.
- [34] A. V. Oppenheim and R. W. Schaffer. *Discrete-Time Signal Processing*. Prentice Hall, 1989.
- [35] D. Pan. A tutorial on MPEG/audio compression. *IEEE Multimedia*, pages 60–74, Summer 1995.
- [36] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.
- [37] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing. Principles, Algorithms, and Applications. Fourth Edition*. Pearson, 2007.
- [38] C. M. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56:1107–1108, June 1968.
- [39] T. A. Ramstad, S. O. Aase, and J. H. Husøy. *Subband Compression of Images: Principles and Examples: Principles and Examples*, volume 6. Elsevier Science, 1995.
- [40] R.T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.
- [41] A. Ruszczyński. *Nonlinear Optimization*. Princeton University Press, 2006.
- [42] C. E. Shannon. Communication in the presence of noise. *Proc. Institute of Radio Engineers*, 37(1):10–21, Jan. 1949.
- [43] P. Stoica and R. Moses. *Spectral Analysis of Signals*. Prentice Hall, 2005.
- [44] W. Sweldens. The lifting scheme: a new philosophy in biorthogonal wavelet constructions. *Wavelet Applications in Signal and Image Processing III*, pages 68–79, 1995.
- [45] W. Sweldens. The lifting scheme: a custom-design construction of biorthogonal wavelets. *Applied and computational harmonic analysis*, 3:186–200, 1996.
- [46] D. S. Taubman and M. W. Marcellin. *JPEG2000. Image Compression. Fundamentals, Standards and Practice*. Kluwer Academic Publishers, 2002.
- [47] M. Vetterli and J. Kovacevic. *Wavelets and Subband Coding*. Prentice Hall, 1995.
- [48] M. Vetterli and H. J. Nussbaumer. Simple FFT and DCT algorithms with reduced number of operations. *Signal Processing*, 6:267–278, 1984.
- [49] R. Webster. *Convexity*. Oxford University Press, Oxford, 1994.

- [50] S. Winograd. On computing the discrete Fourier transform. *Math. Comp.*, 32:175–199, 1978.
- [51] R. Yavne. An economical method for calculating the discrete Fourier transform. *Proc. AFIPS Fall Joint Computer Conf.*, 33:115–125, 1968.

# Index

- AD conversion, 1
- affine function, 398
- algebra, 96
- Alias cancellation, 233
- Alias cancellation condition, 236
- Aliasing, 233
- analysis, 14
  - equations, 14
- Analysis filter components of a forward filter bank transform, 241
- Angular frequency, 104
- Arithmetic operation count
  - DCT, 155
  - DFT direct implementation, 55
  - FFT, 77
  - revised DCT, 158
  - revised FFT, 158
  - symmetric filters, 149
  - with tensor products, 347
- audioread, 6
- audiowrite, 6
- Backtracking line search, 420
- Bandpass filter, 112
- barrier method, 453
- barrier problem, 454
- Basis
  - $\mathcal{C}$ , 175
  - $\mathcal{D}$ , 294
  - $\phi_m$ , 166
  - $\psi_m$ , 170
  - DCT, 141
  - for  $V_{N,T}$ , 14, 22
  - Fourier, 50
- basis, 463
- Biorthogonal
  - bases, 256
- Biorthogonality, 256
- bit rate, 1
- Bit-reversal
  - DWT, 291
  - FFT, 73
- block diagonal matrices, 175
- block matrix, 71
- Blocks, 352
- Cascade algorithm, 208
- Causal filter, 269
- central path, 456
- central point, 456
- chain rule, 393
- Change of coordinate matrix, 464
- Change of coordinates, 464
  - in tensor product, 345
- Channel, 241
- Compact support, 37
- Complex Fourier coefficients, 23
- Computational molecule, 329
  - Partial derivative in  $x$ -direction, 335
  - Partial derivative in  $y$ -direction, 336
  - Second order derivatives, 338
  - smoothing, 334
- concave, 398
- condition number, 422
- Conjugate transpose, 52
- continuous
  - sound, 1
- Continuous-time Fourier transform, 47

- contraction, 408
- convex combination, 399
- convex function (multivariate), 397
- convex optimization, 444
- convex set, 396
- Convolution
  - analog, 37
  - kernel, 37
  - vectors, 91
- convolve, 92
- coordinate matrix, 345
- Coordinate vector, 464
- Coordinates in  $\phi_m$ , 167
- Coordinates in  $\psi_m$ , 170
- Cosine matrices, 142
- Cosine matrix inverse
  - type I, 230
  - type II, 142
  - type III, 142, 146
- critical sampling, 214
- CTFT, 47
- DCT
  - I, 229
- dct, 143
- DCT basis, 141
- DCT coefficients, 141
- DCT matrix, 141
- DCT-I factorization, 229
- DCT-II factorization, 142
- DCT-III factorization, 142
- DCT-IV factorization, 146
- Detail space, 169
- DFT coefficients, 51
- DFT matrix factorization, 72
- Diagonalization
  - with  $F_N$ , 95
- diagonally dominant, 401
- digital
  - sound, 1
- digital filter, 95
- Direct sum
  - linear transformations, 185
  - vector spaces, 169
- Dirichlet kernel, 44
- Discrete Cosine transform, 141
- Discrete Fourier transform, 51
- Discrete Wavelet Transform, 173
- downsampling, 214
- Dual
  - detail space, 257
  - mother wavelet, 255
  - multiresolution analysis, 257
  - resolution space, 257
  - scaling function, 255
  - wavelet transforms, 220
- dual problem, 446
- duality, 445
- duality gap, 446
- DWT kernel parameter `dual`, 220
- DWT parameter `bd_mode`, 226
- eigenvalue, 465
- eigenvector, 465
- elementary lifting matrix
  - even type, 287
  - odd type, 287
  - used for non-symmetric filters, 299
  - used for symmetric filters, 291
- error-resilient, 352
- Fejer kernel, 44
- FFT, 70
  - twiddle factors, 79
- fft, 74
- FFT algorithm
  - Non-recursive, 81
  - Radix, 82
  - Split-radix, 82
- Filter
  - bandpass, 112
  - high-pass, 112
  - ideal high-pass, 112
  - ideal low-pass, 112
  - length, 90
  - linear phase, 138
  - low-pass, 112
  - moving average, 113
  - MP3 standard, 117
  - time delay, 96
- Filter bank, 241
  - Cosine-modulated, 245



- Filter bank transform, 241
- Filter coefficients, 88
- Filter echo, 112
- FIR filters, 125
- First crime of wavelets, 207
- fixed point, 407
- flop count, 85
- Forward filter bank transform, 241
  - in a wavelet setting, 216
- Fourier analysis, 21
- Fourier coefficients, 12
- Fourier domain, 14
- Fourier matrix, 51
- Fourier series, 12
  - square wave, 14
  - triangle wave, 17
- Fourier space, 12
- Frequency domain, 14
- Frequency response
  - continuous, 104
  - filter, 37
  - vector, 95
- gradient, 391
- gradient method, 420
- gradient related, 421
- Haar wavelet, 180
- Hessian matrix, 391
- Highpass filter, 112
- idct, 143
- Ideal high-pass filter, 112
- Ideal low-pass filter, 112
- IDFT, 52
- IDFT matrix factorization, 72
- ifft, 74
- IMDCT, 147
- Implementation
  - Cascade algorithm to plot wavelet functions, 262
  - DCT, 153
  - DCT2, 349
  - DFT, 55
  - FFT
    - Nonrecursive, 81
    - revised, 157
    - Split-radix, 82
  - FFT2, 349
  - Filtering an image, 331
  - Generic DWT, 179
  - Generic DWT2, 369
  - Generic IDWT, 179
  - Generic IDWT2, 369
  - IDCT, 154
  - IDCT2, 349
  - IFFT2, 349
  - lifting step
    - non-symmetric, 300
  - listening to detail part in sound, 181
  - listening to high-frequency part in sound, 65
  - listening to low-frequency part in sound, 65
  - listening to low-resolution part in sound, 181
  - Tensor product, 341
  - Wavelet kernel
    - alternative piecewise linear wavelet, 200
    - alternative piecewise linear wavelet with 4 vanishing moments, 302
    - CDF 9/7 wavelet, 300
    - Haar wavelet, 178
    - orthonormal wavelets, 301
    - piecewise linear wavelet, 194
    - piecewise quadratic wavelet, 302
    - Spline 5/3 wavelet, 300
- impulse response, 98
- imread, 322
- imshow, 322
- imwrite, 322
- In-place
  - bit-reversal implementation, 73
  - DWT implementation, 178
  - FFT implementation, 73
  - lifting implementation, 291
- In-place implementation
  - DWT, 291
- Inner product

- of functions in a Fourier setting, 11
  - of functions in a tensor product setting, 355
  - of functions in a wavelet setting, 163
  - of vectors, 50
- interior point method, 453
- interpolating polynomial, 62
- interpolation formula, 64
  - ideal
    - periodic functions, 64
- Inverse Discrete Wavelet Transform, 173
- Jacobi matrix, 391
- Jensen 's inequality, 399
- JPEG
  - standard, 351
- JPEG2000
  - lossless compression, 275
  - lossy compression, 277
  - standard, 275
- Karush-Kuhn-Tucker conditions, 438
- Kernel transformations, 175
- KKT conditions, 438
- Kronecker tensor product, 343
- Lagrangian function, 433
- least square error, 11
- length of a filter, 90
- Lifting factorization, 290
  - alternative piecewise linear wavelet, 295
  - alternative piecewise linear wavelet with 4 vanishing moments, 301
  - CDF 9/7 wavelet, 296
  - orthonormal wavelets, 299
  - piecewise linear wavelet, 295
  - piecewise quadratic wavelet, 302
  - Spline 5/3 wavelet, 296
- linear convergence, 393
- linear optimization, 389
- Linear phase filter, 138
- linearized feasible direction, 440
- linearly independent, 463
- Lipschitz, 408
- logarithmic barrier function, 455
- loglog, 80
- Lowpass filter, 112
- LTI filters, 97
- matrix of a linear transformation relative to bases, 465
- maximum, 383
- maximum likelihood, 387
- MDCT, 146
- minimum, 383
- mother wavelets, 173
- MP3
  - and the DCT, 159
  - FFT, 67
  - filters, 117
  - standard, 45
  - window, 108
- MP3 standard
  - matrixing, 243
  - partial calculation, 243
  - windowing, 243
- MRA-matrix, 213
- multiresolution analysis, 205
- multiresolution model, 163
- Near-perfect reconstruction, 233
- Newton's method, 410, 420, 422
- Newton's method with equality constraints, 453
- nonlinear equations, 407
- nonlinear optimization problem, 432
- objective function, 383
- optimal control, 389
- optimality conditions, 416
- Order  $N$  complex Fourier basis for  $V_{N,T}$ , 22
- Order of an algorithm, 75
- Orthogonal
  - basis, 464
  - matrix, 463
  - vectors, 464
- Orthogonal decomposition theorem, 464
- Orthonormal

- basis, 464
  - MRA, 206
- Orthonormal wavelets, 240
- Outer product, 331
- Parallel computing
  - with the DCT, 155
  - with the DWT, 352
  - with the FFT, 77
- penalty term, 437
- Perfect reconstruction, 233
- perfect reconstruction condition, 236
- Perfect reconstruction filter bank, 242
- Periodic function, 5
- Phase distortion, 233
- play, 6
- polyhedron, 397
- Polyphase
  - component of a vector, 77
- Polyphase components, 285
- Polyphase representation, 285
- positive definite, 390
- positive semidefinite, 390
- primal problem, 446
- projection, 464
- psycho-acoustic model, 45
- pure digital tone, 50
- pure tone, 5
- QMF filter banks, 239
  - Alternative definition, 240
  - Classical definition, 239
- quasi-Newton method, 411
- Regular point, 433
- Resolution space, 165
- Reverse filter bank transform
  - in a wavelet setting, 216
- Reverse filter bank transforms, 242
- roots, 268
- samples, 1
- sampling, 1
  - frequency, 1
  - period, 1
  - rate, 1
- scaling function, 167, 206
- separable extension, 354
- sound channel, 7
- Sparse matrix, 463
- square wave, 7
- Standard
  - JPEG, 351
  - JPEG2000, 275
  - MP3, 47
- stationary point, 416
- steepest descent method, 420
- strictly convex, 416
- subband
  - HH, 364
  - HL, 364
  - LH, 364
  - LL, 364
- Subband coding, 241
- Subband samples of a filter bank transform, 241
- superlinear convergence, 393
- Support, 37
- Symmetric
  - vector, 133
- Symmetric extension
  - of function, 33
  - used by the DCT, 133
  - used by wavelets, 224
- Symmetric restriction of a symmetric filter, 138
- synthesis, 14
  - equation, 14
  - vectors, 52
- Synthesis filter components of a reverse filter bank transform, 242
- tangent vector, 440
- tensor product, 316
  - of function spaces, 354
  - of functions, 354
  - of matrices, 332
  - of vectors, 331
- Tiles, 352
- time domain, 14
- time-invariant, 96
- toc, 80

- Toeplitz matrix, 88
  - circulant, 88
- Transpose DWT, 226
- Transpose IDWT, 226
- triangle wave, 8
  
- Unitary matrix, 52
- upsampling, 215
  
- Vector space
  - of symmetric vectors, 133
  
- Wavelets
  - Alternative piecewise linear, 198
  - CDF 9/7, 276
  - Orthonormal, 279
  - Piecewise linear, 191
  - Spline, 273
  - Spline 5/3, 275
- weak duality, 445
- weak Slater condition, 447
- window, 108
  - Hamming, 108
  - Hanning, 111
  - in the MP3 standard, 108
  - rectangular, 108