Digital Communications and Signal Processing

An Introduction using Octave or Matlab

Aldebaro Klautau
Federal University of Pará (UFPA), Brazil

October 13th, 2013

Aldebaro Barreto da Rocha Klautau Junior
Ph. D. University of California at San Diego (UCSD), 2003
Professor at the Federal University of Pará (UFPA)
Belém – Pará – Brazil
Email: a.klautau@ieee.org

ISBN-13: 978-85-911100-2-5

Created in Brazil

Visit the book web site: http://www.aldebaro.ufpa.br

Contents

List of Figures viii

List of Tables xiii

Listings xiv

Preface xvi

1 Analog and Digital Signals 1
- 1.1 To Learn in This Chapter . . . 1
- 1.2 Analog and Digital Signals . . . 1
- 1.3 Digitizing Signals . . . 3
- 1.4 Basic Signal Manipulation and Representation . . . 3
  - 1.4.1 Manipulating the independent variable . . . 4
  - 1.4.2 Using impulses to represent signals . . . 5
  - 1.4.3 Using step functions to represent signals . . . 6
- 1.5 Complex, Sampled and Discrete-time Signals . . . 7
  - 1.5.1 Complex-valued signals . . . 7
  - 1.5.2 Discrete-time signals . . . 7
  - 1.5.3 Sampled signals . . . 8
- 1.6 Modeling the Sampling Step . . . 9
- 1.7 C/D and D/C conversions . . . 12
- 1.8 Relating Continuous-time, Sampled and Discrete-time Signals . . . 14
  - 1.8.1 Nyquist frequency . . . 16
  - 1.8.2 Frequency normalization in Matlab/Octave . . . 16
- 1.9 An Introduction to Quantization . . . 17
  - 1.9.1 Uniform quantization . . . 17
  - 1.9.2 Granular and overload regions . . . 19
  - 1.9.3 Quantizer output: Real or binary number . . . 20
  - 1.9.4 Coding and decoding for quantization . . . 21
  - 1.9.5 Representing numbers in fixed-point . . . 23
  - 1.9.6 IEEE 754 floating-point standard . . . 26
- 1.10 Signal Classification: Randomness, Periodicity and Power . . . 30
  - 1.10.1 Random signals and their generation . . . 30
  - 1.10.2 Periodic and aperiodic signals . . . 33
  - 1.10.3 Power and energy signals . . . 35
- 1.11 Power and Energy in Discrete-Time . . . 38
  - 1.11.1 Power and energy of discrete-time signals . . . 38

1.11.2 Power and energy of signals represented as vectors 39
1.11.3 Power and energy of vectors whose elements are not time-ordered 39
1.11.4 Power and energy of discrete-time random signals 40
1.12 Relating Power in Continuous and Discrete-Time 40
1.13 Correlation: A Measure of Dependence / Similarity 41
1.13.1 Autocorrelation function 42
1.13.2 Cross-correlation 47
1.14 A Linear Model for Quantization 51
1.15 Applications 52
1.16 Comments and Further Reading 73
1.17 Review Exercises 74
1.18 Exercises 76

2 Transforms and Signal Representation 80
2.1 To Learn in This Chapter 80
2.2 Matrix multiplication is a linear transform 80
2.3 Inner Products to Obtain the Transform Coefficients 82
2.4 Block Transforms 84
2.4.1 DCT transform 86
2.4.2 DFT transform 89
2.4.3 Haar transform 92
2.4.4 Unitary matrices lead to energy conservation 94
2.4.5 Orthogonal but not unitary also allows easy inversion 95
2.5 Fourier Transforms and Series 96
2.5.1 Fourier series for continuous-time signals 98
2.5.2 Discrete-time Fourier series (DTFS) 103
2.5.3 Continuous-time Fourier transform using frequency in Hertz . . . 105
2.5.4 Continuous-time Fourier transform using frequency in rad/s . . . 106
2.5.5 Discrete-time Fourier transform (DTFT) 106
2.6 Relating discrete and analog frequencies 107
2.7 Summary of equations for DFT / FFT Usage 108
2.8 Laplace Transform 111
2.9 Z Transform 117
2.9.1 Some pairs and properties of the Z-transform 118
2.9.2 Z-transform region of convergence 118
2.10 Applications 122
2.11 Comments and Further Reading 143
2.12 Review Exercises 144
2.13 Exercises 144

3 Analog and Digital Systems 148

4 Spectral Estimation Techniques 149

5 Principles of Communications with Focus on AM 150

6 Baseband Digital Modulation with Focus on PAM 151

7 Passband Digital Modulation with Focus on QAM 152

8 Channels: Modeling, Simulating and Mitigating their Effects 153

9 Optimal Receivers and Probability of Error for AWGN 154

10 Multicarrier systems 155

Appendices 157

Appendix A Useful Mathematics 157

A.1 Even and odd functions . . . 157
A.2 Euler's formula . . . 157
A.3 Trigonometry . . . 157
A.4 Manipulating complex exponentials . . . 158
A.5 Q function . . . 158
A.6 Matched filter and Cauchy-Schwarz's inequality . . . 160
A.7 Geometric series . . . 160
A.8 Sum of squares . . . 160
A.9 Summations and integrals . . . 160
A.10 Partial fraction decomposition . . . 161
A.11 Calculus . . . 164
A.12 Sinc Function . . . 164
A.13 Rectangular Integration to Define Normalization Factors for Functions . 164
A.13.1 Two normalizations for the histogram . . . 165
A.13.2 Two normalizations for PSD estimation via FFT . . . 166
A.14 Linear Algebra . . . 167
A.14.1 Inner products and norms . . . 167
A.14.2 Projection of a vector using inner product . . . 168
A.14.3 Orthogonal basis allows inner products to transform signals . . . 170
A.14.4 Moore-Penrose pseudoinverse . . . 172
A.15 Gram-Schmidt orthonormalization procedure . . . 173
A.16 Principal component analysis (PCA) . . . 174
A.17 Fourier Analysis: Properties . . . 178
A.18 Fourier Analysis: Pairs . . . 179
A.19 Probability and Stochastic Processes . . . 181
A.19.1 Random variables . . . 181
A.19.2 Expected Values . . . 183
A.19.3 Orthogonal versus uncorrelated . . . 183
A.19.4 PDF of a sum of two independent random variables . . . 183
A.19.5 Stochastic (random) processes . . . 184
A.19.6 Proper and circular complex-valued random variables . . . 196
A.19.7 Rotationally symmetric signal constellations . . . 198

A.19.8 Cyclostationary random processes . . . 199
A.19.9 Two cyclostationary signals: sampled and discrete-time upsampled 205
A.19.10 Converting a WSC into WSS by randomizing the phase . . . 207
A.20 Estimation Theory . . . 212
A.20.1 Probabilistic estimation theory . . . 212
A.20.2 Minimum mean square error (MMSE) estimators . . . 213
A.20.3 Orthogonality principle . . . 213
A.21 One-dimensional linear prediction over time . . . 213
A.21.1 The innovations process . . . 213
A.22 Vector prediction exploring spatial correlation . . . 216
A.23 Spatial whitening applied to interference mitigation . . . 218
A.24 Space-time prediction . . . 219
A.25 Non-linear decorrelation . . . 219
A.26 noise prediction and DFE . . . 219
A.27 Decibel (dB) and Related Definitions . . . 219
A.28 Insertion loss and insertion frequency response . . . 221
A.29 Discrete and Continuous-Time Impulses . . . 223
A.29.1 Discrete-time impulse function . . . 223
A.29.2 Why defining the continuous-time impulse? Some motivation . . . 223
A.29.3 Definition of the continuous-time impulse as a limit . . . 224
A.29.4 Continuous-time impulse is a distribution, not a function . . . 224
A.29.5 Mathematical properties of the continuous-time impulse . . . 225
A.29.6 Convolution with an impulse . . . 226
A.30 System Properties . . . 226
A.30.1 Linearity (additivity and homogeneity) . . . 226
A.30.2 Time-invariance (or shift-invariance) . . . 227
A.30.3 Memory . . . 228
A.30.4 Causality . . . 228
A.30.5 Invertibility . . . 229
A.30.6 Stability . . . 229
A.30.7 Properties of Linear and time-invariant (LTI) systems . . . 229

Appendix B Useful Softwares and Programming Tricks 230
B.1 The GNU Radio (GR) Project . . . 230
B.1.1 For those willing to stop reading and run code . . . 230
B.1.2 Basic facts about GR . . . 231
B.2 GR-Based SDR Hardware (USRP, RTL-SDR Dongle, etc.) . . . 232
B.2.1 Universal Software Radio Peripheral (USRP) . . . 232
B.3 Using GNU Radio and USRP . . . 234
B.4 Matlab and Octave . . . 234
B.4.1 Octave Installation . . . 236
B.5 Manipulating signals stored in files . . . 237
B.5.1 Hex / Binary File Editors . . . 237
B.5.2 ASCII Text Files: Unix/Linux versus Windows . . . 237

B.5.3 Binary Files: Big versus Little-endian 239
B.5.4 Some Useful Code to Manipulate Files 240
B.5.5 Interpreting binary files with complex headers 243

Glossary **248**
Text Conventions 248
Main Abbreviations 250
Main Symbols 252

Bibliography **255**

Index **258**

List of Figures

1.1 Example of analog signal. Note the abscissa is continuous and the amplitude is not quantized. . . . 2
1.2 Example of digital signal obtained by digitalizing the analog signal in Figure 1.1 . . . 2
1.3 Example of discrete-time signal . . . 8
1.4 Signals classification including the sampled signals. . . . 8
1.5 Example of sampled signal . . . 9
1.6 An impulse train with unitary areas and $T_s = 125$ μs. . . . 10
1.7 Example of a sampled signal obtained with a sampling frequency smaller than the required for accurately representing the original signal (shown in dotted lines). . . . 10
1.8 Example of C/D conversion assuming $T_s = 0.2$ s. . . . 12
1.9 Signal reconstruction with a ZOH. . . . 13
1.10 Input/output relation of a 3-bits quantizer with $\Delta = 1$. . . . 19
1.11 Input/output relation of a 3-bits quantizer with $\Delta = 0.5$. . . . 20
1.12 A quantizer Q that outputs real numbers can be seen as containing a quantizer $\tilde{Q}_b$ that outputs binary numbers followed by a decoder. . . . 21
1.13 Fixed-point representation Q3.4 of the real number 5.0625. . . . 24
1.14 Comparison of step sizes for IEEE 754 floating points with single and double precision . . . 28
1.15 Example of conversion using proportion when the dynamic ranges of both analog and digital signal are available . . . 30
1.16 Waveform representation of a random signal with 100 samples draw from a Gaussian distribution $\mathcal{N}(0, 1)$. . . . 31
1.17 Histogram of the signal in Figure 1.16 with 10 bins. . . . 31
1.18 PDF estimate from the histogram in Figure 1.17 . . . 32
1.19 Comparison of normalized histogram and the correct Gaussian $\mathcal{N}(4, 0.09)$ when using 10,000 samples and 100 bins . . . 33
1.20 Graph of the signal $x[n] = \sin(0.2n)$. . . . 36
1.21 Graph of the signal $x[n] = \sin((3\pi/17)n)$. . . . 36
1.22 A sinusoid of period N=8 samples and its autocorrelation, which is also periodic each 8 lags . . . 47
1.23 The a) unbiased and b) raw (unscaled) autcorrelations for the sinusoid of Figure 1.22 with a new period of N=15 samples. . . . 48
1.24 Sinusoid of amplitude 4 V immersed in AWGN of power 25 W. The bottom graph is a zoom showing the first 100 samples. . . . 49
1.25 Autocorrelations of sine plus noise . . . 49
1.26 Continuous-time version of the AWGN channel model. . . . 51

1.27 Example of sound recorded at $F_s = 44.1$ kHz with the Audacity sound editor. . . . 53
1.28 Some Audacity options for saving an uncompressed WAVE file. The two non-linear PCMs are indicated. . . . 54
1.29 Cosine obtained with Listing 1.17 and a loopback cable connecting the soundboard DAC and ADC. . . . 57
1.30 Setup for loopback of the sound system using an audio cable. . . . 59
1.31 Example of options provided by Windows and the sound board. All the enhancements for both recording and playback devices should be disabled. 60
1.32 Audacity window after reading in the 'impulses.wav' file. . . . 60
1.33 Audacity window after simultaneously recording and playing 'impulses.wav' with a loopback. . . . 61
1.34 Zoom of the response to the second impulse in Figure 1.32. . . . 62
1.35 Representation of signals related by $y[n] = x[-n]$ 64
1.36 Scatter plot of customer age versus purchased units for three products. . 66
1.37 Autocorrelation of the sunspot data. . . . 68
1.38 Autocorrelation of a cosine of 300 Hz 70
1.39 First graph shows signals $x(t)$ and $y(t - 0.25)$ contaminated by AWGN at a SNR of 10 dB. . . . 72
1.40 Example of continuous-time signal. . . . 76

2.1 Rotation of a vector $\mathbf{x}$ by an angle $\theta = \pi/2$ radians using $\mathbf{y} = \mathbf{Ax}$ with $\mathbf{A}$ given by Eq. (2.2). . . . 81
2.2 The first three ($k = 0, 1, 2$) and the last ($k = 31$) basis functions for a 32-points DCT. . . . 87
2.3 The angles corresponding to W_N in the unity circle, for $N = 3, 4, 5, 6$. . 90
2.4 Computational cost of the DFT calculated via matrix multiplication versus an FFT algorithm. Note that $N = 4096$ are currently used in standards such as VDSL2, for example, and it is clearly unreasonable to use matrix multiplication. . . . 92
2.5 The first four ($k = 0, 1, 2, 3$) and the last ($k = 31$) basis functions for a 32-points Haar. . . . 93
2.6 Basis functions $k = 0, 17$ and the last four ($k = 28, 29, 30, 31$) for a 32-points Haar. . . . 94
2.7 Fourier series basis functions for analyzing signals with period $T_0 = 1/50$ seconds. Because the basis functions are complex-valued signals, the plots show their real (top) and imaginary (bottom) parts. . . . 101
2.8 DTFS / DFT of $x[n] = 10\cos(\frac{\pi}{6}n + \pi/3)$ calculated with $N = 12$. . . . 104
2.9 Complete representation of the DTFS / DFT of $x[n] = 10\cos(\frac{\pi}{6}n + \pi/3)$ indicating the periodicity $X[k] = X[k + N]$. . . . 105
2.10 Spectrum $X(f)$ (top) and $X(e^{j\Omega})$ when $F_s = 60$ Hz. . . . 108
2.11 Real part of $e^{\sigma+j10\pi}$. The values of σ are 0.3 and -0.3 for the first (left) and second graphs, respectively. . . . 112
2.12 Magnitude (in dB) of Eq. (2.42). . . . 114

2.13 Phase (in rad) of Eq. (2.42). 114
2.14 Graph of Figure 2.12 with the identification of the corresponding values of the Fourier transform (magnitude). 115
2.15 The values of the magnitude of the Fourier transform corresponding to Figure 2.14. 116
2.16 Two dimensional representation of Figure 2.15 obtained with the command freqs in Matlab/Octave showing the peak at $\omega = 2$ rad/s due to the respective pole. 116
2.17 Magnitude (in dB) of Eq. (2.49). 122
2.18 Phase (in rad) of Eq. (2.49). 122
2.19 Pole / zero diagram for Eq. (2.49). 123
2.20 Graph of Figure 2.17 with the identification of the corresponding values of the DTFT (unity circle $|z| = 1$). 123
2.21 The values of the magnitude of the DTFT corresponding to Figure 2.20. 124
2.22 Magnitude (top) and phase (bottom) of the DTFT corresponding to Eq. (2.49). These plots can be obtained with the Matlab/Octave command freqz and are a more convenient representation than, e. g., Figure 2.21. 124
2.23 Signal $x[n] = \delta[n - 11]$ analyzed by 32-points DCT and Haar transforms. 126
2.24 A segment of one channel of the original ECG data. 127
2.25 Original and reconstructed ECG signals with DCT of $N = 32$ points and discarding 26 (high-frequency) coefficients. 128
2.26 Performance of five DCT-based ECG coding schemes. The number of points is varied $N \in \{4, 8, 32, 64, 128\}$ and $K = 1, 2, \ldots, M - 1$. 129
2.27 A zoom of the eye region of the Lenna image. 130
2.28 Spectrum of $x(t) = 4 + 10\cos(2\pi 50t + 2) + 4\sin(2\pi 150t - 1)$. 132
2.29 Unilateral spectrum of (real) signal $x(t) = 4 + 10\cos(2\pi 50t + 2) + 4\sin(2\pi 150t - 1)$. 133
2.30 Alternative representation of the DTFS / DFT of $x[n] = 10\cos(\frac{\pi}{6}n + \pi/3)$ using fftshift. 135
2.31 Five cosine signals $x_i[n] = 10\cos(\Omega_i n)$ with frequencies $\Omega_i = 0$, $2\pi/32$, $4\pi/32$, π, $31\pi/16$ for $i = 0, 1, 2, 16, 32$, and the real part of their DTFS using $N = 32$ points. 136
2.32 Analysis with DFT of 32 points of $x[n]$ composed by three sinusoids and a DC level. 137
2.33 Spectrum of a signal $x[n] = 4\cos((2\pi/6)n)$ with period of 6 samples obtained with a 16-points DFT, which created spurious components. . . 138
2.34 Explicitly repeating the block of N cosine samples from Figure 2.33 to indicate that spurious components are a manifest of the lack of a perfect cosine in time-domain. 138
2.35 Three periods of each signal: pulse train $x[n]$ with $N = 10$ and $N_1 = 5$ and amplitude assumed to be in Volts (a), the magnitude (b) and phase (c) of its DTFS. 139
2.36 Behavior when N_1 of Figure 2.35 is decreased from $N_1 = 4$ to 1. 140

2.37 DTFT of an aperiodic pulse with $N_1 = 5$ non-zero samples and DTFT estimates obtained via a DFT of $N = 20$ points. . . . 141
2.38 A version of Figure 2.37 using a DFT of $N = 256$ points. . . . 142
2.39 A version of Figure 2.37 using freqz with 512 points representing only the positive part of the spectrum. . . . 142
2.40 Reproducing the graphs generated by freqz in Figure 2.39. . . . 143
2.41 DFT with $N = 8$ points of a signal sampled at $F_s = 100$ kHz. . . . 146

A.1 Q function for three different SNR ranges. . . . 159
A.2 The perpendicular line for obtaining the projection $\mathbf{p}_{xy}$ of a vector $\mathbf{x}$ onto $\mathbf{y}$ in $\mathbb{R}^2$. . . . 168
A.3 Projections of a vector $\mathbf{x}$ and $\mathbf{y}$ onto each other. . . . 169
A.4 Scatter plot of the input data and the basis functions obtained via PCA and Gram-Schmidt orthonormalization. . . . 177
A.5 Scatter plots of two-dimensional Gaussian vector $\mathbf{x}$ (represented by x) and PCA transformed vectors $\mathbf{y}$ (represented by +) . . . 177
A.6 Scatter plots of two-dimensional Gaussian vector $\mathbf{x}$ (x) and Gram-Schmidt transformed vectors $\mathbf{y}$ (+). . . . 178
A.7 PMF for a dice result. . . . 182
A.8 Obtaining probability from a pdf (density function) requires integrating over a range. . . . 183
A.9 Example of a finite-duration random signal. . . . 184
A.10 Example of five realizations of a discrete-time random process. . . . 185
A.11 Example of evaluating a random process at time instants $n = 4$ and $n = 6$, which correspond to the values of two random variables $\mathcal{X}[4]$ and $\mathcal{X}[6]$. . . . 186
A.12 Example of a joint pdf of the continuous random variables $\mathcal{X}[4]$ and $\mathcal{X}[6]$. 186
A.13 Correlation for data in matrix victories. . . . 188
A.14 Version of Figure A.13 using an image. . . . 189
A.15 Comparison between the 3-d representation of the autocorrelation matrix in Figure A.13 and the one using lags as in Eq. (A.59). . . . 190
A.16 Comparison between autocorrelation representations using images instead of 3-d graphs as in Figure A.15. . . . 190
A.17 Representation of a WSS autocorrelation matrix that depends only on the lag l. . . . 192
A.18 Suggested taxonomy of random processes. . . . 193
A.19 Correlation for random sequences with two equiprobable values: . . . 194
A.20 Alternative representation of the correlation values for the polar case of Figure A.19. . . . 195
A.21 One-dimensional ACF $R_X[l]$ for the data corresponding to Figure A.19 (unipolar and polar codes). . . . 196
A.22 ACF estimated using ergodicity and waveforms with 1,000 samples for each process. . . . 197
A.23 Functions used to characterize cyclostationary processes. . . . 200

A.24 Single realization $x[n]$ of Eq. (A.69) (top) and the ensemble variance over time (bottom plot), which has a period of $P/2 = 15$ samples. 201
A.25 Cyclostationary analysis of the modulated white Gaussian noise.. 202
A.26 Autocorrelation of the cyclostationary $m_u[n]$ polar signal **upsampled** by $L = 4$. 205
A.27 Autocorrelation of the cyclostationary $m_u[n]$ unipolar signal obtained with **upsampling** by $L = 4$. 206
A.28 Realizations of an upsampled polar signal ($L = 4$) with random initial sample. 207
A.29 ACF for the same polar process that generated the realizations in Figure A.28. 208
A.30 Three realizations of the random process corresponding to upsampling by 2 and randomly shifting a sinusoid of period $N = 4$ and amplitude $A = 4$. 209
A.31 Correlation of the WSS process corresponding to Figure A.30. 209
A.32 Autocorrelation matrices for the a) WSS process obtained by phase randomization of the b) WSC process. 210
A.33 Basic setup for measuring the insertion loss of a device under test (DUT). 221
A.34 PDF for a dice roll result when the random variable is assumed to be continuous. 224
A.35 A 3-d representation of the system $H(z) = 3 - 2z^{-1}$. 227

B.1 GNU Radio Companion interface highlighting the icons to execute the simulation. 230
B.2 USRP block diagram. 233
B.3 Block diagram describing a wireless system using GMSK. The blocks correspond to objects in Python that interface with GNU Radio code in C++ language. 234
B.4 Experimental setup using two USRPs for getting familiar with GNU Radio. 235
B.5 Screenshot of the FileViewer software. 237
B.6 Screenshot of the FileViewer dialog window that allows to convert between Linux and Windows text files. 238
B.7 Screenshot of the FileViewer software showing the result of converting the Windows file of Figure B.5 to the Linux format. 238
B.8 Interpretation of the file in Figure B.7 as short (2-bytes) big-endian elements. 239
B.9 Contents in hexadecimal of file floatsamples.bin generated with Listing B.1. 244
B.10 Interpretation as big-endian floats of file floatsamples.bin generated with Listing B.1. 244
B.11 Contents interpreted as floats of big-endian file htk_file.bin generated with HTK. 246

List of Tables

1.1 Notation used for continuous and discrete-time signals. 1
1.2 New signals $y_1[n] = x[n-2]$, $y_2[n] = x[-n+3]$, $y_3[n] = x[2n]$ and $y_4[n] = x[n^2]$, obtained by manipulating $x[n] = t^2(u[n-3] - u[n-7])$. . 4
1.3 Typical sampling frequencies. 11
1.4 The notation and values of the Nyquist frequency. 16
1.5 Examples of binary numbering schemes used as output codes in A/D conversion for $b = 3$ bits 22
1.6 Total energy E and average power $\mathcal{P}$ for two kinds of signal assuming an infinite time interval. 37
1.7 Autocorrelation functions and their respective equation numbers. 42
1.8 Example of autocorrelation for a real signal $[1, 2, 3]$ $(n = 0, 1, 2)$. 44
1.9 Example of calculating the unscaled autocorrelation for a complex-valued signal $[1 + j, 2, 3]$ $(n = 0, 1, 2)$, where $j = \sqrt{-1}$. 45

2.1 Examples of transforms and applications. 80
2.2 Examples of inner product definitions. 82
2.3 The four pair os equations for Fourier analysis with eternal sinusoids and the description of their spectra: c_k, $X(f)$ (or $X(\omega)$), $X[k]$ and $X(e^{j\Omega})$. . 96
2.4 Duality of periodicity and discreteness in Fourier analysis. 97
2.5 Units for each pair of Fourier equations in Table 2.3. 98
2.6 Summary of equations useful for signal processing with FFT. 110

A.1 Analogy between using the histogram and DFT for estimation, where $\hat{g}(x[n])$ is the estimated function and $\hat{f}(x[n]) = \kappa\hat{g}(x[n])$ its normalized version. The unit of $\hat{f}(x[n])$ is indicated within parentheses. 166

B.1 Examples of SDR Hardware Platforms. 233

2 Nomenclature of special frequencies. 249

Listings

1.1 MatlabOctaveCodeSnippets/snip_signals_sinusoid_generation.m 15
1.2 MatlabOctaveCodeSnippets/snip_signals_quantizer.m 18
1.3 MatlabOctaveFunctions/ak_quantizer.m 19
1.4 MatlabOctaveCodeSnippets/snip_signals_quantizer_use.m 20
1.5 MatlabOctaveCodeSnippets/snip_signals_data_precision.m 27
1.6 MatlabOctaveCodeSnippets/snip_signals_delta_calculation.m 28
1.7 MatlabOctaveCodeSnippets/snip_signals_numerical_error.m 28
1.8 MatlabOctaveCodeSnippets/snip_signals_single_precision.m 29
1.9 MatlabOctaveCodeSnippets/snip_signals_estimate_pdf.m 32
1.10 MatlabOctaveCodeSnippets/snip_signals_gaussian_rand_gen.m 32
1.11 MatlabOctaveCodeSnippets/snip_signals_unscaled_autocorrelation.m . 45
1.12 MatlabOctaveCodeSnippets/snip_signals_sinusoid_autocorrelation.m . 46
1.13 MatlabOctaveCodeSnippets/snip_signals_noisy_sinusoid.m 50
1.14 MatlabOctaveCodeSnippets/snip_signals_wavread.m 55
1.15 MatlabOnly/snip_signals_recordblocking.m 56
1.16 MatlabOnly/snip_signals_wavwrite.m 56
1.17 MatlabOnly/snip_signals_realtimeLoopback.m 56
1.18 MatlabOnly/snip_signals_digitize_signals.m 57
1.19 MatlabOnly/snip_signals_realtimeWithDspSystem.m 58
1.20 MatlabOctaveCodeSnippets/snip_signals_inpulse_train.m 60
1.21 MatlabOctaveCodeSnippets/snip_signals_amplitude_normalization.m . 62
1.22 MatlabOctaveCodeSnippets/snip_signals_timereversal.m 63
1.23 MatlabOctaveCodeSnippets/snip_signals_peak_detection.m 67
1.24 MatlabOctaveCodeSnippets/snip_signals_fundamental_frequency.m . . 68
1.25 MatlabOctaveCodeSnippets/snip_signals_cross_correlation.m 70
1.26 MatlabOctaveCodeSnippets/snip_signals_time_delay.m 71
1.27 MatlabOctaveCodeSnippets/snip_signals_time_aligment.m 72
1.28 MatlabOctaveCodeSnippets/snip_signals_2Drandom.m 75
1.29 MatlabOctaveCodeSnippets/snip_signals_sampling_inequality.m 79

2.1 MatlabOctaveCodeSnippets/snip_transforms_segmentation.m 84
2.2 MatlabOctaveFunctions/ak_dctmtx.m 86
2.3 MatlabOctaveCodeSnippets/snip_transforms_DTFS.m 104
2.4 MatlabOctaveCodeSnippets/snip_transforms_laplace_basis.m 112
2.5 MatlabOctaveCodeSnippets/snip_transforms_granschmidt_debug.m . 124
2.6 MatlabOnly/snip_transforms_ilaplace.m 126
2.7 MatlabBookFigures/figs_transforms_dctimagecoding 130

2.8 MatlabOctaveBookExamples/ex_transforms_check_orthogonality.m . . . 131
2.9 MatlabOctaveCodeSnippets/snip_transforms_fftshift.m . . . 134
2.10 MatlabOctaveCodeSnippets/snip_transforms_DTFS_sinusoid.m . . . 136
2.11 MatlabOctaveCodeSnippets/snip_transforms_DTFS_pulses.m . . . 139
2.12 MatlabOctaveCodeSnippets/snip_transforms_DTFT_pulse.m . . . 141
2.13 MatlabOctaveCodeSnippets/snip_transforms_freqz.m . . . 142
2.14 MatlabOctaveCodeSnippets/snip_transforms_KLT.m . . . 145

A.1 MatlabOctaveCodeSnippets/snip_transforms_projection.m . . . 169
A.2 MatlabOctaveCodeSnippets/snip_transforms_non_orthogonal_basis.m 171
A.3 MatlabOctaveCodeSnippets/snip_systems_pseudo_inverse.m . . . 172
A.4 MatlabOctaveFunctions/ak_gram_schmidt.m . . . 173
A.5 MatlabOctaveFunctions/ak_pcamtx.m . . . 175
A.6 MatlabOctaveFunctions/ak_correlationEnsemble.m . . . 187
A.7 MatlabOctaveFunctions/ak_convertToWSSCorrelation.m . . . 195
A.8 MatlabOctaveCodeSnippets/snip_appprobability_modulatednoise.m . . 201
A.9 MatlabOctaveCodeSnippets/snip_appprobability_cyclo_analysis_ensemble.m 202
A.10 MatlabOctaveCodeSnippets/snip_appprobability_cyclo_analysis.m . . 203
A.11 MatlabOctaveCodeSnippets/snip_appprobability_cyclo_spectrum_periodograms 204
A.12 MatlabOctaveCodeSnippets/snip_digi_comm_upsampled_autocorr.m . 206
A.13 MatlabOctaveCodeSnippets/snip_app_correlationEnsemble.m . . . 210

B.1 Java_Language/FileManipulation.java . . . 242

Preface

The information age brought new possibilities to improve the learning process, which can complement the traditional lectures and homeworks. With a clear impact in engineering, it is the era in which ITU and other standardization bodies opened access to many of their documents and there are several interesting open source and open hardware projects to help learning through practice. Hence, this book suggests several practical applications for which open source tools (Octave, GNU Radio, GSM projects, etc.) and low cost hardware (USRP, DVB-T dongle, HackRF peripherals, etc.) are key ingredients.

Current communication systems heavily rely on digital signal processing (DSP) and this motivated the effort to present both. This way the book can benefit from using the same nomenclature for both DSP and communications. But there is a well-known exploration versus exploitation tradeoff when defining the contents of a book, especially given that DSP and digital communications are broad areas. In fact, when I teach them, I feel like the guide of a group of tourists with seven days to visit Europe and low budget. This book is the result of my belief that, in spite of being impossible to visit all nice places, it is realistic to learn the basics of two very important engineering subjects and have fun along the process, especially by using resources of interesting open projects.

Because there are so many good and comprehensive textbooks on the subject, I took the route of leaving out topics that are often part of classical courses in favor of including topics that I observe being required when building modern systems such as software defined radios. The choices were biased by my experience in research and development projects with companies such as Ericsson, Brasilsat and Comunix, which substantially influenced my teaching. Hence, the book aims at self-taught readers with a bias towards practice. It does not claim to be a textbook or the ideal reading for those preparing for exams.

This book benefits from free and open source. Accordingly, software developed for the book was made available at the book web site `http://www.aldebaro.ufpa.br`. Even the figures can be reproduced by the reader with the provided source code. Both Mathwork's Matlab and Octave are capable of running most of the code. Matlab's object-oriented programming (OOP) and specific toolboxes that hide important details were avoided. The intention is to motivate the reader to understand and develop his/her own software, not to become familiar with a library or GUI. This strategy also allowed to make most of the code compliant with both Octave and Matlab. The scripts that run only on one of them were organized in specific directories.

Because URLs significantly change over time, instead of listing them in this printed copy, all references identified by [url], such as [urlFMitu] (a unique identifier following the prefix *url*), are organized (and kept updated) at the book web site.

I adopted a self publishing strategy that allows the whole book to be printed in full color with a relatively low list price. On the other hand, I performed several tasks that are typically taken care of by a specialized publisher.[1] It was not possible to have a professional proofreader reviewing the text and I apologize for any grammar error.

In 2008, the State Government of Pará, via its research agency, FAPESPA, sponsored my first book and got me started in this activity. I acknowledge here not only this support but all the work that the State Government did at that point in favor of science and technology in Pará, a Brazilian state surrounded by the magnificent Amazon, known for the world's largest tropical rain forest and river basin. The cover[2] of this book depicts this region and its needs for information and communications technologies, to move its economy from primary exploration of natural resources and establish new and green industries.

The current book was finalized at the University of California, San Diego (UCSD), during a sabbatical leave from the Federal University of Para (UFPA), Brazil, sponsored by the CAPES Foundation. Ten years after my Ph.D. graduation, I found myself again very grateful to my always advisor, Professor Alon Orlitsky, for hosting my stay as a visiting scholar at UCSD.[3]

I also acknowledge my students and colleagues at UFPA, researchers at Ericsson and Brasilsat, friends and family for their support. The book is dedicated to my parents, Aldebaro and Regina. Now that my spouse and I try to raise three kids, I better understand the wonderful parents I have. Their wisdom still amazes me when I recollect teachings and arguments. My father accomplished so many amazing things, some of strong impact to those living in Pará, and still managed to make his kids proud by winning sport matches (soccer, table tennis, etc.), coaching us, being our math teacher, our friend, etc., while my mother has always unconditionally supported me, with love so intense that it is visible. In the context, an example of her important presence is that when I was loosing interest for engineering, she gave me my first computer, which led me to ask "-How does this work?" and ended up defining my career. And to share my gratitude to God for letting me enjoy writing this book, I use a quote from St. Paul's 2nd Letter to the Corinthians: "Not that we are sufficient of ourselves to think any thing as of ourselves; but our sufficiency is of God".

[1] Regarding Latex typesetting, I was lucky to count with the expertise of Martin Sievers [urlFMlat].
[2] The book cover art is by Bernardo Magalhães with Libra Design [urlFMlib].
[3] I acknowledge the wonderful support from UCSD library staff, especially Ms. Deborah Kegel.

1 Analog and Digital Signals

1.1 To Learn in This Chapter

Some of the topics discussed in this chapter are:

- Distinguish digital, analog, discrete-time and sampled signals
- Represent signals using impulses and step functions
- Relate sampled and discrete-time signals via C/D and D/C conversions
- Models for the sampling and quantization processes
- Review binary number systems used in digital signal processing
- Calculate power, energy, fundamental period and correlation
- Use correlation for detecting periodicity and aligning signals
- Artificially generate signals (random and deterministic) for simulations
- Analyze signals using Matlab/Octave
- Use actual hardware to digitize signals
- Read/write digitized signals from/to binary files

Specific topics are organized as "Examples" along the text or "Applications" in the end of the chapter. These can be eventually skipped, but the reader is invited to stop just reading and explore these topics using a computer.

1.2 Analog and Digital Signals

In this text, a *signal* is a description of how an *amplitude* varies over time. Hence, it is useful to classify signals according to the behavior of these two variables: the independent variable representing progress in time and the amplitude, the dependent variable. If time evolution is represented by a real-valued variable $t \in \mathbb{R}$, the function $x(t)$ is called a *continuous-time signal*. If the progress over time is represented by an integer index $n \in \mathbb{Z}$, the sequence $x[n]$ is called a *discrete-time signal*.

Table 1.1: *Notation used for continuous and discrete-time signals.*

	Continuous-time, $t \in \mathbb{R}$	Discrete-time, $n \in \mathbb{Z}$
Quantized amplitude	$x_q(t)$	$x_q[n]$ (**digital**)
Not quantized	$x(t)$ (**analog**)	$x[n]$

Similarly, the amplitude can freely assume any real value or be restricted to only pre-specified values from a set $\mathscr{M}$. In the latter case, the amplitude is said to be *quantized*. The number M of elements in $\mathscr{M}$, i. e., the cardinality $M = |\mathscr{M}|$, indicates whether the system is binary ($M = 2$), quaternary ($M = 4$) or M-ary. A subscript q will be used to denote a quantized signal, such as in $x_q(t)$.

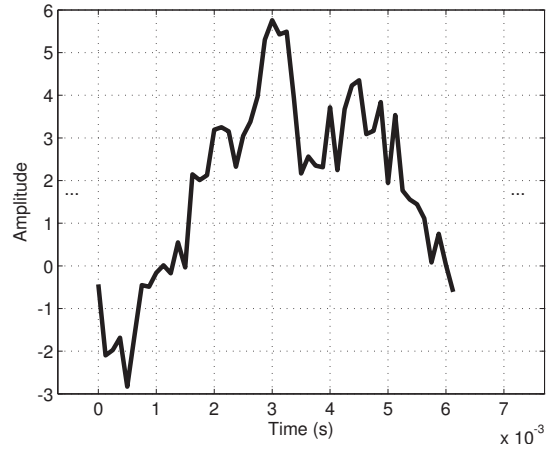


Figure 1.1: *Example of analog signal. Note the abscissa is continuous and the amplitude is not quantized.*

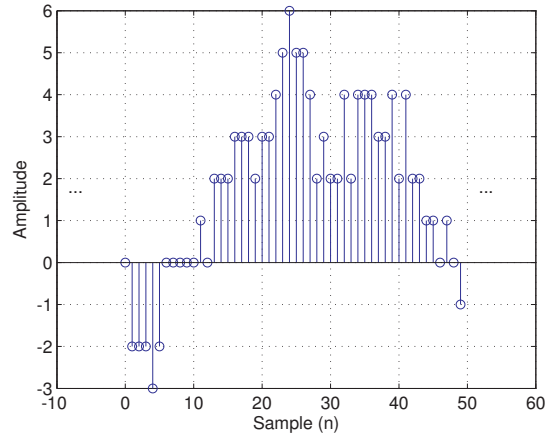


Figure 1.2: *Example of digital signal obtained by digitalizing the analog signal in Figure 1.1. Note both the abscissa (dimensionless indices) and amplitude of the digital signal are discrete. In this case, the quantized amplitudes assume values in* $\mathscr{M} = \{-3, -2, \ldots, 5, 6\}$.

Based on the previous definitions, it is possible to define *analog* and *digital* signals, which are the most common signals in practice. An analog signal $x(t)$ is a continuous-time signal that has real-valued amplitudes. A digital signal $x_q[n]$ is a discrete-time signal with quantized amplitudes. Table 1.1 summarizes a useful taxonomy of signals. Figure 1.1 and Figure 1.2 provide examples of analog and digital signals, respectively.

Signals that exist in the real-world are inherently analog. Even a measured DC power supply regulated to output 0 or 5 Volts will present a small random fluctuation due to circuit imperfections and noise. It could then be (strictly) classified as an analog signal $x(t)$. But given ranges specified by its circuits, a traditional digital system (e. g., a computer) can only process signals that can be characterized as digital signals. The

interfaces between digital systems and the analog world require analog to digital (A/D) and digital to analog (D/A) conversions.

It should be noted that the notation $x_q[n]$ (and $x(t)$) is ambiguous in the sense that it is widely used to represent both: a) the complete sequence and b) a sample at time n. In most scenarios both interpretations are valid because if someone provides a equation for a given sample, such as

$$x_q[n] = 3n \tag{1.1}$$

for example, which is valid for all $n \in \mathbb{Z}$, then it can be used to reconstruct the whole sequence and the notation $x_q[n] = 3n$ indicates such sequence as well its value specifically at "time" n.

1.3 Digitizing Signals

To use in practice (and effectively learn) digital signal processing techniques, it is important to be able to convert a real-world analog signal into a digital format, and then process it with a computer, microcontroller, FPGA or DSP chip, for example. A brief review of the A/D process is discussed in the sequel.

The ADC chip converts the input signal $x(t)$ into a vector of samples $x_q[n]$. The ADC executes two tasks: a) extract samples to accurately describe the signal and b) represent each of these samples with a reasonable accuracy. The first task depends on the adopted sampling frequency F_s, which is the number of samples extracted from the signal per time unit (more specifically, one second). For example, $F_s = 8,000$ Hz corresponds to obtaining 8,000 samples to represent each second of signal. The higher F_s, the more accurate the representation tends to be.

The second task depends on the number b of bits used to represent each sample. For example, if $b = 2$, each sample can be represented by only $N = 4$ distinct values: 00, 01, 10 and 11. The mapping between these binary values and amplitudes is somehow arbitrary. For example, 00 can represent -5 V while 01 can represent -100 V. In general, b bits allow the ADC to output $N = 2^b$ distinct values. ADCs with large values for F_s and b are more expensive. Sometimes the tradeoff is to use relatively large F_s with small b (e.g., 2.2 GHz with 10 bits) or vice-versa (e.g., 2.5 MHz with 24 bits).

Applications 1.1 to 1.5 suggest practicing the concepts of A/D conversion using a computer sound board. Before proceeding, it is recommended to try them and realize how these concepts are used via simple experiments.

1.4 Basic Signal Manipulation and Representation

It is important not only being knowledgeable on manipulating signals using software, but also algebraically, developing expressions and "theoretical" analysis. This sections discusses basic tools for representing and manipulating signals.

1.4.1 Manipulating the independent variable

Many signal processing tasks require manipulating t (for continuous-time signals) or n (for discrete-time). A convenient way to introduce this manipulation is by examples. Define the signal $x[n] = n^2$ for $n = 3, 4, 5$ and 6, and zero otherwise. This signal has only four non-zero sample values $[9, 16, 25, 36]$ at $n = 3, 4, 5$ and 6, respectively. The task here is to find new signals based on $x[n]$, namely: $y_1[n] = x[n-2]$, $y_2[n] = x[-n+3]$, $y_3[n] = x[2n]$ and $y_4[n] = x[n^2]$.

An interpretation of $y_1[n]$ is that it is a sequence of events that is related to the sequence $x[n]$ but with a time difference. For example, if $n = 6$ is interpreted as 6 o'clock and $x[6] = 36$ is the heart rate of an animal at that moment, the same measurement $y_1[8] = 36$ is found at $y_1[n]$ but two time units later (8 o'clock). Hence, the notation $y_1[n] = x[n-2]$ indicates that $x[n]$ and $y_1[n]$ present the same ordinate values, but these values occur two samples later in $y_1[n]$ when compared to $x[n]$.

One can create a mapping such as Table 1.2, which fills up the first column with values of n in the region of interest. Then, it is simple to get columns for $n-2$, $-n+3$, $2n$ and n^2.

Table 1.2: *New signals* $y_1[n] = x[n-2]$, $y_2[n] = x[-n+3]$, $y_3[n] = x[2n]$ *and* $y_4[n] = x[n^2]$, *obtained by manipulating* $x[n] = t^2(u[n-3] - u[n-7])$. *Columns* $y_1[n], y_2[n], y_3[n]$ *and* $y_4[n]$ *are filled up based on the auxiliary columns* $n-2$, $-n+3$, $2n$ *and* n^2, *but they correspond to amplitudes at the value of* n *given in column 1. For example, the second row indicates that* $y_1[-2] = 0, y_2[-2] = 25, y_3[-2] = 0$ *and* $y_4[-2] = 16$.

n	$x[n]$	$n-2$	$y_1[n]$	$-n+3$	$y_2[n]$	$2n$	$y_3[n]$	n^2	$y_4[n]$
-3	0	-5	0	6	36	-6	0	9	0
-2	0	-4	0	5	25	-4	0	4	16
-1	0	-3	0	4	16	-2	0	1	0
0	0	-2	0	3	9	0	0	0	0
1	0	-1	0	2	0	2	0	1	0
2	0	0	0	1	0	4	16	4	16
3	9	1	0	0	0	6	36	9	0
4	16	2	0	-1	0	8	0	16	0
5	25	3	9	-2	0	10	0	25	0
6	36	4	16	-3	0	12	0	36	0
7	0	5	25	-4	0	14	0	49	0
8	0	6	36	-5	0	16	0	64	0
9	0	7	0	-6	0	18	0	81	0

It may be convenient to memorize the *where-to-shift* rule that says (t is assumed, but the same applies to n):

- assuming that $t_0 > 0$, $x(t - t_0)$ is a delayed version (right-shifted) of $x(t)$ and $x(t + t_0)$ is an anticipated version (left-shifted) of $x(t)$.

However, other manipulations are trickier. For example, $x(-t+3)$ corresponds to delaying $x(-t)$ by 3 units of time instead of anticipating it. A common mistake is to think that $x(-t+3) = x(-(t-3))$, and start by delaying $x(t)$ by 3 and then flipping over the vertical axis. In summary, the where-to-shift rule is valid only for the two specific cases it was stated, while there are other important manipulations of the independent variable, such as $x(\alpha t+\beta)$, discussed in Example 1.1.

It should be noticed that almost all manipulations of t are equally applied to manipulating n of discrete-time signals, such as the where-to-shift rule. An exception, is when the equation that determines the independent variable of a discrete-time signal $x[n]$ leads to a non-integer number, which is invalid because $n \in \mathbb{Z}$. For example, $x[2n]$ behaves slightly different than $x(2t)$ given that the discrete-time signal is not defined for non-integer $n = \pm 0.5, \pm 1.5, \pm 2.5, \ldots$. Therefore, only the amplitude values of $x[n]$ for n even are present in $x[2n]$, while the values for n odd are discarded.

Example 1.1. **Simultaneously scaling and shifting a signal**. The signal $x(\alpha t+\beta)$, with $\alpha, \beta \in \mathbb{R}$, is commonly found in signal processing operations. It can be written as $x((t+\gamma)/\xi)$, with $\gamma, \xi \in \mathbb{R}$ and $\alpha = 1/\xi$ and $\beta = \gamma/\xi$.

For example, assume that $x(t) = u(t) - u(t-4)$ and the task is to obtain $y(t) = x(\alpha t+\beta)$ with $\alpha = 2$ and $\beta = 3$. In this case, for $t = -1.5$, $y(-1.5) = x(0)$ and carefully mapping the signals for other values of t, as in Table 1.2, leads to $y(t) = u(t+1.5) - u(t-0.5)$. Note that $\alpha = 2$ made the support of $y(t)$ to be half of the support of $x(t)$. But the time-shift was $\beta/\alpha = 1.5$, not 3. This indicates that $x(\alpha t+\beta)$ cannot be obtained by independently finding $x(\alpha t)$ and then shifting by β. If one wants to follow this strategy, the shift is by β/α.

In the case of $y(t) = x(-2t+3)$, the result is $y(t) = u(t+0.5) - u(t-1.5)$, which can be obtained by first finding $x(2t)$, then flipping it with respect to the y-axis (as in $x(-t)$) and then delaying the result by 3/2. As before, the support of $y(t)$ is half of the support of $x(t)$.

As the last example, consider the same $x(t) = u(t) - u(t-4)$, but $y(t) = x((t-4)/3)$, which corresponds to $\gamma = -4$ and $\xi = 3$ or, alternatively, $\alpha = 1/3$ and $\beta = -4/3$. In this case, $y(t) = u(t-4) - u(t-16)$, which has a support of 12, three times the support of $x(t)$, and a shift of $\beta/\alpha = -4$. □

1.4.2 Using impulses to represent signals

If you are not familiar with *impulses*, please first check Appendix A.29.

Recall that the notation $\delta(t-t_0)$ indicates that a continuous-time impulse occurs at time $t = t_0$. For example, $\delta(t-3)$ is a delayed impulse occurring at $t = 3$ and $\delta(t+2)$ is an anticipated impulse occurring at $t = -2$. A similar reasoning applies to the discrete-time $\delta[n-n_0]$.

A sampled or discrete-time signal can be conveniently represented by a sum of scaled and shifted impulses. To get the basic idea, consider the following example. An array $[3.0, -1.3, 2.1, 0, 4.2]$ stores the only non-zero samples of a signal $x[n]$. How would you write an expression for $x[n]$ without using impulses? A wrong guess would

be $x[n] = 3.0 - 1.3 + 2.1 + 0 + 4.2$ because that corresponds to having a constant DC value of $x[n] = 8, \forall n$. The impulse helps locating the desired amplitude in time. Considering the first sample occurs at $n = 0$, an expression for $x[n]$ using impulses is $x[n] = 3.0\delta[n] - 1.3\delta[n-1] + 2.1\delta[n-2] + 4.2\delta[n-4]$. Generalizing, any discrete-time signal can be written as

$$x[n] = \sum_{n_0=-\infty}^{\infty} x[n_0]\delta[n - n_0]. \tag{1.2}$$

In our example, $x[n_0]$ assumes the values $3.0, -1.3, 2.1, 0, 4.2$ for $n_0 = 0, \ldots, 4$, respectively. As explained, writing $x[n] = x[0] + x[1] + x[2] + x[3] + x[4]$ does not lead to the proper result because shifted impulses $\delta[n - n_0]$ are required at the proper locations.

To complement the example in Eq. (1.1), consider interpreting Eq. (1.2) as a single sample at a specific "time" $n = n_1$. To make that clear, one can write:

$$x[n_1] = \sum_{n_0=-\infty}^{\infty} x[n_0]\delta[n_1 - n_0] \tag{1.3}$$

and interpret that $\delta[n_1 - n_0] = 1$ if $n_1 = n_0$ and zero otherwise. Hence, the summation eliminates all samples of $x[n]$ other than $x[n_1]$, which is the amplitude at n_1. Both interpretations suggested by Eq. (1.2) and Eq. (1.3) are useful.

The continuous-time version of Eq. (1.2) is

$$x(t) = \int_{-\infty}^{\infty} x(\tau)\delta(\tau - t)d\tau,$$

which uses the *sifting* property of impulses (see Appendix A.29). This expression shows that, as for discrete-time signals, any continuous-time signal $x(t)$ can be represented as a composition of impulses.

1.4.3 Using step functions to represent signals

The continuous-time step function $u(t)$ is defined as 1 for $t > 0$ and 0 for $t < 0$. There is a discontinuity at $t = 0$ and the amplitude is conveniently assumed to be $u(t)|_{t=0} = 0.5$. The discrete-time version $u[n]$ does not have discontinuities and is defined as 1 for $n \geq 0$ and 0 for $n < 0$. The step function is very useful to indicate signals that are zero for negative values of the independent variable t or n. Another use is to describe finite-duration signals. More strictly, the *support* of a function $f(x)$ can be defined as the set of values of x for which $f(x) = 0$. When a signal ($x[n]$ or $x(t)$) has finite support, it is called a finite-duration signal.

It is easy to represent finite-duration signals using a programming language (Java, C, etc.) or an environment such as Matlab or Octave. Algebraically, the step function can be used. Say for example that the signal $x[n]$ has only four non-zero samples $[9, 16, 25, 36]$ at $n = 3, 4, 5$ and 6. This signal could be written as $x[n] = n^2(u[n-3] - u[n-7])$.

A side note is that $u(t)$ can simplify integrals by limiting the integration interval.

For example, $\int_{-\infty}^{\infty} x(t)dt$ when $x(t) = e^{-2t}u(t)$ can be obtained by

$$\int_{-\infty}^{\infty} e^{-2t}u(t)dt = \int_{0}^{\infty} e^{-2t}dt, \tag{1.4}$$

where the role played by $u(t)$ is absorbed by changing the inferior limit to 0. After this step, $u(t)$ can be eliminated from the integral. In other words, when $u(t)$ appears in integrals it can be often taken care of by simply adjusting the integral limits.[1]

1.5 Complex, Sampled and Discrete-time Signals

Some signals are useful for developing theoretical models, even if they do not exist in practice. This section describes three important representations for signals that are not physically realizable. Two of them are the *sampled* $x_s(t)$ and the *discrete-time* $x[n]$ signals, which are used as intermediate signals in stages of well-defined block diagrams that model sampling and quantization. Before discussing them, a brief motivation for using complex-valued signals is presented.

1.5.1 Complex-valued signals

The definition of a signal $x(t)$ (or $x[n]$) can be expanded to include complex-valued amplitudes $x(t) \in \mathbb{C}$. For example, complex-valued signals are widely using will be in digital communications to represent passband signals as will be discussed in Chapter 5. It should be observed that some existing physical channels are not able to transmit a complex analog signals $x(t)$, but many softwares adopt complex-valued digital signals, such as the DSP code of smartphones. One reason is that it is mathematically convenient to use a single complex-valued signal to represent two (typically related) real-valued (or simply real) signals.

1.5.2 Discrete-time signals

As mentioned, the analog to digital (A/D) conversion is typically performed by a single ADC chip. However it is convenient to model the A/D conversion by splitting it in two stages: *sampling* and *quantization*. One of the important reasons for adopting these two stages is that, while sampling is a linear operation, the input-output relation of quantizers is nonlinear (you may want take a peek at Figure 1.10 and see that the relation corresponds to a stairs function). There are several tools, such as the Laplace and Fourier transforms for dealing with linear operations. Working with nonlinear systems is more involved. Because of that, most "digital" signal processing theory does not assume the amplitude is quantized and, therefore, is called "discrete-time" signal processing.

[1] There is no need for using, e. g., the integration by parts of Eq. (A.24) to solve Eq. (1.4). This is a common mistake of DSP beginners.

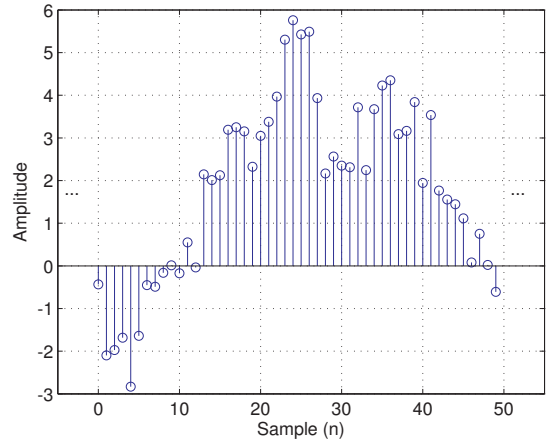


Figure 1.3: *Example of discrete-time signal. Note the abscissa is discrete, but the amplitude (ordinate) is not quantized.*

Figure 1.3 illustrates a discrete-time signal. Note the abscissa is $n \in \mathbb{N}$ as for a digital signal, but the amplitude can assume any value. This figure should be compared with Figure 1.1 and Figure 1.2, observing for example the unit of each axis.

1.5.3 Sampled signals

In a *sampled signal* $x_s(t)$, the information is represented by uniformly spaced impulses with distinct areas. Because the impulses have undefined amplitude values, $x_s(t)$ will not be considered here a continuous-time signal, but an intermediate representation between the continuous and discrete-time worlds as indicated in Figure 1.4. The sampled signal is very useful for theoretical modeling, as will be discussed in the next paragraphs.

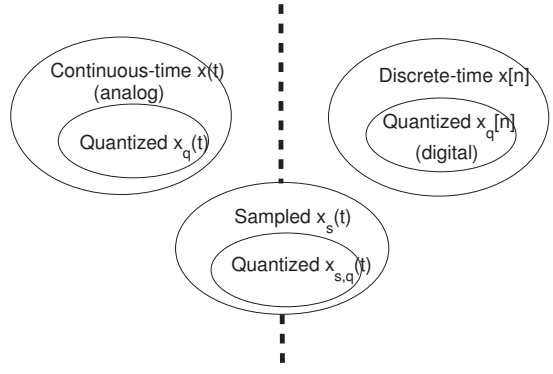


Figure 1.4: *Signals classification including the sampled signals.*

In this text, a sampled signal $x_s(t)$ is assumed to be the result of periodic sampling. Not all signals that contain impulses $\delta(t)$ will be denoted as $x_s(t)$. Hence, the sampled

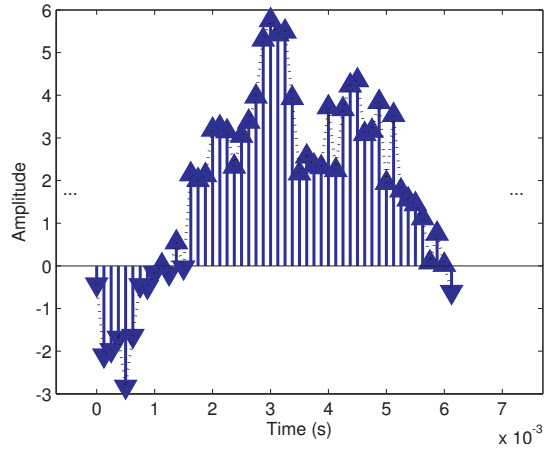


Figure 1.5: *Example of a sampled signal obtained from Figure 1.1. Note the abscissa is continuous but the amplitudes of the impulses go to infinite. Conventionally, the impulse height is used to indicate the impulse area and the contour created by the arrow heads resembles the original analog signal.*

signal is composed by periodically spaced impulses whose areas correspond to the value of the original analog signal $x(t)$ at the impulse location. There is a convention for showing plots of sampled signals with the impulse heights corresponding to their areas (see Appendix A.29), but the amplitude of $x_s(t)$ is $\pm\infty$ at the impulses positions. The amplitude is zero if t is not a multiple of T_s, i. e., $x(t) = 0, \forall t \neq kT_s$, where $k \in \mathbb{Z}$ is any integer.

Figure 1.5 presents an example of a sampled signal. It can be compared with Figure 1.1 and Figure 1.2. Figure 1.5 was obtained assuming the time interval between consecutive impulses is $T_s = 125$ μs (or, equivalently, that $F_s = 1/T_s = 8$ kHz). Note also that the sampled signal $x_s(t)$ in Figure 1.5 exists for all t, while a discrete-time signal $x[n]$ is considered to exist only for $n \in \mathbb{Z}$. Another distinction is that t in $x_s(t)$ (as well as in $x(t)$) has dimension of time (assumed to be in seconds) while n in $x[n]$ is dimensionless.

1.6 Modeling the Sampling Step

In *periodic* (or *uniform*) sampling of a continuous-time function $x(t)$ with a sampling interval T_s, the sampling frequency is

$$F_s = \frac{1}{T_s}. \tag{1.5}$$

For example, digital telephony adopts $F_s = 8000$ Hz, which corresponds to $T_s = 125$ μs.

Periodic sampling can be modeled as the multiplication of $x(t)$ by a periodic impulse train. Figure 1.6 depicts the infinite duration impulse train for sampling at 8 kHz. As conventionally done, the impulses heights indicates their areas.

The signal segment in Figure 1.1 has 6.125 ms of duration. Using $F_s = 8$ kHz

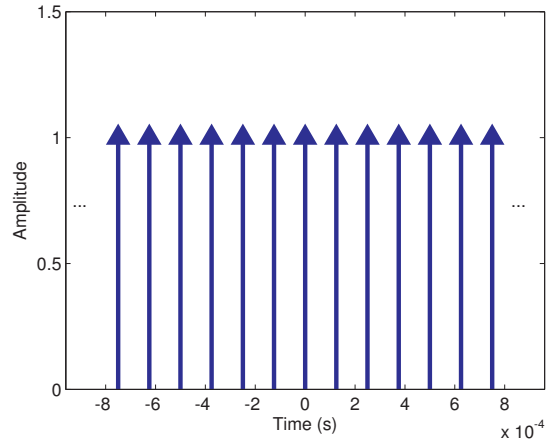


Figure 1.6: *An impulse train with unitary areas and* $T_s = 125\ \mu s$.

to sample this segment results in 50 samples. Multiplying the analog signal by the train of impulses and using the impulse sifting property leads to the sampled signal in Figure 1.5, which is depicted with the original analog signal superimposed.

Figure 1.5 shows that, in this case, the sampled signal $x_s(t)$ represents very well the original analog signal $x(t)$, in the sense that the envelope of the impulse areas correspond to a good match with respect to the original signal. This is not the case in the following example. Consider now that the sampling interval is increased by a factor of 4 to $T_s = 500\mu$s. Figure 1.7 indicates that in this case $x_s(t)$ cannot represent well, for example, the variation between the samples indicated in the figure.

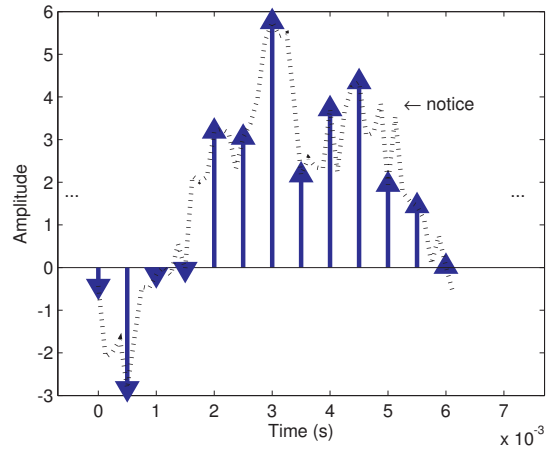


Figure 1.7: *Example of a sampled signal obtained with a sampling frequency smaller than the required for accurately representing the original signal (shown in dotted lines).*

In terms of computational cost, it is often convenient to use the smallest possible value of F_s. But choosing F_s too small may lead to a sampled signal that does not represent well the original signal as noted in Figure 1.7. The *sampling theorem* specifies

the minimum value that F_s must assume in order to be able to perfectly reconstruct $x(t)$ from $x_s(t)$. It is stated here without proof, which is deferred to Chapter 3.

Theorem 1. Sampling theorem. Assuming the maximum frequency of a real-valued signal $x(t)$ is $f_{\max}$, the sampling frequency must obey

$$F_s > 2f_{\max}, \tag{1.6}$$

in order to allow the perfect reconstruction of $x(t)$ from its sampled version $x_s(t)$ or the corresponding $x[n]$. If $f_{\max}$ is given in Hz, F_s is the number of real-valued samples per second, also given in Hz (alternatively, it can be denoted in samples per second, or SPS). □

In special cases, when $x(t)$ is complex-valued (Section **??**) or a "passband" signal (Section **??**), a lower value of F_s can be adopted and still allow perfect reconstruction.

The perfect reconstruction guaranteed by Eq. (1.6) assumes the existence of an ideal interpolation (or filtering) process, which is not realizable in practice. Hence, F_s is typically $2.5f_{\max}$ or even larger. The sampling frequency of some common signals[2] are shown in Table 1.3.

Table 1.3: *Typical sampling frequencies.*

Signal	$f_{\max}$	Explanation for $f_{\max}$	F_s
Telephone speech	3400 Hz	Band [300, 3400] Hz suffices for intelligibility	8 kHz
Audio (CD format)	20 kHz	Humans can hear up to ∼20 kHz	44.1 kHz
Electrocardiogram (ECG)	$<$ 250 Hz	Clinical studies	500 Hz

Sampling is so important that it will be further discussed later with the help of Fourier transforms in Section **??**. But before delving into a more rigorous mathematical description of sampling, it is useful to understand the expression of a periodic impulse train

$$p(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT_s),$$

where T_s is the sampling interval and $k \in \mathbb{Z}$. The expression for $p(t)$ and the sifting property of the impulse allows to model the sampled signal as

$$x_s(t) = x(t)p(t) = \sum_{k=-\infty}^{\infty} x(kT_s)\delta(t - kT_s). \tag{1.7}$$

Signals other than impulses (e. g., pulses) can be used for sampling, but an impulse train is the only option discussed here due to the mathematical convenience of Eq. (1.7).

[2] ECG is discussed, for example, at [url1ecg].

1.7 C/D and D/C conversions

According to the adopted convention, the following four signals appear during the A/D process along the stages of sampling (SAMPLING) and quantization (QUANTIZATION): analog, sampled, discrete-time and digital. The A/D process can be represented via the following block diagram:

$$x(t) \rightarrow \boxed{\text{SAMPLING}} \rightarrow x_s(t) \rightarrow \boxed{\text{C/D}} \rightarrow x[n] \rightarrow \boxed{\text{QUANTIZATION}} \rightarrow x_q[n],$$

where the sampled signals $x_s(t)$ is converted to the discrete-time signal $x[n]$ via the continuous-to-discrete (C/D) operation.

The C/D operation is a mathematical model of part of the sampling process that actually happens in the ADC and is always associated to a given sampling interval T_s.

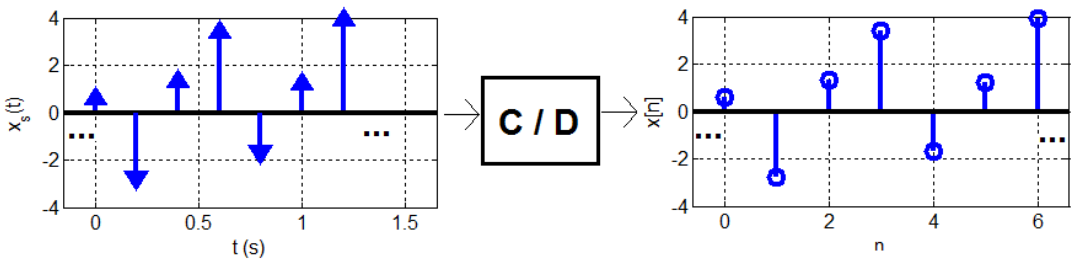


Figure 1.8: *Example of C/D conversion assuming* $T_s = 0.2$ *s.*

For example, if $x_s(t) = 3\delta(t+2T_s) - 1\delta(t+T_s) + 8\delta(t) - 5\delta(t-3T_s)$, the C/D conversion outputs $x[n] = 3\delta[n+2] - 1\delta[n+1] + 8\delta[n] - 5\delta[n-3]$. Recall that $x_s(t)$ exists for all values of t while $x[n]$ is a discrete time signal and $n \in \mathbb{Z}$. Another mathematical detail is that the non-zero amplitudes of $x_s(t)$ are $\pm\infty$, but the amplitudes of the corresponding $x[n]$ are well defined and equal to the area of the impulses in $x_s(t)$. As another example, Figure 1.8 illustrates a C/D conversion assuming $F_s = 0.2$ s.

Eventually, when mathematically manipulating signal expressions, the sampling operation may not be explicit, as discussed in the following example.

Example 1.2. **Simplified notation for the C/D conversion of a signal** $x(t)$. This example discusses a simplified notation for the C/D conversion that is sometimes adopted in the literature.

Assume the continuous-time signal $x(t) = 4e^{-2t}u(t)$ should be transformed to a discrete-time $x[n]$ with a sampling period T_s. This conversion is often denoted as:

$$x[n] = x(t)|_{t=nT_s} = 4e^{-2nT_s}u[n], \tag{1.8}$$

which can be confusing because $x(t)|_{t=nT_s} = 4e^{-2nT_s}u(nT_s)$ and the samples of the continuous-time step function $u(nT_s)$ became $u[n]$ (nT_s was "substituted" by n), while nT_s remained (was not substituted by n) in the exponential. The reason is that, when performing a C/D conversion, the occurrences of nT_s as part of the independent variable

(within $(\cdot)$) are converted to n, as depicted in Figure 1.8. However, the factor nT_s remains when it influences the amplitude (dependent variable).

The notation $x[n] = x(t)|_{t=nT_s}$ for the C/D process in Eq. (1.8) is somehow incomplete. It does not use impulses and, consequently, it does not make explicit the intermediate step of creating a sampled signal $x_s(t)$. Instead of Eq. (1.8), it is possible to describe this C/D process as

$$\begin{aligned} x[n] &= \text{C/D}\left\{x_s(t)\right\} \\ &= \text{C/D}\left\{\sum_{n=-\infty}^{\infty} x(nT_s)\delta(t-nT_s)\right\} \\ &= \text{C/D}\left\{\sum_{n=-\infty}^{\infty} 4e^{-2nT_s}u(nT_s)\delta(t-nT_s)\right\} \\ &= 4e^{-2nT_s}u[n], \end{aligned} \tag{1.9}$$

where C/D$\{\cdot\}$ denotes the C/D process. However, because this notation is cumbersome, the reader should be also familiar with the one adopted in Eq. (1.8). □

As the C/D is part of the A/D process, the D/C is included in the D/A, which is discussed in the sequel.

The D/A process is not the inverse, but somehow simpler than the A/D. The following signals are involved in a D/A model: digital, quantized sampled and analog. A D/C conversion is useful to model one stage of the D/A process as illustrated in the following block diagram:

$$x_q[n] \rightarrow \boxed{\text{D/C}} \rightarrow x_{s,q}(t) \rightarrow \boxed{\text{RECONSTRUCTION}} \rightarrow x(t).$$

The reconstruction is also called interpolation or filtering.

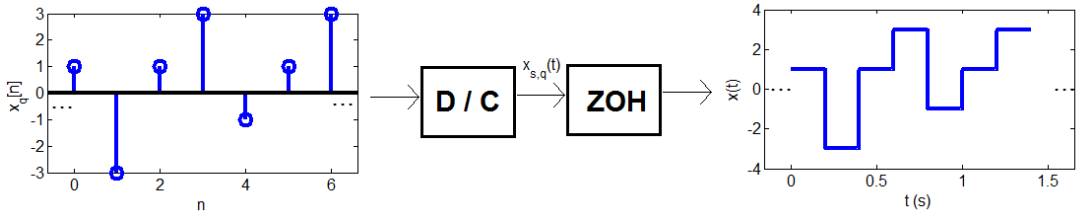


Figure 1.9: *Signal reconstruction with a ZOH.*

Among many alternatives, reconstruction can be implemented by the so-called *zero-order holder* (ZOH). As illustrated in Figure 1.9, the ZOH sustains the amplitude of $x[n_0]$, which is conveyed by the area of its corresponding impulse, during an interval of T_s seconds until the new sample $x[n_0+1]$ updates this amplitude and so on. Figure 1.9 assumes that the digital signal $x_q[n]$ has amplitudes from the set $\mathcal{M} = \{-3, -1, 1, 3\}$ and $T_s = 0.2$ s.

Depending on the reconstruction algorithm used in the D/A conversion, $x(t)$ may be limited to a finite set of amplitudes (quantized), but making this distinction is not important in the current context. For example, in Figure 1.9, the ZOH output could be denoted as $x_q(t)$, but more practical reconstruction schemes generate an analog signal with amplitudes not belonging to $\mathcal{M}$.

The reconstruction block will be discussed in more details in Section **??**, after Fourier analysis and linear systems are introduced.

1.8 Relating Continuous-time, Sampled and Discrete-time Signals

This section discusses relations among $x(t)$, $x_s(t)$ and $x[n]$ when periodic sampling is used. It is useful to observe the conversion of $x(t)$ into $x[n]$ via uniform sampling with sampling frequency of F_s with the intermediate step of creating $x_s(t)$. The goal is not to prove, but to motivate the *fundamental expression*

$$\omega = \Omega F_s, \tag{1.10}$$

where F_s is the sampling frequency assumed to be in Hertz, ω is the continuous-time angular frequency given in radians per second and Ω is the discrete-time angular frequency given in radians.

To understand Eq. (1.10), one can write the expression for a sampled signal $x_s(t)$ with $t = nT_s$, where $T_s = 1/F_s$ is the sampling interval, as exemplified in the sequel.

Example 1.3. **Fundamental expression relating continuous and discrete-time angular frequencies.** Assume $x(t) = 10\cos(24\pi t)$ and substitute $t = nT_s$ to create a sampled signal with the impulse sifting property as in Eq. (1.7):

$$x_s(t) = \sum_{n=-\infty}^{\infty} 10\cos(24\pi n T_s)\delta(t - nT_s). \tag{1.11}$$

In this case, the C/D conversion of Eq. (1.11) results in

$$x[n] = 10\cos(24\pi n T_s) = 10\cos\left(\frac{24\pi}{F_s}\, n\right).$$

The impulses in Eq. (1.11) were necessary to locate the sample values and also to inform that the amplitude of $x_s(t)$ is zero when $t \neq nT_s$. But after the C/D conversion, the continuous-time impulses are not necessary anymore. Note that the original angular frequency $\omega = 24\pi$ rad/s was converted to the angular frequency in discrete-time $\Omega = 24\pi/F_s$ rads.

For example, if $F_s = 36$ Hz in Example 1.3, the cosine with $\omega = 24\pi$ rad/s (that is equivalent to 12 Hz in this case) will be mapped to $\Omega = 2\pi/3$ rad. Looking in the other direction, e. g., an angle of $\Omega = \pi$ rad is always mapped to $\omega = \pi F_s$, which in this case is $\omega = 36\pi$ rad/s or, equivalently, 18 Hz. □

In fact, Eq. (1.10) is valid for all pairs of sampled and discrete-time signals that are related by a C/D or D/C conversion.

In essence, a continuous-time cosine $x(t) = \cos(\omega t)$ has radians per second (rad/s) as the unit of the angular frequency ω. Multiplying ω by t, which is given in seconds, leads to an angle in rad. In contrast, the unit of the angular frequency Ω of a discrete-time cosine $x[n] = \cos(\Omega n)$ is given in rad because n is dimensionless.

The different units of ω (rad/s) and Ω (rad) will play a fundamental role in discrete-time signal processing: ω can assume any value in the range $[-\infty, \infty]$ while functions in which Ω is an angle (the argument of cosines and sines) are typically evaluated only in a range of 2π, such as $[0, 2\pi[$. The reason is that any function of the angle Ω has a period of 2π. For example, $f(\Omega) = \cos(3\Omega)/\cos(5\Omega)$ is periodic because $f(\Omega) = f(\Omega + 2\pi), \forall\Omega$. Obviously, if Ω is not an angle, as in $f(\Omega) = (3+\Omega)/(5+\Omega)$ for example, the periodicity is not necessarily observed. It is a common practice to use the notation $f(e^{j\Omega})$ to indicate that f is a function of an angle Ω such as, for example, in Eq. (2.29).

Example 1.4. **Another example of conversion between discrete and continuous-time domains**. Assume a discrete-time signal $x[n] = 10\cos((2\pi/7)\ n)$. It is possible to indicate that its angular frequency is $\Omega = 2\pi/7$ rad because the signal repeats itself every $N = 7$ samples. However, because n is dimensionless, there is no information about time in this case. If it is stated that $x[n]$ was obtained via sampling at $F_s = 10$ Hz, $x[n]$ would be representing a cosine of angular frequency $\omega = 20\pi/7$ rad/s. If $F_s = 100$ Hz, then $\omega = 200\pi/7$ rad/s and so on. □

In summary, F_s in Eq. (1.10) plays the role of a normalization factor that relates continuous-time signals and their discrete-time counterparts obtained by sampling and C/D conversion.

Example 1.5. **Two approaches to simulate continuous-time signals**. Listing 1.1 indicates how to use the fundamental Eq. (1.10) to deal with discrete-time signals. The task is to generate a cosine of 600 Hz using Matlab/Octave. There are two fundamental approaches:

1. use "continuous-time" frequencies f (in Hz) or ω (in rad/s), with discretized time t in seconds, or
2. use "discrete-time" frequencies Ω (in rad), obtained via Eq. (1.10), and using a dimensionless "time" n.

Compare the two approaches in Listing 1.1.

Listing 1.1: *MatlabOctaveCodeSnippets/snip_signals_sinusoid_generation.m*

```
Fs=8000; %sampling frequency (Hz)
Ts=1/Fs; %sampling interval (seconds)
N=20000; %number of desired samples
f0=600; %cosine frequency (Hz)
%%%%First alternative to generate a cosine of 600 Hz
```

```
t=0:Ts:(N-1)*Ts; %discretized continuous-time axis (sec.)
x1=5*cos(2*pi*f0*t); %amplitude=5 V and frequency = f0 Hz
%%%%Second alternative: work directly in discrete-time
w0=2*pi*f0*Ts; %w0 is in rad, convert from rad/s to rad
n=0:N-1; %discrete-time axis (do not use Ts anymore)
x2=5*cos(w0*n); %amplitude=5 V and frequency = w0 rad
plot(x1-x2); %plot error between two alternative sequences
soundsc(x1,Fs) %for fun: playback one of them to listen
```

Note that the sequences x1 and x2 are essentially the same but there are (small) numerical errors. □

1.8.1 Nyquist frequency

As indicated by Eq. (1.6), if the sampling theorem is obeyed, the maximum frequency in the original continuous-time signal $x(t)$ is restricted to $f_{\max} < F_s/2$ Hz, where $F_s/2$ is called the *Nyquist frequency*.

Using Eq. (1.10), one can see that angle $\Omega = \pi$ rad will be mapped into $\omega = \Omega F_s = \pi F_s$ rad/s, which corresponds to the Nyquist frequency $F_s/2$ Hz. This is consistent with the fact that π represents the highest frequency in discrete-time processing.

1.8.2 Frequency normalization in Matlab/Octave

It is sometimes inconvenient to show graphs using Ω in rad as the abscissa. For example, a graph in the range $[0, 2\pi[$ could distract the user with values such as 6.28 approximating 2π. This is one of the reasons behind the convention adopted by Matlab/Octave to use, instead of Ω in rad, a normalized frequency

$$f_N = \Omega/\pi. \tag{1.12}$$

The division by π maps the range $[0, \pi]$ rad to $[0, 1]$.

From Eq. (1.10) and having F_s as sampling frequency, a given frequency f in Hz is mapped in Matlab/Octave into

$$f_N = \frac{f}{F_s/2}, \tag{1.13}$$

i. e., f is divided by the Nyquist frequency. For example, if $F_s = 10$ Hz, a frequency f=4 Hz after the sampling process would be represented as $f_N = 4/5$ in Matlab/Octave.

Table 1.4: *The notation and values of the Nyquist frequency.*

f (Hz)	ω (rad/s)	Ω (rad)	Matlab/Octave normalized (Ω/π)
$F_s/2$	πF_s	π	1

Table 1.4 summarizes the discussed information about the Nyquist frequency and its normalized version in Matlab/Octave.

1.9 An Introduction to Quantization

Similar to sampling, quantization is very important, for example, because computers and other digital systems use a limited number b of bits to represent numbers. In order to convert an analog signal to be processed by a computer, it is therefore necessary to quantize its sampled values.

A quantizer Q is a mapping $Q : \mathbb{R} \to \mathscr{M}$ from a real number $x[n] \in \mathbb{R}$ to an element $x_q[n]$ of a finite set $\mathscr{M}$ as represented pictorially by:

$$x[n] \rightarrow \boxed{\text{Q}} \rightarrow x_q[n] \in \mathscr{M}.$$

The cardinality of this set is $|\mathscr{M}| = M$ and typically $M = 2^b$, where b is the number of bits used to represent each output $x_q[n]$. The quantization error is

$$e_q[n] \triangleq x[n] - x_q[n].$$

The higher b, the more accurate the representation tends to be.

1.9.1 Uniform quantization

Unless otherwise stated, this text assumes *uniform* quantizers with a *quantization step* Δ. When the quantization is uniform, all the steps in the quantizer "stairs" have both height and width equal to Δ. In this case, the quantizer will be defined by only two numbers: Δ and $\hat{X}_{\min}$, where $\hat{X}_{\min}$ is the minimum value of the quantizer output $x_q[n] \in \mathcal{M}$, where

$$\mathcal{M} = \{\hat{X}_{\min}, \hat{X}_{\min} + \Delta, \hat{X}_{\min} + 2\Delta, \ldots, \hat{X}_{\min} + (M-1)\Delta\}. \tag{1.14}$$

The value of Δ is also called the least significant bit (LSB) of the ADC, because it represents the value (e. g., in Volts) that corresponds to a variation of a single bit. This is also emphasized in Eq. (1.17) in the context of modeling the representation of real numbers in fixed-point as a quantization process.

In order to design a quantizer, it is important to estimate its input dynamic range $[X_{\min}, X_{\max}]$, where $X_{\min}$ and $X_{\max}$ are the minimum and maximum amplitude values assumed by $x[n]$.

There are several strategies for designing a quantizer. The optimum quantizer, which minimizes the quantization error, must be designed according to the input signal statistics (more specifically, the probability density function of $x[n]$) as in the Lloyd's algorithm.[3]

Alternatively, suboptimal strategies take into account only the input dynamic range $[X_{\min}, X_{\max}]$, where $X_{\min}$ and $X_{\max}$ are the minimum and maximum amplitude values assumed by $x[n]$. Among these strategies, a very simple one is to choose $\hat{X}_{\min} = X_{\min}$

[3] Lloyd's paper is listed at [url1qua].

and

$$\Delta = \left| \frac{X_{\max} - X_{\min}}{M} \right|.$$

In this case, the minimum quantizer output is $\hat{X}_{\min} = X_{\min}$, but the maximum value does not reach $X_{\max}$ and is given by $\hat{X}_{\max} = X_{\max} - \Delta$. The reason is that, as indicated in Eq. (1.14), $\hat{X}_{\max} = \hat{X}_{\min} + (M-1)\Delta$. This can be fixed by adopting

$$\Delta = \left| \frac{X_{\max} - X_{\min}}{M-1} \right|, \tag{1.15}$$

which leads to $\hat{X}_{\max} = X_{\max}$. For example, if the quantizer has $b = 2$ bits and it is assumed $X_{\min} = -1$ V and $X_{\max} = 3$ V, Eq. (1.15) leads to $\Delta = (3-(-1))/(4-1) \approx 1.33$ V and the quantizer output levels are $\mathcal{M} = \{-1, 0.33, 1.66, 3\}$.

One common requirement for a quantizer is to reserve one quantization level to represent zero (otherwise, it would output a non-zero value even with no input signal). The levels provided by Eq. (1.15) can be refined by counting the number M_{neg} of levels representing negative numbers and adjusting $\hat{X}_{\min}$ such that $\hat{X}_{\min} + \Delta M_{\text{neg}} = 0$. Listing 1.2 illustrates the procedure.

Listing 1.2: *MatlabOctaveCodeSnippets/snip_signals_quantizer.m*

```
Xmin=-1; Xmax=3; %adopted minimum and maximum values
b=2; %number of bits of the quantizer
M=2^b; %number of quantization levels
delta=abs((Xmax-Xmin)/(M-1)); %quantization step
QuantizerLevels=Xmin + (0:M-1)*delta %output values
isZeroRepresented = find(QuantizerLevels==0); %is 0 there?
if isempty(isZeroRepresented) %zero is not represented yet
    Mneg=sum(QuantizerLevels<0); %number of negative
    Xmin = -Mneg*delta; %update the minimum value
    NewQuantizerLevels = Xmin + (0:M-1)*delta %new values
end
```

For the given example, the original set $\mathcal{M} = \{-1, 0.33, 1.66, 3\}$ is converted to $\mathcal{M} = \{-1.33, 0, 1.33, 2.67\}$. Notice the new value of $\hat{X}_{\max}$ is $X_{\max} - \Delta$.

If it can be further assumed that $X_{\max} = |X_{\min}|$ (bipolar $x[n]$), Eq. (1.15) simplifies to

$$\Delta = \frac{2X_{\max}}{M-1}.$$

In this case, one quantization level can be reserved to represent zero, while $M/2$ and $M/2-1$ levels represent negative and positive values, respectively. Most ADCs adopt this division of quantization levels when operating with bipolar inputs. For example, an 8-bits ADC can output signed integers from -128 to 127.

Because *outliers* (a sample numerically distant from the rest of the data, typically a noisy sample) can influence too much a design strictly based on $X_{\min}$ and $X_{\max}$, a better approach is to observe the *histogram* (see Figure 1.17, for an histogram example) of the quantizer input and pick reasonable values to use in Eq. (1.15). For example, if

the input data follows a Gaussian distribution, the sample mean μ and variance σ^2 can be estimated and the dynamic range assumed to be $X_{\min} = \mu - 3\sigma$ and $X_{\max} = \mu + 3\sigma$ to have the quantizer covering approximately 99% of the samples.

1.9.2 Granular and overload regions

Figure 1.10 depicts the input/output relation of a 3-bits quantizer with $\Delta = 1$. There are three regions of operation of a quantizer: the granular and two overload (or saturation) regions. An overload region is reached when the input $x[n]$ falls outside the quantizer output *dynamic range*. The granular is between the two overload regions. In Figure 1.10, because $\Delta = 1$, operation in the granular region corresponds to the rounding operation round to the nearest integer. For example, round(2.4)=2 and round(2.7)=3. When $\Delta \neq 1$, the quantization still corresponds to rounding to the nearest $\hat{X} \in \mathscr{M}$, but $\hat{X}$ is not restricted to be an integer anymore.

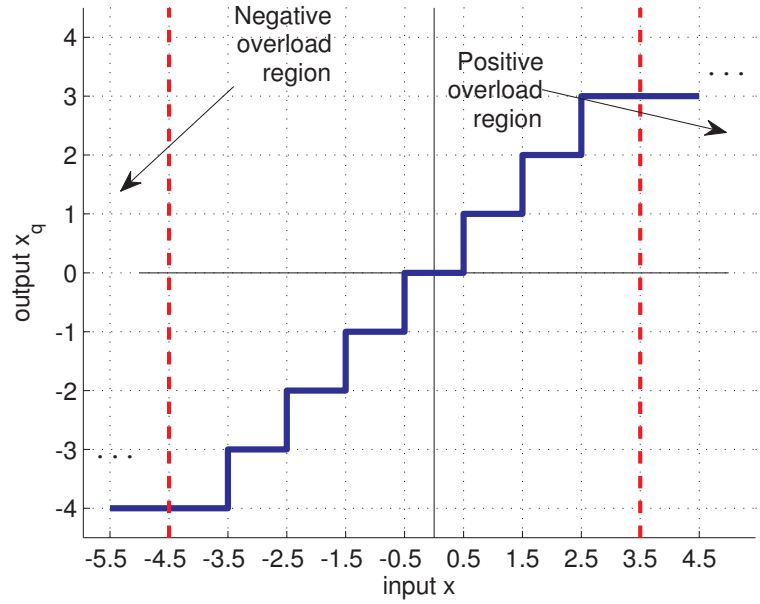


Figure 1.10: *Input/output relation of a 3-bits quantizer with* $\Delta = 1$.

Figure 1.11 depicts the input/output relation of a 3-bits quantizer with $\Delta = 0.5$. Close inspection of Figures 1.10 and 1.11 shows that the error $e_q[n]$ is in the range $[-\Delta/2, \Delta/2]$ within the granular region, but can grow indefinitely when the input falls in one of the overload regions.

Listing 1.3 illustrates a Matlab/Octave function that implements a conventional quantizer with $\hat{X}_{\min} = -2^{b-1}\Delta$ and saturation.

Listing 1.3: *MatlabOctaveFunctions/ak_quantizer.m*

```
function [x_q,x_i]=ak_quantizer(x,delta,b)
% function [x_q,x_i]=ak_quantizer(x,delta,b)
```

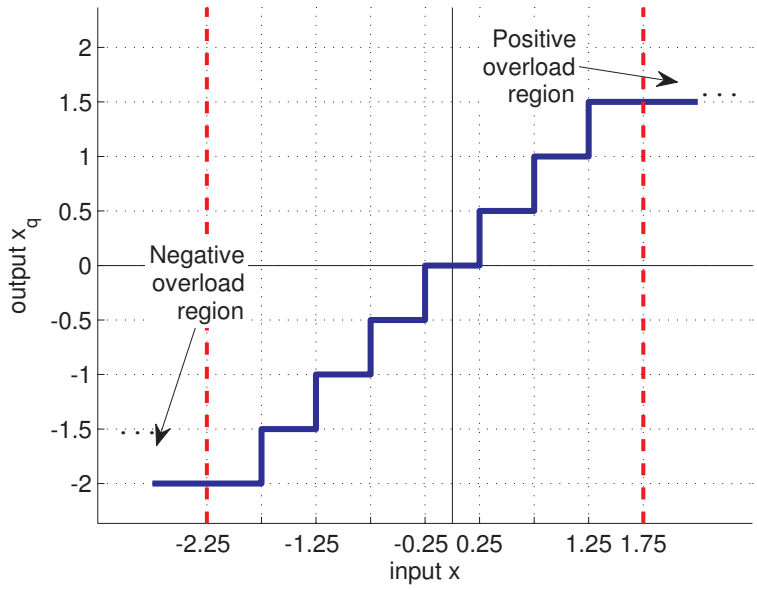


Figure 1.11: *Input/output relation of a 3-bits quantizer with* $\Delta = 0.5$.

```
%This function assumes the quantizer allocates 2^(b-1) levels to
%negative output values, one level to the "zero" and 2^(b-1)-1 to
%positive values. See ak_quantizer2.m for more flexible allocation.
x_i = x / delta; %quantizer levels
x_i = round(x_i); %nearest integer
indices=find(x_i > 2^(b-1) - 1);
x_i(indices) = 2^(b-1) - 1; %impose maximum
indices=find(x_i < -2^(b-1));
x_i(indices) = -2^(b-1); %impose minimum
x_q = x_i * delta;  %quantized and decoded output
```

The commands in Listing 1.4 can generate a quantizer input-output graph such as Figure 1.11 using the function ak_quantizer.m.

Listing 1.4: *MatlabOctaveCodeSnippets/snip_signals_quantizer_use.m*

```
delta=0.5; %quantization step
b=3; %number of bits
x=[-5:.01:4]; %define input dynamic range
[x_hat, passos] = ak_quantizer(x,delta,b); %quantize
plot(x,x_hat), grid %generate graph
```

1.9.3 Quantizer output: Real or binary number

The A/D conversion to obtain the digital signal $x_q[n]$ that represents $x(t)$ can be modeled as

$$x(t) \rightarrow \boxed{\text{SAMPLING}} \rightarrow x_s(t) \rightarrow \boxed{\text{C/D}} \rightarrow x[n] \rightarrow \boxed{Q} \rightarrow x_q[n].$$

Example 1.6. **Quantization example**. For example, let $x(t_0) = 12804.6542$ Volts be the amplitude of $x(t)$ at $t = t_0$, such that $x[n_0] = 12804.6542$, where n_0 is the value of n corresponding to $t = n_0 T_s$. The value of $x[n_0]$ quantized with $b = 16$ bits is assumed to be $x_q[n_0] = Q\{x[n_0]\} = 12804.5$ (which was obtained with Matlab's fi(12804.6542) command). □

It is often convenient to have the quantizer output $x_q[n]$ being represented by *real* numbers, as in Example 1.6, because it can be easily manipulated. However, a *binary* number $x_b[n]$ better represents the actual output of an ADC. There is often a relatively simple mapping from $x_b[n]$ to $x_q[n]$ called *decoding*. But we have a decision to make with respect to nomenclature: should we assume a quantizer outputs a binary codeword or its corresponding real number? The convention adopted here is to assume that a quantizer outputs either binary or real numbers, but represent the first case using the notation $\tilde{Q}_b$ and reserve Q for the latter case, as Figure 1.12 illustrates.

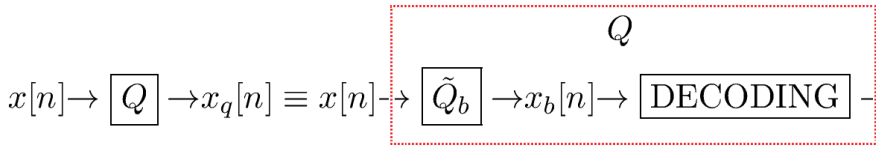


Figure 1.12: *A quantizer Q that outputs real numbers can be seen as containing a quantizer $\tilde{Q}_b$ that outputs binary numbers followed by a decoder.*

Example 1.7. **Distinguishing the alternative outputs of a quantizer**. In Example 1.6, where $Q\{x[n_0]\} = 12804.5$, it can be assumed that an ADC quantizer $\tilde{Q}_b$ outputs $x_b[n_0] = \tilde{Q}\{x[n_0]\} = 0110010000001001$, which in this case is $x_d[n_0] = 25609$ in decimal. The decoding then generates $x_q[n_0]$ by multiplying $x_b[n_0]$ (or, equivalently, $x_d[n_0]$) by a scaling factor $\Delta = 0.5$, which corresponds to

$$x_q[n_0] = x_d[n_0] \times \Delta = 25609 \times 0.5 = 12804.5.$$

Distinguishing $x_d[n_0]$ and $x_b[n_0]$ may seem an abuse of notation because in this case they are the same number described in decimal and binary numeral systems, respectively. However, the notation is useful because there are many distinct ways of mapping $x_b[n_0]$ to $x_d[n_0]$ and vice-versa. □

In fact, when considering the decoding process, there are various binary numeral systems that differ specially on how negative numbers are represented. Some options are briefly discussed in the sequel.[4]

1.9.4 Coding and decoding for quantization

The decoding mapping $x_b \to x_q$ in Figure 1.12 can be arbitrary, but in most cases very simple schemes are adopted.

[4] Binary numeral systems are also discussed at [url1bin].

For example, a common simplifying assumption is that x_q can be written as $x_q = x_i\Delta$, where $x_i \in \mathbb{Z}$ is an integer, and the normalization by the step size $\Delta > 0$ is the second stage of decoding:

$$x_b[n] \rightarrow \boxed{\text{DECODING (stage 1)}} \rightarrow x_i[n] \rightarrow \boxed{\times\ \Delta} \rightarrow x_q[n]. \tag{1.16}$$

The *step size* Δ indicates the variation (e.g., in Volts) corresponding to a variation of one in the integer value.

In many digital signal processing applications, the value of Δ is not important and it is implicitly assumed $\Delta = 1$. In this case, the digital signal $x_q[n]$ is effectively represented by the integers $x_i[n]$.

For example, considering Example 1.7 and an original sample value in analog domain $x(t_0) = 12804.5$ Volts, it may be represented and processed in digital domain as $x_q[n_0] = x_d[n_0] = 25609$. This corresponds to an implicit assumption that $\Delta = 1$ instead of the actual $\Delta = 0.5$ of Example 1.7. If at some point it becomes necessary to relate $x(t)$ and $x_q[n]$, such as when the average power in both domains should match, the scaling factor Δ could then be used.

Concentrating now on the first stage of Block (1.16), where the actual decoding occurs, Table 1.5 provides an example of practical codes. For example, note in Table 1.5 that $x_b = 100$ corresponds to 7 in Gray code and -4 in two's complement.

Table 1.5: *Examples of binary numbering schemes used as output codes in A/D conversion for $b = 3$ bits. The first column indicates values for x_b while the others indicate x_i values.*

Binary number	Unsigned integer	Two's complement	Offset	Sign and magnitude	Gray code
000	0	0	−4	0	0
001	1	1	−3	1	1
010	2	2	−2	2	3
011	3	3	−1	3	2
100	4	−4	0	"−0"	7
101	5	−3	1	−1	6
110	6	−2	2	−2	4
111	7	−1	3	−3	5

Three of the codes exemplified in Table 1.5 will be discussed in more details because they are very important from a practical point of view: *unsigned integer*, *offset* and *two's complement* output coding.

The unsigned representation, sometimes called *standard-positive binary coding*, represents numbers in the range $[0, 2^b - 1]$. It is simple and convenient for arithmetic operations such as addition and subtraction. However, this representation cannot represent negative values. An alternative is the offset representation, which is capable of representing numbers in the range $[-2^{b-1}, 2^{b-1} - 1]$. This code can be obtained by

subtracting the "offset" 2^{b-1} from the unsigned representation. The two's complement is a popular alternative that is also capable of representing numbers in the range $[-2^{b-1}, 2^{b-1} - 1]$ and allows for easier arithmetics than the offset representation, as studied in textbooks of digital electronics. Note that the offset representation can be obtained by inverting the most significant bit (MSB) of the two's complement. For this reason it is sometimes called *offset two's complement*.

Representing a negative number in two's complement one can first represent the absolute value in standard-positive binary coding, then invert all bits and sum 1 to the result. For example, assuming -81 should be represented with $b = 8$ bits, the first step would lead to 01010001. Inverting the bits leads to 10101110 and summing to 1 gives the final result 10101111. Note that two's complement requires sign extension (see, e. g., [url1two]) by extending the MSB. For example, if -81 were to be represented with $b = 16$ bits, the correct codeword would be 1111111110101111, 0000000010101111.

Table 1.5 covers the mappings (or quantizers) adopted in most ADCs and fall within a class of number representation called *fixed-point*. But the representation of real numbers within computers typically use a more flexible representation belonging to another class called *floating-point*. Even if there is no ADC involved and the operation is performed in the "digital domain", representing a real number with b bits can be modeled as a quantizer in the same way as discussed in the context of A/D conversion.

1.9.5 Representing numbers in fixed-point

Both fixed and floating-point representations are capable of representing real numbers. A real number can be written as the sum of an integer part and a fractional part that is less than one and non-negative.

The name "fixed-point" is due to the fact that the b bits available to represent a number are organized as follows:

- 1 bit representing the sign (positive or negative) of the integer part and, consequently, of the number itself
- b_i bits representing the integer part of the number
- b_f bits representing the fractional part of the number

such that $b = b_i + b_f + 1$ bits. For simplicity, the sign bit is assumed here to be the MSB, but in practice that depends on the adopted standard and the machine endianness (see Appendix B.5.3 for a brief explanation of big and little endian formats).

One way of thinking the "point" is that there is flexibility on choosing the power of 2 that is associated to the second bit, at the right of the MSB. After this design choice is made, a fictitious point separates the bits corresponding to the integer part from the fractional part bits. There is a convention of denoting the values chosen for b_i and b_f as $Qb_i.b_f$.

Figure 1.13 shows an example of fixed-point representation using Q3.4, for which $b_i = 3$, $b_f = 4$ and, consequently $b = 8$. In this case, the point is located between the fourth and fifth bits. The binary codeword $x_b = 01010001$ is mapped to a real number

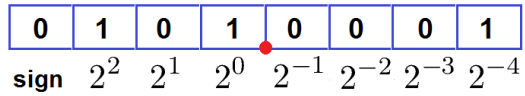


Figure 1.13: *Fixed-point representation Q3.4 of the real number 5.0625. The "point" is always after the bit corresponding to the weight* 2^0.

according to the position of the point:

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 5.0625.$$

It is very common to normalize the numbers to make them fit the range $[-1, 1[$. In this case b_i can be zero, $b_f = b - 1$ and the second bit has a weight in base-2 equals to 2^{-1} such that the point is between the MSB and this bit. The other bit weights are then $2^{-2}, \ldots, 2^{-b_f}$, according to their respective position. For example, the outputs of a quantizer with $b = 2$ bits when represented in the format Q0.1 ($b_i = 0$, $b_f = 1$) assume four possible outputs $x_q \in \{-1, -0.5, 0, 0.5\}$. In general, the weight 2^{-b_f} of the LSB corresponds to the value of the step size, i. e.

$$\Delta = 2^{-b_f}. \tag{1.17}$$

Note that the number 5.0625 in Figure 1.13 can also be obtained by identifying that x_b corresponds to $x_d = 81$ and calculating $x_q = x_i\Delta = 81 \times 2^{-4} = 5.0625$. If the point were moved to the position between the MSB and second bit, which corresponds to choosing Q0.7, the quantizer would have $\Delta = 2^{-7}$ and the same codeword $x_b = 01010001$ would be mapped to $x_q = 0.6328125$.

In summary, when a fixed-point representation is assumed, the quantizer has step $\Delta = 2^{-b_f}$ and can output numbers in the range $[-2^{b_i}, 2^{b_i} - \Delta]$. Algorithm 1 describes the process for representing a real number x in fixed-point using $Qb_i.b_f$.

For example, using Algorithm 1 to represent $x = -7.45$ in Q3.4, the first step is to calculate $x/\Delta = -7.45 \times 2^4 = -119.2$. Then, using round leads to $x_i = -119$, which corresponds to $x_q = -7.4375$ and $x_b = 10001001$ in two's complement. This algorithm is the same implemented in Listing 1.3 but explicitly outputs the binary codeword x_b.

Choosing the point position depends on the dynamic range of the numbers that will be represented and processed. This is typically done with the help of simulations and is a challenging task due to the tradeoff between range (the larger b_i the better) and precision (the larger b_f the better). In sophisticated algorithms, it is typically required to assume distinct positions for the point at different stages of the process. In such cases, using specialized tools such as Mathwork's Fixed-Point Designer [url1mfi] can significantly help.

Example 1.8. **Fixed-point in Matlab and Octave**. Both Matlab and Octave have toolboxes for fixed-point arithmetic. But, unfortunately, they are not compatible. Matlab's Fixed-Point Toolbox is more sophisticated and better documented. If you have

Algorithm 1: Fixed-point conversion.

Input: x, b_i, b_f // value to quantize, integer and fractional # bits
Output: x_b, x_i, x_q // binary, integer and decoded forms of $Q\{x\}$

$x_i = x/\Delta$ // Equivalent to multiplying by $1/\Delta = 2^{b_f}$
if x_i *is not an integer* **then**
    round x_i to the nearest integer // floor is sometimes used instead
if $x_i < -2^{b_i}$ **then**
    $x_i = -2^{b_i}$ // Implement *saturation* for negative values
    Print: *Warning!* Consider increasing number b_i of integer bits
if $x_i > 2^{b_i} - \Delta$ **then**
    $x_i = 2^{b_i} - \Delta$ // Implement *saturation* for positive values
    Print: *Warning!* Consider increasing number b_i of integer bits
$x_q = x_i \times \Delta$ // Multiply by Δ to find decoded value
if $x_i < 0$ **then**
    Obtain x_b using two's complement // with b bits and sign extension
else
    Simply obtain x_b as the standard-positive binary coding of x_i

the Matlab's toolbox, try for example doc fi. The Matlab command for representing $x = -7.45$ as in the previous example is:

```
x=-7.45;signed=1;b=8;bf=4;x_q=fi(x,signed,b,bf)
```

which outputs:

```
x_q = -7.437500000000000
          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 8
        FractionLength: 4
             RoundMode: nearest
          OverflowMode: saturate
           ProductMode: FullPrecision
  MaxProductWordLength: 128
               SumMode: FullPrecision
      MaxSumWordLength: 128
         CastBeforeSum: true
```

Octave users have to look after the Fixed Point Toolbox for Octave.[5] Its syntax is a = fixed(bi, bf, x), where a is an object of class FixedPoint. The class field a.x stores

[5] Version 0.7.10 is relatively old (2004) and is currently not part of Octave-Forge, but there are interested users [url1fix].

the value in the regular floating point format.[6] The Octave command for representing $x = -7.45$ as in the previous example is:

```
x=-7.45;bi=3;bf=4;x_q=fixed(bi,bf,x)
```

It should be noted that the following commands bi=0; bf=1;a=fixed(bi,bf,0.4) output a=0 because fixed implements truncation (to 0) instead of rounding (to $\Delta = 0.5$ in this case). As another example, c=fixed(bi,bf,-0.6) outputs c=-1 (-0.6 was truncated to -1). Following object-oriented programming (OOP) practice, one should be consistent when dealing with the objects. For example, a+4 returns an error message because a is an object, not a number, and a.x+4 should be used instead. On the other hand, a+c is a valid operation that returns -1 when the objects were created with the previous commands. □

1.9.6 IEEE 754 floating-point standard

Choosing the point position provides a degree of freedom for trading off range and precision in a fixed-point representation. However, there are situations in which a software needs to deal simultaneously with small and large numbers, and having distinct positions for the point all over the software is not practical or efficient. For example, if distances in both nanometers and kilometers should be manipulated with reasonable precision, the dynamic range of $10^3/10^{-9} = 10^{12}$ would require $b_i > 40$ bits for the integer part or keeping track of two distinct point positions. Hence, in many cases, a *floating-point* representation is more adequate.

A number in floating point is represented as in scientific notation:

$$x_q = \text{significand} \times \text{base}^{\text{exponent}}. \tag{1.18}$$

For example, assuming the base is 10 for simplicity, the number $x_q = 123.4567$ is interpreted as $x_q = 1.234567 \times 10^2$, where 1.234567 is the significand and 2 the exponent. While in fixed point the $b-1$ bits available to represent a signed number are split between b_i and b_f, in floating point the b bits are split between the significand and exponent.

The key aspect of Eq. (1.18) is that, as the value of the exponent changes, the "point" has new positions and allows to play a different trade off between range and precision: the values a floating point x_q can assume are not uniformly spaced. While fixed-point numbers are uniformly spaced by $\Delta = 2^{-b_f}$, which then imposes the maximum error within the range, the error magnitude in a floating point representation depends on the value of the exponent. For a given error in the significand representation, the larger the exponent, the larger the absolute error of x_q with respect to the number it represents. This is a desired feature in many applications and floating point is used for example in all personal computers. The fixed point niche are embedded systems, where power consumption and cost are more stringent requirements.

[6] The command whos allows to inspect the variable classes / types.

The IEEE 754 standard for representing numbers and performing arithmetics in floating point is adopted in almost all computing platforms. Among other features[7], it provides support to *single* and *double* precision, which are typically called "float" and "double", respectively, in programming languages such as C and Java.

The wide adoption of IEEE 754 is convenient because a binary file written with a given language in a given platform can be easily read with a distinct language in another platform. In this case, the only concern is to make sure the *endianness* is the same (Appendix B.5.3). The two following examples discuss numerical errors and quantization in practical scenarios.

Example 1.9. **Floating point representation in Matlab/Octave and numerical errors**. Unless specified otherwise, Matlab/Octave uses double precision. For example, the commands clear all;x=3;whos generate the following output:

```
Variables in the current scope:
   Attr Name        Size                     Bytes  Class
   ==== ====        ====                     =====  =====
        x           1x1                          8  double
Total is 1 element using 8 bytes
```

in Octave and equivalent information in Matlab. As indicated, numbers in double precision use $b = 64$ bits while $b = 32$ are used in single precision ("float"). A double allocates 11 bits to the exponent and 52 to the significand, while in float precision these numbers are 8 and 23, respectively. The sign bit is used for the significand, but the exponent can also be a positive or negative number. Hence, one can consider that one exponent bit is used to represent its own sign.

The following Matlab/Octave code can be used to investigate the ranges for single and double precision:

Listing 1.5: *MatlabOctaveCodeSnippets/snip_signals_data_precision.m*

```
str = 'Ranges for double before and after 0:\n%g to %g and %g to %g';
sprintf(str, -realmax, -realmin, realmin, realmax)
str = 'Ranges for float before and after 0:\n%g to %g and %g to %g';
sprintf(str,-realmax('single'),-realmin('single'), ...
    realmin('single'), realmax('single'))
```

The output is:

```
ans = Ranges for double before and after 0:
-1.79769e+308 to -2.22507e-308 and 2.22507e-308 to 1.79769e+308
ans = Ranges for float before and after 0:
-3.40282e+038 to -1.17549e-038 and 1.17549e-038 to 3.40282e+038
```

From this output, it would be a mistake to consider that $\Delta = 2.22507e-308$ and $1.17549e-038$ for double and single precision, respectively. Recall that the floating point numbers are non-uniformly spaced.

[7] See, e. g., [url1flo] for more details.

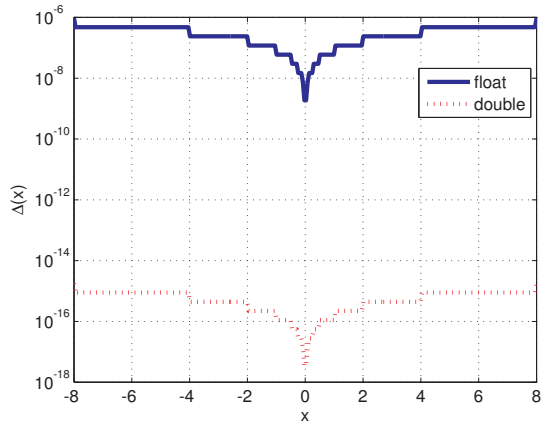


Figure 1.14: *Comparison of step sizes for IEEE 754 floating points with single and double precision in the range* $[-8, 8]$. *Note as* $\Delta(x)$ *increases with* $|x|$.

Given that the step $\Delta(x)$ varies from a number x to the next number $x + \Delta(x)$ in floating point, Matlab/Octave provides the command `eps(x)` to obtain $\Delta(x)$. Figure 1.14 provides a comparison obtained with Listing 1.6.

Listing 1.6: *MatlabOctaveCodeSnippets/snip_signals_delta_calculation.m*

```
N=300; delta_x=zeros(1,N); x=linspace(-8,8,N); %define range
%use loops to be compatible with Octave. Matlab allows delta_x=eps(x)
for i=1:N, delta_x(i) = eps(single(x(i))); end %single precision
semilogy(x,delta_x); hold on
for i=1:N, delta_x(i) = eps(x(i)); end  %double precision
semilogy(x,delta_x,'r:'); legend('float','double'); grid
```

Figure 1.14 indicates that care must be exercised especially when dealing with single precision, which is a requirement of many DSP chips, for example. Even double precision can cause strange behavior. A good example is provided by Listing 1.7, from Mathwork's documentation [url1flm].

Listing 1.7: *MatlabOctaveCodeSnippets/snip_signals_numerical_error.m*

```
a = 0.0; %a uses double precision
for i = 1:20
  a = a + 0.1; %20 times 0.1 should be equal to 2
end
a == 2 %checking if a is 2 returns false due to numerical errors
```

The design of algorithms that are robust to numerical errors, such as matrix inversion, is the focus of many textbooks. Besides trying to adopt robust algorithms, a DSP programmer needs to always be aware of the possibility of numerical errors. Taking the example of the previous code, instead of a check such as `if (a==2)`, it is often better to write

```
if abs(a-2) < eps %check if a is 2 (consider numerical errors)
```

where eps corresponds to eps(1) and is the default when a better guess for the range of interest (eps(2) in the example) is not available.

It is possible to instruct Matlab/Octave to use single (using the function single) or double precision (the default) as illustrated in Listing 1.8, which uses the FFT algorithm (to be discussed in Chapter 2) to compare the options with respect to speed.

Listing 1.8: *MatlabOctaveCodeSnippets/snip_signals_single_precision.m*

```
N=2^20; %FFT length (one may try different values)
xs=single(randn(1,N)); %generate random signal using single precision
xd=randn(1,N); %generate random signal using double precision
tic %start time counter
Xs=fft(xs); %calculate FFT with single precision
disp('Single precision: '), toc %stop time counter
tic %start time counter
Xd=fft(xd); %calculate FFT with double precision
disp('Double precision: '), toc %stop time counter
```

Note that benchmarking is tricky and using single precision may not be faster than double precision. On a given laptop, Listing 1.8 executed on Matlab returned 0.073124 and 0.104728 seconds, which indicates that double precision was approximately 1.43 times slower than single precision. Executing the code on the same machine using Octave led to approximately 0.06 seconds to both double and single precision. □

Example 1.10. **The step size of the ADC quantizer is often unavailable**. When the quantization is performed in the digital-domain, such as when representing a real number in fixed point, the value of the step Δ is readily available. However, the value of the step Δ_{AD} in the quantization process that takes place within an ADC chip is often not available.

For example, as explained in Application 1.1, the Matlab/Octave function's wavread returns $x_i[n]$ (when the option native is used in Matlab) or a scaled version of $x_i[n]$, where the scaling factor is not Δ_{AD} but 2^{b-1}. The value of Δ_{AD} that the sound board's ADC used to quantize the analog values obtained via a microphone for example, is often not registered in audio files (e. g., 'wav'), which store only the samples $x_b[n]$ and other information such as F_s. Application 1.5 discusses more about Δ_{AD} in commercial sound boards.

File formats developed for other applications may include the value of Δ_{AD} in their headers. For example, Application 2.4 uses an ECG file that stores the information that $\Delta_{\text{AD}} = 1/400$ mV. The analog signal power can then be estimated as 3.07 mW by normalizing the digital samples with $x_i[n] \times \Delta_{\text{AD}}$.

If the correct value of Δ_{AD} is not informed, it is still possible to relate the analog and digital signals in case there is information about the analog signal dynamic range.

As an example of using proportion to relate x_q and x_i, assume a 2-bits ADC that adopts unsigned integers as indicated in Table 1.5. The task is to relate the integer

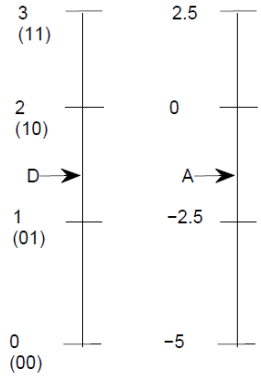


Figure 1.15: *Example of conversion using proportion when the dynamic ranges of both analog and digital signal are available. In this case the* step size *is found as* $\Delta = 2.5$ *V and the value* A *(or* x_q*) is related to the integer representation* D *(or* x_i*) as* $x_q = \Delta x_i - 5$.

codewords $x_i[n] \in \{0, 1, 2, 3\}$ (corresponding to 00, 01, 10 and 11, respectively) to $x_q[n]$ values, which can represent Volts, Amperes, etc. It is assumed that the original $x(t)$ is given in Volts and continuously varies in the range $[-5, 2.5]$. The relation between x_i and x_q can be found by proportion as:

$$\frac{x_i - 0}{3 - 0} = \frac{x_q - (-5)}{2.5 - (-5)},$$

which gives

$$x_q = \left(\frac{7.5}{3}\right) x_i - 5 = 2.5 x_i - 5$$

and, consequently, an estimate of $\Delta = 2.5$ V. In this case, the ADC associates the code 00 to -5 V, 01 to -2.5 V, 10 to 0 V and 11 to 2.5 V, as illustrated in Figure 1.15. □

1.10 Signal Classification: Randomness, Periodicity and Power

This section discusses several categories of signals.

1.10.1 Random signals and their generation

Discrete-time random signals can be represented by vectors in which the elements are outcomes of random variables (see Appendix A.19). For example, assuming all elements of [2, 0, 2, 3, 3, 2, 3] are outcomes of the same random variable X, one can calculate the average $\mathbb{E}[\mathsf{X}] \approx 2.14$, the standard deviation $\sigma \approx 1.07$ and other statistical moments of X. This vector could be modeled as a finite-duration random signal. Alternatively, a vector or random samples correspond to a *realization* of a discrete-time random process. In both cases, for simplicity, these signals will be denoted as $x[n]$, the same notation for non-random signals.

It is easy to generate random signals $x[n]$ in Matlab/Octave. For example, the command `x=randn(1,100)` generates 100 samples distributed according to a *standard Gaussian* (zero-mean and unity-variance) distribution $\mathcal{N}(0,1)$, where the notation $\mathcal{N}(\mu,\sigma^2)$ assumes the second argument is the variance σ^2, not the standard deviation σ. These signals can be visualized as a time-series,[8] such as in Figure 1.16.

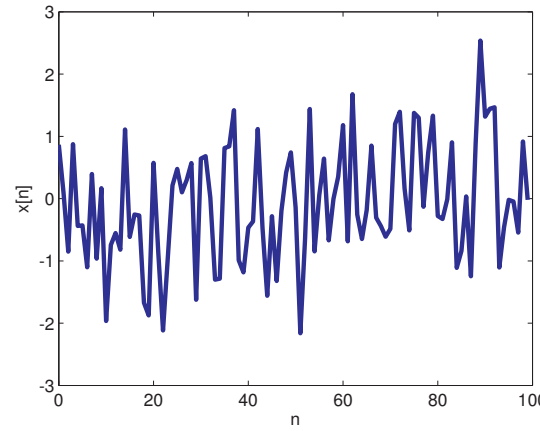


Figure 1.16: *Waveform representation of a random signal with 100 samples draw from a Gaussian distribution* $\mathcal{N}(0,1)$.

The time-domain visualization can be complemented by plotting the probability distribution of the random signal. Figure 1.17 illustrates the histogram of the signal depicted in Figure 1.16. The histogram is calculated by dividing the dynamic range in *bins* and counting the number of samples that belong to each bin. Figure 1.17 was obtained by using 10 bins (the default).

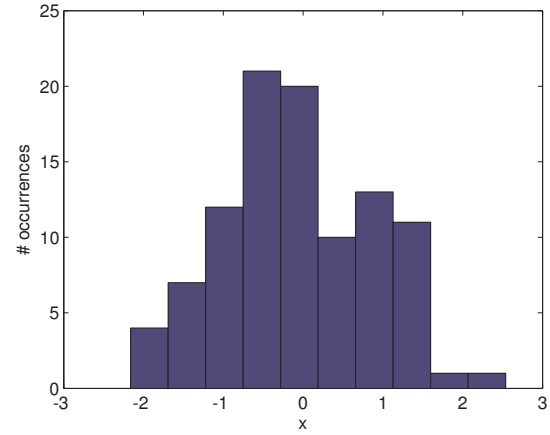


Figure 1.17: *Histogram of the signal in Figure 1.16 with 10 bins.*

[8] As usually done for signals with many samples, the discrete-time $x[n]$ was depicted as a continuous-time signal with the `plot` function instead of `stem`.

Normalizing the histogram by the total number of samples provides an estimate of the probability mass function (PMF). When this result is further normalized by the bin width, an estimate of the probability density function (PDF) is obtained (see Appendix A.13). The function ak_normalize_histogram.m can be used to estimate a PDF from a histogram and was used to obtain Figure 1.18 according to the commands in Listing 1.9.

Listing 1.9: *MatlabOctaveCodeSnippets/snip_signals_estimate_pdf.m*

```
B=10; %number of bins
x=randn(1,100); %random numbers ~ G(0,1)
[n2,x2]=ak_normalize_histogram(x,B);%PDF via normalized histogram
a=-3:0.1:3; %use range of [-3std, 3std] around the mean
plot(x2,n2,'o-',a,normpdf(a),'x-') %estimate vs. theoretical PDF
```

Figure 1.18 indicates that 100 samples and 10 bins provide only a crude estimation.

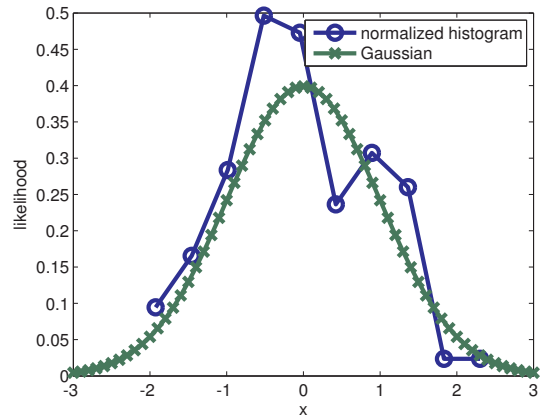


Figure 1.18: *PDF estimate from the histogram in Figure 1.17. The histogram values were divided by the product of the total number of samples and bin width. A standard Gaussian PDF is superimposed for the sake of comparison.*

Figure 1.19 was obtained by using 10 thousand samples from a Gaussian $\mathcal{N}(4, 0.09)$ and using 100 bins for the histogram. Now the Gaussian estimation is relatively good. Listing 1.10 was used to generate the samples of $\mathcal{N}(4, 0.09)$ and illustrates how to draw samples from a Gaussian with any given mean and variance from calls to a random number generator that outputs samples from a standard Gaussian $\mathcal{N}(0, 1)$.

Listing 1.10: *MatlabOctaveCodeSnippets/snip_signals_gaussian_rand_gen.m*

```
newMean=4; %new mean
newVariance=0.09; %new variance
N=10000; %number of random samples
x=sqrt(newVariance)*randn(1,N)+newMean;
```

It should be noted that the normalized histogram of a continuous PDF indicates *likelihood*, not probability. The likelihood function is the PDF viewed as a function of the parameters. Therefore, it is possible to have values *larger than one* in the ordinate, such as in Figure 1.19.

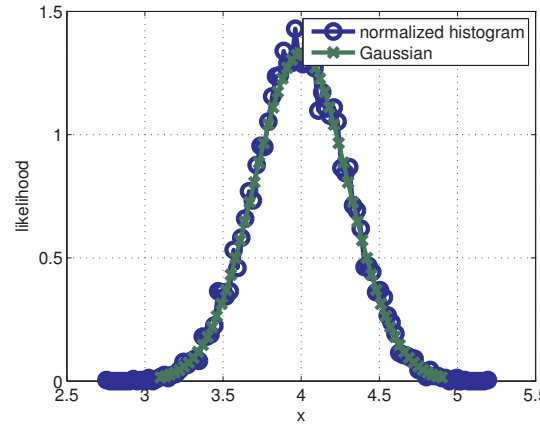


Figure 1.19: *Comparison of normalized histogram and the correct Gaussian* $\mathcal{N}(4, 0.09)$ *when using 10,000 samples and 100 bins. Note the likelihood can be larger than one because it is not a probability.*

As mentioned, the command x=sqrt(newVariance)*randn(1,N)+newMean is useful for obtaining Gaussians with arbitrary mean and variance. When using the random number generator rand for uniformly distributed samples, one should notice that the dynamic range is $[0, 1]$, the mean is 0.5 and the variance of a uniform PDF is

$$\sigma^2 = \mathbb{E}[(x-\mu)^2] = \int_a^b \frac{1}{b-a}\left(x - \frac{(a+b)}{2}\right)^2 dx = \frac{S^2}{12}, \tag{1.19}$$

where $S = b-a$ is the *support* (range from a to b of values that have non-zero likelihood) of the PDF, which in the example leads to $S^2/12 = 1/12 \approx 0.0833$. Drawing from rand, it is possible to generate N samples uniformly distributed in an arbitrary range $[a, b]$ with the command x = a + (b-a) * rand(N,1).

1.10.2 Periodic and aperiodic signals

Periodicity in continuous-time

A signal $x(t)$ (same discussion applies to $x[n]$) is periodic if a given segment of $x(t)$ is eternally repeated, such that $x(t) = x(t+T)$ for some $T > 0$, where T is called the period. For example, if $T = 10$ seconds and $x(t) = x(t+10)$, for all values of t.

In the example of $T = 10$, it is easy to check that $x(t) = x(t+20)$, $x(t) = x(t+30)$ and so on. In other words, a signal with period T is also periodic in $2T, 3T, \ldots$. The *fundamental period* T_0 is the smallest value of T for which $x(t) = x(t+T_0)$. Note that

the definition imposes $T_0 > 0$ and the same is required for the period $N_0 > 0$ of a discrete-time signal.

Example 1.11. **Using the LCM and GCD for a periodic signal composed by commensurate frequencies**. Two frequencies f_1 and f_2 are called *commensurate* if their ratio f_1/f_2 can be written as a rational number m/n, where m, n are non-zero integers. Instead of frequencies, one can use their associated time periods.

Assume a signal $x(t) = \sum_{i=1}^{N} x_i(t)$ is composed by the sum of N periodic components $x_i(t)$, each one with period T_i and frequency $f_i = 1/T_i$. The set of frequencies $\{f_i\}$ is commensurate if all pairs are commensurate. In this case, the fundamental period T_0 of $x(t)$ can be found using the least common multiple (LCM) of the periods $\{T_i\}$ while its frequency $F_0 = 1/T_0$ can be found using the greatest common divisor (GCD) of the frequencies $\{f_i\}$. Assuming both LCM and GCD are defined only for integer numbers, it may be needed to extract a common factor and later reintroduce it. A numerical example helps: let $x(t) = \cos(2\pi f_1 t) + \cos(2\pi f_2 t + \pi/2) + \sin(2\pi f_3 t)$ be composed by sinusoids with frequencies $f_1 = 5/2$, $f_2 = 1/6$ and $f_3 = 1/8$ Hz, which corresponds to periods $T_1 = 0.4$, $T_2 = 6$ and $T_3 = 8$ seconds, respectively. To find the LCM, one may need to multiply all periods by 10 and then calculate that $\text{LCM}(4, 60, 80) = 240$. Dividing this result by the factor 10 leads to $T_0 = 24$ s. This LCM could be obtained in Matlab/Octave with `lcm(lcm(4,60),80)` given that this function is limited to accept two input arguments only. □

Periodicity in discrete-time

One important thing is that the discrete-time counterpart of some periodic analog signals may be non-periodic. For example the signal $x(t) = \cos(3t)$ is periodic with period $T = 2\pi/3$ s. However, the discrete-time signal $x[n] = \cos(3n)$ is non-periodic. In fact, all discrete-time cosines (and sines) $x[n] = A\cos(\omega n + \phi)$ are periodic only if $\omega/(2\pi)$ is a ratio m/N of two integers m and N as proved below.[9] The value of $x[n + N]$ is

$$x[n + N] = A\cos(\Omega(n + N) + \phi) = A\cos(\Omega n + \Omega N + \phi).$$

If the parcel ΩN in previous expression is a multiple of 2π, then $x[n + N] = x[n], \forall n$. Hence, $\omega N = 2\pi m$, which leads to the condition

$$\frac{m}{N} = \frac{\Omega}{2\pi} \tag{1.20}$$

for a discrete-time cosine (and sine) signal to be periodic. For example, $x[n] = \cos(3n)$ is non-periodic because $\omega = 3$ and $3/(2\pi)$ cannot be written as a ratio of two integers. In contrast, $x[n] = \cos((2\pi/8)n + 0.2)$ is periodic with period $N = 8$ ($m = 1$ in this case). The signal $x[n] = \cos(7\pi n)$ is periodic because $\omega = 7\pi$ and $\omega/(2\pi) = 7/2$, with $m = 7$ and $N = 2$.

[9] When m/N is not a rational number, the discrete-time sinusoid is called *almost-periodic* [Cor68,Gia99].

In summary, to find the period T of a continuous-time cosine $\cos(\omega t + \phi)$, one can obtain the term ω that multiplies t and calculate

$$T = \frac{2\pi}{\omega},$$

which is given in seconds if ω is in rad/s. In contrast, a discrete-time cosine $\cos(\Omega n + \phi)$ may not be periodic. If someone tries to simply calculate $N = \frac{2\pi}{\Omega}$, it may end up with a non-integer period. The condition for periodicity is to be able to write $2\pi/\Omega$ as a ratio of integers, i. e.

$$\frac{N}{m} = \frac{2\pi}{\Omega},$$

where N is the period in samples.

To understand the role of m, consider the signal $\cos((3\pi/17)n)$. In this case, $\frac{2\pi}{\Omega} = 34/3$ cannot be the period because it is not an integer. However, if one allows for $m = 3$ times the number of samples specified by $\frac{2\pi}{\Omega}$, the result is the integer period $N = m\frac{2\pi}{\Omega} = 34$ samples. See Application 1.7 for a discussion on finding m and N using Matlab/Octave.

Sometimes, it is misleading to guess the period of a discrete-time cosine or sine via the observation of its graph. Figure 1.20 depicts the graph of $x[n] = \sin(0.2n)$ and was obtained with the following code:

```
M=100, w=0.2; %%num of samples and angular freq. (rad)
n=0:M-1; %generate abscissa
xn=sin(w*n); stem(n,xn); %generate and plot a sinusoid
```

In this case, the signal seems to have a period around 31 samples at a first glance (because $2\pi/\Omega \approx 31.4$). But, for example, $x[n]$ will never be 0 at the beginning of a cycle for a value of n other than $n = 0$. Therefore, in spite of resembling a periodic signal, the angular frequency Ω is such that $2\pi/\Omega = 10\pi$ is a irrational number and a cycle of numbers will never repeat. In this case, the signal is called *almost* or *quasi* periodic.[10]

It is useful to visualize a discrete-time sinusoid that is periodic with $m > 1$. Figure 1.21 depicts the graph of $x[n] = \sin((3\pi/17)n)$ and illustrates the repetition of $m = 3$ sine *envelopes* within a period of $N = 34$ samples.

1.10.3 Power and energy signals

It is important to understand the concepts of power $\mathcal{P}$ and energy E of a signal. One reason is that, in some cases, the equation to be used for a specific analysis (autocorrelation, for example) differs depending if $\mathcal{P}$ or E are not finite. This section assumes continuous-time but the concepts are also valid for discrete-time signals.

If E is the energy dissipated by a signal during a time interval Δt, the average power along Δt is

$$\mathcal{P} = \frac{E}{\Delta t}.$$

[10] See, e. g., [Gia99, Ant07] to see the importance of almost periodic signals in random processes.

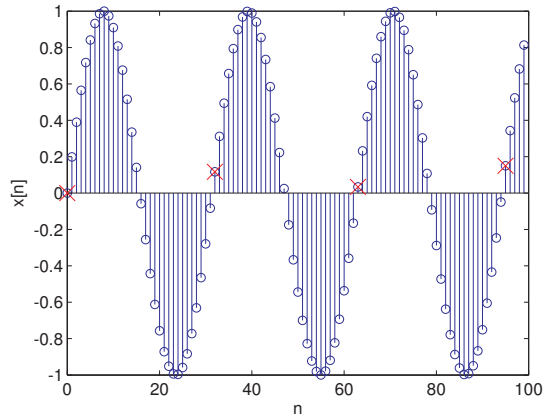


Figure 1.20: *Graph of the signal* $x[n] = \sin(0.2n)$. *Observe carefully that this signal is not periodic. The first non-negative sample of the sine cycle will never exactly repeat its value, as indicated by the 'x' marks.*

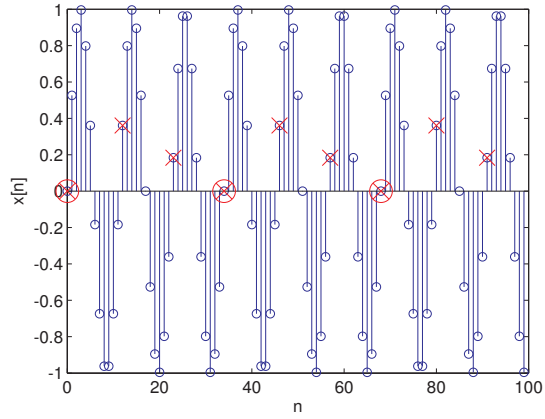


Figure 1.21: *Graph of the signal* $x[n] = \sin((3\pi/17)n)$. *The signal has period* $N = 34$ *samples as indicated by the combined marks 'x' and 'o'. This value of* N *corresponds to* $m = 3$ *cycles of a sine envelope corresponding to* $2\pi/\Omega = 34/3 \approx 11.3$.

If $p(t) = |x(t)|^2$ is the instantaneous power of $x(t)$, E can be calculated as

$$E = \int_{\langle \Delta t \rangle} p(t)dt. \tag{1.21}$$

If the interval Δt is not specified, it is implicitly assumed (by *default*) the whole time axis $]-\infty, \infty[$ and

$$E = \int_{-\infty}^{\infty} p(t)dt.$$

In this case, $\mathcal{P}$ is defined as the limit

$$\mathcal{P} = \lim_{\Delta t \to \infty} \left[\frac{1}{\Delta t} \int_{-\Delta t/2}^{\Delta t/2} p(t) dt \right]. \tag{1.22}$$

Note that, because the time interval goes to infinite (denominator), $\mathcal{P}$ is zero unless the energy E (numerator) also goes to infinite. This situation suggests the definition of *power* and *energy* signals, which have *finite* power and energy, respectively. Their characteristics are summarized in Table 1.6, which also indicates that there is a third category for signals that have neither finite power or energy.

Table 1.6: *Total energy E and average power $\mathcal{P}$ for two kinds of signal assuming an infinite time interval.*

Category	E	$\mathcal{P}$	Example(s)
Power signal	∞	finite	$\cos(\omega t)$ and other periodic signals
Energy signal	finite	0	$e^{-t}u(t)$ and $t^2[u(t) - u(t-5)]$
Neither	∞	∞	t

The most common power signals are periodic. In this case, the energy E_T in one period T

$$E_T = \int_{\langle T \rangle} p(t) dt$$

can be used to easily calculate

$$\mathcal{P} = \frac{E_T}{T}$$

because what happens in one period is replicated along the whole time axis.

The most common energy signals have a finite duration, such as $x(t) = t^2[u(t) - u(t-5)]$. Assuming the signals have finite amplitude, their energy in a finite time interval cannot be infinite. Note that infinite duration signals, such as $x(t) = e^{-t}u(t)$, can also have a finite energy in case their amplitude decay over time.

It is assumed throughout this text that the signals are currents $i(t)$ or voltages $v(t)$ over a resistance R, such that the instantaneous power is

$$p(t) = v(t)i(t) = \frac{1}{R}v^2(t) = i^2(t)R.$$

Besides, to deal with signals $x(t)$ representing both currents and voltages without bothering about the normalization by R, it is assumed that $R = 1$ Ohm. Hence, the instantaneous power is $p(t) = x^2(t)$ for any real $x(t)$ and, more generally, $p(t) = |x(t)|^2$ in case $x(t)$ is complex-valued.

Throughout the book, unless stated otherwise, $x(t)$ is assumed to be in Volts, $p(t)$ and $\mathcal{P}$ in Watts and E in Joules. A dimensional analysis of $p(t) = x^2(t)$ should not be interpreted directly as Watts = Volts^2, but Watts = Volts^2/Ohm, where the normalization by 1 Ohm is implicit. Two examples are provided in the sequel.

Example 1.12. **Sinusoid power.** Sinusoids and cosines can be represented by $x(t) = A\cos(\omega_0 t + \theta)$ and are power signals with average power $\mathcal{P} = \frac{A^2}{2}$. The phase θ does not influence the power calculation. The proof follows.

The angular frequency is $\omega_0 = \frac{2\pi}{T}$ rad/s, where T is the period in seconds.

$$E_T = \int_{\langle T \rangle} p(t)dt = \int_{\langle T \rangle} x^2(t)dt = A^2 \int_{\langle T \rangle} \cos^2(\omega_0 t + \theta)dt.$$

Using the identity $\cos^2 a = \frac{1}{2}(\cos(2a) + 1)$ (see Appendix):

$$E_T = \frac{A^2}{2} \int_{\langle T \rangle} (\cos(2\omega_0 t + 2\theta) + 1)dt = \frac{A^2 T}{2}.$$

The first parcel of the integral is zero, independent of 2θ because T corresponds exactly to two periods of the cosine with angular frequency $2\omega_0$, while the second parcel is T. The average power is

$$\mathcal{P} = \frac{E_T}{T} = \frac{A^2}{2},$$

which is a result valid for any sinusoid or cosine. □

Example 1.13. **Power of a DC signal.** A constant signal $x(t) = K$ (i. e., a DC signal) has power $\mathcal{P} = K^2$ because the energy at any interval Δt is $E = K^2 \Delta t$. □

The root-mean-square (RMS) value x_{rms} of any signal $x(t)$ is the DC value that corresponds to the same power $\mathcal{P}$ of $x(t)$, i. e., $x_{\text{rms}}^2 = \mathcal{P}$ or, equivalently, $x_{\text{rms}} = \sqrt{\mathcal{P}}$. For example, the RMS value of a cosine $x(t) = A\cos(\omega_0 t + \theta)$ is $x_{\text{rms}} = \frac{A}{\sqrt{2}}$ because a DC signal $y(t) = \frac{A}{\sqrt{2}}$ has the same average power as $x(t)$.

As discussed in Section A.29.4, $\delta(t)$ is a distribution and it is tricky to define the energy or power of a sampled signal, which is the topic of Section **??**.

1.11 Power and Energy in Discrete-Time

The concepts of power and energy are better defined for continuous-time than for discrete-time signals and vectors. Dealing with the concept of power of a discrete-time signal $x[n]$ requires some caution because n does not have the dimension of time. In contrast to $x[n]$, the notation x_i is used when i should not be interpreted as "time" but, for example, as the i-th element of a set.

1.11.1 Power and energy of discrete-time signals

The following definition of average power will be adopted in this text

$$\mathcal{P} \triangleq \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x[n]|^2. \tag{1.23}$$

and interpreted as Watts given that $x[n]$ is assumed to be in Volts. This is a sensible definition, as will be discussed in Section 1.12.

Accordingly, the energy E of a discrete-time signal is

$$E \triangleq \sum_{n=-\infty}^{\infty} |x[n]|^2. \tag{1.24}$$

such that $\mathcal{P} = E/N$ when the support of $x[n]$ are N samples.

A similar situation occurs when dealing with vectors.

1.11.2 Power and energy of signals represented as vectors

Here, $x[n]$ denotes the n-th element of a vector $\mathbf{x}$. The order of these elements is assumed to correspond to a time evolution, such that a finite-dimension vector is equivalent to a finite-duration discrete-time sequence. Hence, the energy E of vector $\mathbf{x}$ is its squared Euclidean norm:

$$E = \sum_{n=1}^{N} |x[n]|^2 = ||\mathbf{x}||^2, \tag{1.25}$$

where N is the dimension of $\mathbf{x}$. Accordingly, the power of such finite-dimension vector is

$$\mathcal{P} = \frac{E}{N} = \frac{1}{N} \sum_{n=1}^{N} |x[n]|^2. \tag{1.26}$$

1.11.3 Power and energy of vectors whose elements are not time-ordered

In contrast to the previous equations, there are cases in which the vector elements are not indexed according to a time evolution.

It should be noticed that the average of the squared norms of several vectors should be interpreted as their average energy, not power. For example, assuming there are M vectors $\mathbf{x}_1, \ldots, \mathbf{x}_M$ of dimension N, an average energy is obtained with

$$\overline{E} = \frac{1}{M} \sum_{i=1}^{M} E_i = \frac{1}{M} \sum_{i=1}^{M} ||\mathbf{x}_i||^2, \tag{1.27}$$

and interpreted in Joules.

Eq. (1.26) and Eq. (1.27) are similar due to the connection between vectors and finite-duration discrete-time signals. The distinction that allows to observe their results as "power" or "energy" relies on interpreting the index i as "time" or not. In Eq. (1.26), an average of instantaneous power values $|x(i)|^2$ along the "time" i leads to an estimate of power. But when taking an average $\overline{E}$ over vectors of energy E_i in Eq. (1.27), the result is (average) energy. This distinction is important in situations such as Eq. (**??**), when $\overline{E}$ is the average energy of a constellation.

To make sure the concepts are clear, applying Eq. (1.26) to each vectors $\mathbf{x}_i$ and taking the average of their power $\mathcal{P}_i$ leads to

$$\mathcal{P} = \frac{1}{M}\sum_{i=1}^{M}\mathcal{P}_i = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}|x_i(j)|^2 = \frac{\overline{E}}{N}.$$

1.11.4 Power and energy of discrete-time random signals

If $x[n]$ represents a random signal, with samples $x[n_0]$ corresponding to outcomes of a random variable X, an alternate definition is

$$\mathcal{P} \triangleq \mathbb{E}[|\mathsf{X}|^2] = \mathbb{E}[|x[n]|^2],$$

where $\mathbb{E}[\cdot]$ is the expectation operator.

The power of a random signal $x[n]$ (or $x(t)$) can be decomposed into two parcels as follows:

$$\mathcal{P} = \mathbb{E}[\mathsf{X}^2] = \sigma_x^2 + \mu_x^2, \tag{1.28}$$

where the variance σ_x^2 and the squared-mean μ_x^2 correspond to the powers of the AC and DC components of a real $x[n]$, respectively.

The proof is derived from the definition of variance: $\sigma_x^2 \triangleq \mathbb{E}[(\mathsf{X}-\mu_x)^2] = \mathbb{E}[\mathsf{X}^2 - 2\mathsf{X}\mu_x + \mu_x^2]$. Observing that μ_x is a constant and $\mathbb{E}$ is a linear operator, one can write $\sigma_x^2 = \mathbb{E}[\mathsf{X}^2] - 2\mathbb{E}[\mathsf{X}]\mu_x + \mu_x^2 = \mathbb{E}[\mathsf{X}^2] - \mu_x^2$.

Most of the signals in telecommunications and other applications have zero mean ($\mu = 0$). In this case, Eq. (1.28) shows that the power $\mathcal{P} = \sigma_x^2$ coincides with the variance of the random signal and the standard deviation σ_x with its RMS value.

1.12 Relating Power in Continuous and Discrete-Time

The goal here is to relate the power of a discrete-time $x[n]$ to the power of a continuous-time $x(t)$ where these signals are related by an A/D or D/A conversion. A possible processing chain relating these signals with their associated power in parenthesis is:

$$\underset{(\mathcal{P}_c)}{x(t)} \rightarrow \boxed{\text{sampling}} \rightarrow x_s(t) \rightarrow \boxed{\text{C/D}} \rightarrow \underset{(\mathcal{P}_d)}{x[n] = x(nT_s)}. \tag{1.29}$$

Another processing chain of interest is the reconstruction of a continuous-time signal:

$$x(nT_s) = \underset{(\mathcal{P}_d)}{x[n]} \rightarrow \boxed{\text{D/C}} \rightarrow x_s(t) \rightarrow \boxed{h(t)} \rightarrow \underset{(\mathcal{P}_c)}{x(t)} \tag{1.30}$$

which will be discussed in Section **??**.

Special interest lies on systems that have *equivalence between power in discrete and continuous-time*, such that

$$\mathcal{P}_d = \mathcal{P}_c, \tag{1.31}$$

where $\mathcal{P}_d$ and $\mathcal{P}_c$ are given by Eq. (1.23) and Eq. (1.22), respectively.[11]

Eq. (1.31) is further discussed in Section **??**, but here it is derived[12] using the rectangle method to approximate the integral of $p(t) = |x(t)|^2$ as a sum of rectangles with bases T_s and heights $|x[n]|^2$, as follows:

$$\begin{aligned}
\mathcal{P}_c &= \lim_{N\to\infty} \left[\frac{1}{(2N+1)T_s} \int_{-NT_s}^{NT_s} |x(t)|^2 dt \right] \\
&\approx \lim_{N\to\infty} \left[\frac{1}{(2N+1)T_s} \sum_{n=-N}^{N} T_s |x(nT_s)|^2 \right] \\
&= \lim_{N\to\infty} \left[\frac{1}{(2N+1)} \sum_{n=-N}^{N} |x[n]|^2 \right] \\
&= \mathcal{P}_d. \qquad (1.32)
\end{aligned}$$

In summary, when the sampling theorem is obeyed, the signal processing chains (filtering, amplification, etc.) associated to the A/D and D/A processes are assumed here not to alter the power of $x(t)$ and $x[n]$, such that Eq. (1.31) holds.

1.13 Correlation: A Measure of Dependence / Similarity

It is often useful to infer whether or not random variables depend on each other via *correlation*. For example, height and weight are quantities that present positive correlation for the human population. However, happiness and height are uncorrelated. There are measures other than correlation to detect dependencies, such as the *mutual information*, but correlation is simple and yet powerful. Correlation is called a second-order statistics because it considers only pairwise or quadratic dependence.

The correlation C for two complex-valued random variables X and Y is defined as

$$C = \mathrm{cor}(\mathsf{X}, \mathsf{Y}) = \mathbb{E}[\mathsf{X}\mathsf{Y}^*],$$

where $*$ denotes the complex conjugate. The *correlation coefficient*, which is a normalized version of the covariance (not correlation), is defined as

$$\rho = \frac{\mathrm{cov}(\mathsf{X}, \mathsf{Y})}{\sigma_x \sigma_y},$$

where the *covariance* is given by

$$\mathrm{cov}(\mathsf{X}, \mathsf{Y}) = \mathbb{E}[(\mathsf{X} - \mu_x)(\mathsf{Y} - \mu_y)^*]. \qquad (1.33)$$

Note that, when fully characterizing statistical moments of two complex random

[11] This is not the same, but similar to the energy-conservation property of Fourier transforms (e. g., Eq. (A.47), or its version for periodic signals Eq. (A.48)).

[12] Eq. (1.31) can also be obtained by assuming the zero-order hold (ZOH) reconstruction of Figure 1.9.

variables, one can take in account the interactions among their real and imaginary parts such that Eq. (1.33) becomes a 2×2 matrix. Complex-valued variables are briefly discussed in Appendix A.19.6 and real-valued random variables are assumed hereafter.

When the random variables are real-valued, Eq. (1.33) simplifies to

$$\text{cov}(\mathsf{X}, \mathsf{Y}) = \mathbb{E}[\mathsf{XY}] - \mu_x \mu_y.$$

Two real-valued random variables are called *uncorrelated* if, and only if, $\text{cov}(\mathsf{X}, \mathsf{Y}) = 0$, which is equivalent to

$$\mathbb{E}[\mathsf{XY}] = \mu_x \mu_y. \tag{1.34}$$

As mentioned, ρ is calculated by extracting the means $\mathbb{E}[\mathsf{X}] = \mu_x$ and $\mathbb{E}[\mathsf{Y}] = \mu_y$, i. e., using $\text{cov}(\mathsf{X}, \mathsf{Y})$ instead of $\text{cor}(\mathsf{X}, \mathsf{Y})$, and is restricted to $-1 \leq \rho \leq 1$ when the random variables are real-valued. Application 1.10 gives an example of using correlation to perform a simple data mining.

1.13.1 Autocorrelation function

Autocorrelation functions are extension of the correlation concept to signals. Table 1.7 summarizes the definitions that will be explored in this text.

Table 1.7: *Autocorrelation functions and their respective equation numbers.*

Type of process or signal	Equation
General stochastic process	(1.35)
Wide-sense stationary stochastic process	(1.36)
Deterministic continuous-time energy signal $x(t)$	(1.37)
Deterministic continuous-time power signal $x(t)$	(1.39)
Deterministic discrete-time energy signal $x[n]$ (unbiased estimate)	(1.42)
Deterministic discrete-time power signal $x[n]$	(1.43)

The existence of distinct autocorrelation definitions for power and energy signals illustrate the importance of distinguishing them, as discussed in Section 1.10.3.

Definitions of autocorrelation for random signals

An *autocorrelation function* (ACF) defined for stochastic processes (see Appendix A.19.5) is

$$R_X(s,t) = \mathbb{E}[\mathcal{X}(s)\mathcal{X}^*(t)], \tag{1.35}$$

which corresponds to the correlation between random variables $\mathcal{X}(s)$ and $\mathcal{X}(t)$ at two different points s and t in time of the same random process where $\mathcal{X}^*(t)$ is the complex conjugate of $\mathcal{X}(t)$. The purpose of the ACF is to determine the strength of relationship between amplitudes of the signal occurring at two different time instants.

For wide-sense stationary (WSS) processes, the ACF is given by

$$R_X(\tau) = \mathbb{E}[\mathcal{X}(t+\tau)\mathcal{X}^*(t)], \tag{1.36}$$

where the time difference $\tau = s - t$ is called the lag time.

The random process formalism is very powerful and allows for representing complicated processes. But in many practical cases only one realization $x(t)$ (or $x[n]$) of the random process $\mathcal{X}(t)$ is available. The alternative to deal with the signal without departing from the random process formalism is to assume that $\mathcal{X}(t)$ is *ergodic*. In this case, the statistical (or *ensemble*) average represented by $\mathbb{E}[\cdot]$ is substituted by averages over time.

The definitions of autocorrelation given by Eq. (1.35) and Eq. (1.36) are the most common in signal processing, but others are adopted in fields such as statistics. Choosing one depends on the application and the model adopted for the signals.

When dealing with a simple signal, one can naturally adopt one of the definition of autocorrelation for deterministic signals discussed in the sequel.

Definitions of autocorrelation for deterministic signals

A definition of ACF tailored to a deterministic (non-random) energy signal $x(t)$, which does not use expected values as Eq. (1.35), is

$$R_X(\tau) = \int_{-\infty}^{\infty} x(t+\tau)x^*(t)dt = \int_{-\infty}^{\infty} x^*(t-\tau)x(t)dt, \tag{1.37}$$

such that

$$R_X(0) = \int_{-\infty}^{\infty} |x(t)|^2 dt = E, \tag{1.38}$$

where E is the signal energy. It should be noted that this definition of autocorrelation cannot be used for power signals. Power signals have infinite energy and using Eq. (1.37) for $\tau = 0$ leads to $R_X(0) = \infty$ in the case of power signals, which is uninformative.

If $x(t)$ is a deterministic power signal, an useful ACF definition is

$$R_X(\tau) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} x(t+\tau)x^*(t)dt, \tag{1.39}$$

such that

$$R_X(0) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} |x(t)|^2 dt = \mathcal{P}, \tag{1.40}$$

where $\mathcal{P}$ is the signal average power. Eq. (1.39) can also be used when $x(t)$ is a realization of an ergodic stochastic process.

Similarly, there are distinct definitions of autocorrelation for deterministic discrete-time signals. For example, when $x[n]$ is a finite-duration real signal with N samples,

the (unscaled or not normalized) autocorrelation can be written as

$$\hat{R}_X[i] = \sum_{n=i}^{N-1} x[n]x[n-i], \tag{1.41}$$

which has a minus in $n-i$ that does not lead to reflecting the signal.[13]

As an alternative to use expressions more similar to the ones for signals with infinite duration, the (unscaled or not normalized) autocorrelation can be expressed as

$$R_X[i] = \sum_n x[n+i]x^*[n], \;\; i = -(N-1), \ldots, -1, 0, 1, \ldots, N-1$$

and computed assuming a zero value for $x[n]$ when its index is out of range. This corresponds to assuming the signal is extended with enough zeros (*zero-padding*) to the right and to the left. For example, assuming $x[n] = \delta[n] + 2\delta[n-1] + 3\delta[n-2]$, which can be represented by the vector $[1, 2, 3]$, its autocorrelation would be $[3, 8, 14, 8, 3]$, for the lags $i = -2, -1, 0, 1, 2$, respectively.

Table 1.8: *Example of autocorrelation for a real signal* $[1, 2, 3]$ *(*$n = 0, 1, 2$*).*

lag i	valid products $x^*[n]x[n+i]$	$R_X[i]$
-2	$x^*[2]x[0]$	3
-1	$x^*[1]x[0] + x^*[2]x[1]$	8
0	$x^*[0]x[0] + x^*[1]x[1] + x^*[2]x[2]$	14
1	$x^*[0]x[1] + x^*[1]x[2]$	8
2	$x^*[0]x[2]$	3

Note in Table 1.8 that the number of products decreases as $|i|$ increases. More specifically, when computing $R_X[i]$ there are only $N - |i|$ "valid" products. To cope with that, the normalized (and statistically unbiased) definition is

$$R_X[i] = \frac{1}{N-|i|} \sum_n x[n+i]x^*[n], \quad i = -(N-1), \ldots, N-1. \tag{1.42}$$

In Matlab/Octave, the unbiased estimate for the signal $[1, 2, 3]$ can be obtained with:

```
x=[1,2,3];xcorr(x,'unbiased')
```

which outputs [3,4,4.67,4,3]. The unscaled estimate of Table 1.8 can be obtained with xcorr(x,'none') or simply xcorr(x) because it is the default.

Another observation of interest is that for real signals, $R_X(\tau) = R_X(-\tau)$. In general, for complex-valued signals, $R_X(\tau) = R_X^*(-\tau)$, which is called *Hermitian symmetry*. Table 1.9 provides an example. It can also be noted that, for a given lag i, the subtraction of the indexes of all parcels in valid products is $(n+i) - n = i$.

[13] This operation should not be confused with convolution, which is discussed in Chapter 3.

Table 1.9: *Example of calculating the unscaled autocorrelation for a complex-valued signal* $[1+j, 2, 3]$ *($n = 0, 1, 2$), where* $j = \sqrt{-1}$.

lag i	valid products $x^*[n]x[n+i]$	$R_X[i]$
-2	$x^*[2]x[0]$	3+3j
-1	$x^*[1]x[0] + x^*[2]x[1]$	8+2j
0	$x^*[0]x[0] + x^*[1]x[1] + x^*[2]x[2]$	15
1	$x^*[0]x[1] + x^*[1]x[2]$	8-2j
2	$x^*[0]x[2]$	3-3j

Example 1.14. **Software implementation of autocorrelation**. The definitions of $R_X[i]$ used a generic summation $\sum_n$ over n. To be more concrete, an example of Matlab/Octave code to calculate the unscaled autocorrelation $R_X[i]$ is given in Listing 1.11. It can be seen that the property $R_X(\tau) = R_X^*(-\tau)$ is used to obtain the autocorrelation values for negative τ.

Listing 1.11: *MatlabOctaveCodeSnippets/snip_signals_unscaled_autocorrelation.m*

```
%Calculate the unscaled autocorrelation R(i) of x
x=[1+j 2 3] %define some vector x to test the code
N=length(x);
R=zeros(1,N); %space for i=0,1,...N-1
R(1)=sum(abs(x).^2); %R(0) is the energy
for i=1:N-1 %for each positive lag
    temp = 0; %partial value of R
    for n=1:N-i %vary n over valid products
        temp = temp + x(n+i)*conj(x(n));
    end
    R(i+1)=temp; %store final value of R
end
R = [conj(fliplr(R(2:end)))] %append complex conjugate
```

The function xcorr in Matlab/Octave uses a much faster implementation based on the fast Fourier transform (FFT), to be discussed in Chapter 2. When comparing the results of the two methods, the discrepancy is around 10^{-14}, which is a typical order of magnitude for numerical errors when working with Matlab/Octave. □

When infinite duration power signals can be assumed, it is sensible to define

$$R_X[i] = \lim_{N\to\infty} \frac{1}{2N} \sum_{n=-N}^{N} x[n+i]x^*[n]. \tag{1.43}$$

Examples of some signals autocorrelations

Two examples of autocorrelation are discussed in the sequel.

Example 1.15. **Autocorrelation of white signal**. A signal is called "white" when it has an autocorrelation $R[i] = A\delta[i]$ in discrete-time, or $R(\tau) = A\delta(\tau)$ in continuous-time, where $A \in \mathbb{R}$ is an arbitrary value. In other words, a white signal has an autocorrelation that is nonzero only at the origin, which corresponds to having samples that are uncorrelated and, consequently, statistically independent.

This nomenclature will be clarified in Chapter 4 but it can be anticipated that such signals have their power uniformly distributed over frequency and the name is inspired by the property of white light, which is composed by a mixture of color wavelengths.

As mentioned, the samples of a white signal are independent and, if the are also identically distributed according to a Gaussian PDF, the signal is called *white Gaussian noise* (WGN). More strictly, a WGN signal can be modeled as a realization of a wide-sense stationary process. In this case, WGN denotes the stochastic process itself. Using Matlab/Octave, a discrete-time realization of WGN can be obtained with function `randn`, as illustrated in Section 1.10.1. □

Example 1.16. **Autocorrelation of sinusoid**. Using Eq. (1.39), the autocorrelation of a sinusoid $x(t) = A\sin(\omega t + \phi)$ can be calculated as follows (to simplify, it is temporarily adopted $a = \omega t + \phi$):

$$\begin{aligned}
R(\tau) &= \lim_{T\to\infty}\left[\frac{1}{2T}\int_{-T}^{T} A\sin(\omega t+\phi)A\sin(\omega t+\omega\tau+\phi)dt\right] \\
&= \lim_{T\to\infty}\left[\frac{A^2}{2T}\int_{-T}^{T} \sin(a)\left[\sin(a)\cos(\omega\tau)+\sin(\omega\tau)\cos(a)\right]dt\right] \\
&= \lim_{T\to\infty}\left[\frac{A^2\cos(\omega\tau)}{2T}\int_{-T}^{T}\sin^2(a)dt + \frac{\sin(\omega\tau)}{2T}\int_{-T}^{T}\sin(a)\cos(a)dt\right] \\
&= \lim_{T\to\infty}\left[\frac{A^2\cos(\omega\tau)}{4T}\int_{-T}^{T}[\cos(2a)+1]dt + \frac{\sin(\omega\tau)}{4T}\int_{-T}^{T}\sin(2a)dt\right] \\
&= \lim_{T\to\infty}\left[\frac{A^2\cos(\omega\tau)}{2T}\frac{1}{2}\int_{-T}^{T}dt\right] \\
&= \frac{A^2\cos(\omega\tau)}{2}.
\end{aligned} \tag{1.44}$$

The simplification of the fourth equality is due to the fact that the integrals of both $\cos(2a)$ and $\sin(2a)$ are zero given the integration interval is an integer number of their periods. The final result indicates that the autocorrelation of a sinusoid or cosine does not depend on the phase ϕ and is also periodic in τ, with the same period $2\pi/\omega$ that the sinusoid has in t. Therefore, the frequencies contained in the realizations of a stationary random process can be investigated via the autocorrelation $R(\tau)$ of this process.

A simulation with Matlab/Octave can help understanding this result. Figure 1.22 was generated with Listing 1.12.

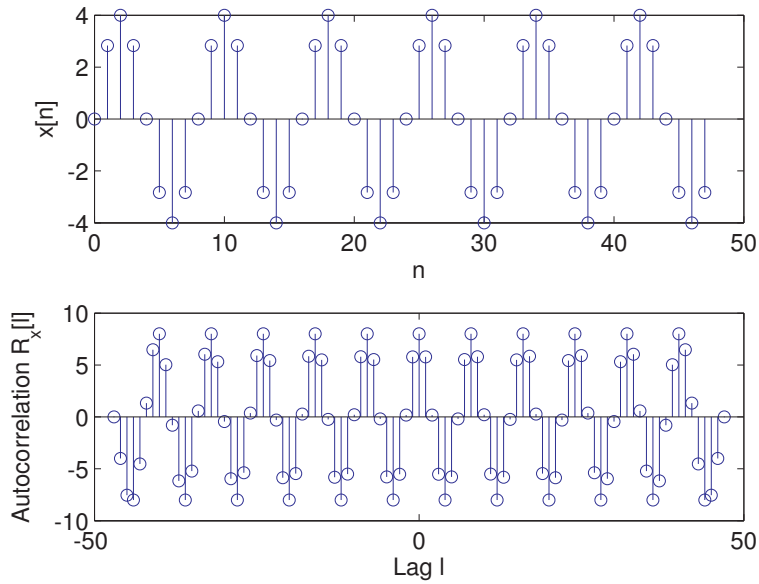


Figure 1.22: *A sinusoid of period N=8 samples and its autocorrelation, which is also periodic each 8 lags. The cosine corresponding to $R_x[l]$ has amplitude $A^2/2 = 4^2/2 = 8$.*

Listing 1.12: *MatlabOctaveCodeSnippets/snip_signals_sinusoid_autocorrelation.m*

```
numSamples = 48; %number of samples
n=0:numSamples-1; %indices
N = 8; %sinusoid period
x=4*sin(2*pi/N*n); %sinusoid (try varying the phase!)
[R,l]=xcorr(x,'unbiased'); %calculate autocorrelation
subplot(211); stem(n,x); xlabel('n');ylabel('x[n]');
subplot(212); stem(l,R); xlabel('Lag l');ylabel('R_x[l]');
```

It can be seen that the x and R are a sine and cosine, respectively, of the same frequency in their corresponding domains (n and l, respectively).

Figure 1.23 was obtained by changing the sinusoid period from N=8 to N=15 and illustrates the effects of dealing with finite-duration signals. Note that both the unbiased and unscaled versions have diminishing values at the end points. □

Using Eq. (1.43) and a calculation similar to the one used in Eq. (1.44), one can prove that the autocorrelation of $x[n] = \sin(\Omega n + \phi)$ is $R_X[i] = \cos(\Omega i)/2$.

1.13.2 Cross-correlation

The cross-correlation function (also called correlation) is very similar to the ACF but uses two distinct signals, being defined for deterministic energy signals as

$$R_{xy}(\tau) \triangleq \int_{-\infty}^{\infty} x(t+\tau)y^*(t)dt = \int_{-\infty}^{\infty} x(t)y^*(t-\tau)dt.$$

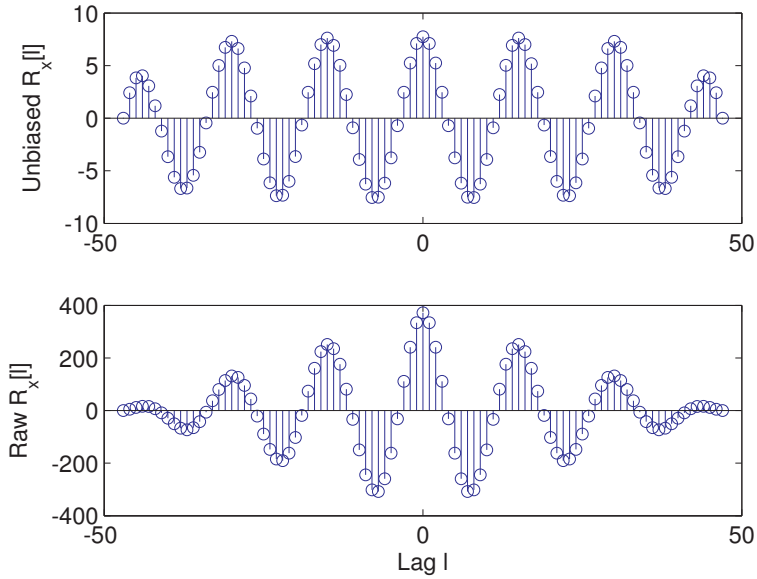


Figure 1.23: *The a) unbiased and b) raw (unscaled) autcorrelations for the sinusoid of Figure 1.22 with a new period of N=15 samples.*

Note the adopted convention with respect to the complex conjugate.

Some important properties of the cross-correlation are:

- $R_{xy}(\tau) = R^*_{xy}(-\tau)$ (Hermitian symmetry),
- $R_{xy}(\tau) = R^*_{yx}(-\tau)$ (swapping arguments is also Hermitian),
- $|R_{xy}(\tau)| \leq \sqrt{R_{xx}(0)R_{yy}(0)}$, (maximum is not necessarily at $\tau = 0$ but is bounded).

In discrete-time, the deterministic cross-correlation for energy signals is

$$R_{xy}[l] \triangleq \sum_{n=-\infty}^{\infty} x[n+l]y^*[n] = \sum_{n=-\infty}^{\infty} x[n]y^*[n-l].$$

When considering random processes, the two random variables are obtained from distinct processes:

$$R_{XY}(s,t) = \mathbb{E}[\mathcal{X}(s)\mathcal{Y}^*(t)] \tag{1.45}$$

or, assuming stationarity (see Section A.19.5) with $s = t + \tau$:

$$R_{XY}(\tau) = \mathbb{E}[\mathcal{X}(t+\tau)\mathcal{Y}^*(t)]. \tag{1.46}$$

Application 1.14 illustrates the use of cross-correlation to align two signals.

Before concluding this section, it is convenient to recall that the autocorrelation of a signal inherits any periodicity that is present in the signal. Sometimes this periodicity is more evident in the autocorrelation than in the signal waveform. The next example illustrates this point by discussing the autocorrelation of a sinusoid immersed in noise.

Example 1.17. **The power of a sum of signals is the sum of their powers in case they are uncorrelated**. Assume that a sinusoid $x[n]$ is contaminated by noise $z[n]$ such that the noisy version of the signal is $y[n] = x[n] + z[n]$. The signal $z[n]$ is a WGN (see Section 1.13.1) that is added to the signal of interest $x[n]$ and, therefore, called additive white Gaussian noise (AWGN). If $x[n]$ and $z[n]$ are uncorrelated, such that $R_{xz}[l] = \mathbb{E}[x[n+l]z[n]] = 0, \forall l$, the autocorrelation $R_y[l]$ of $y[n]$ is given by

$$\begin{aligned} R_y[l] &= \mathbb{E}[y[n+l]y[n]] = \mathbb{E}[(x[n+l] + z[n+l])(x[n] + z[n])] \\ &= R_x[l] + R_{zx}[l] + R_{xz}[l] + R_z[l] \\ &= R_x[l] + R_z[l]. \end{aligned}$$

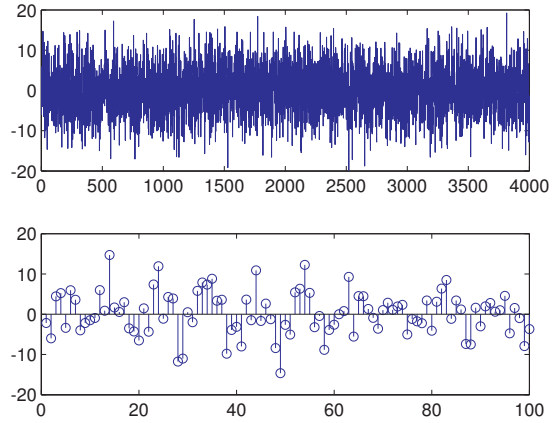


Figure 1.24: *Sinusoid of amplitude 4 V immersed in AWGN of power 25 W. The bottom graph is a zoom showing the first 100 samples.*

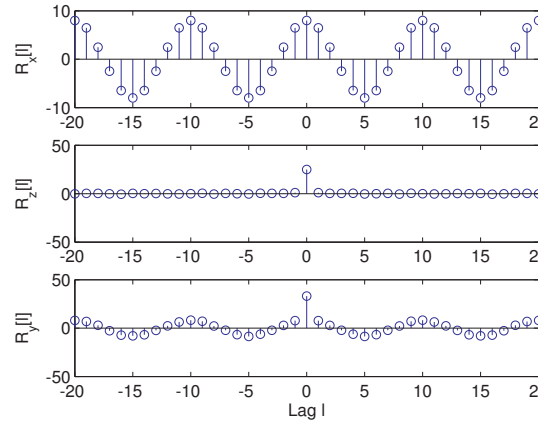


Figure 1.25: *Bottom graph: autocorrelation of the sine plus noise in Figure 1.24, top: autocorrelation of the sine and middle: autocorrelation of the noise.*

Listing 1.13 illustrates a practical use of this result. A sine with amplitude 4 V and

power $4^2/2 = 8$ W is contaminated by AWGN with power of 25 W. All signals are represented by 4,000 samples, such that the estimates are relatively accurate.

Listing 1.13: *MatlabOctaveCodeSnippets/snip_signals_noisy_sinusoid.m*

```
%Example of sinusoid plus noise
A=4; %sinusoid amplitude
noisePower=25; %noise power
f=2; %frequency in Hz
n=0:3999; %"time", using many samples to get good estimate
Fs=20; %sampling frequency in Hz
x=A*sin(2*pi*f/Fs*n); %generate discrete-time sine
randn('state', 0); %Set randn to its default initial state
z=sqrt(noisePower)*randn(size(x)); %generate noise
clf, subplot(211), plot(x+z); %plot noise
maxSample=100; %determine the zoom range
subplot(212),stem(x(1:maxSample)+z(1:maxSample)),pause; %zoom
maxLags = 20; %maximum lag for xcorr calculation
[Rx,lags]=xcorr(x,maxLags,'unbiased'); %signal only
[Rz,lags]=xcorr(z,maxLags,'unbiased'); %noise only
[Ry,lags]=xcorr(x+z,maxLags,'unbiased');%noisy signal
subplot(311), stem(lags,Rx); ylabel('R_x[l]');
subplot(312),stem(lags,Rz);ylabel('R_z[l]');
subplot(313),stem(lags,Ry);xlabel('Lag l');ylabel('R_y[l]');
```

The signal-to-noise ratio (SNR) is a common metric consisting of the ratio between the signal and noise power values and often denoted in dB (seem Appendix A.27). Figure 1.24 illustrates the fact that it is hard to visualize the sinusoid because of the relatively low signal-to-noise ratio $\text{SNR}_{\text{dB}} = 10\log_{10}(8/25) \approx -5$ dB. The waveform does not seem to indicate periodicity.

Figure 1.25 shows the autocorrelations of $y[n]$ and its two parcels. For the lag $l = 0$, the estimated values are $R_x[0] = 8$, $R_z[0] = 24.92$ and $R_y[0] = 33.17$. The theoretical values are 8 W (the sine power), 25 W (noise power) and $8 + 25 = 33$ W (sum of the parcels), respectively. The bottom graph clearly exhibits periodicity and the noise disturbs only the value of $R_y[l]$ at $l = 0$. In summary, two assumptions can simplify the analysis of random signals: that the ACF of the noise is approximately an impulse at the origin ($l = 0$) and that the signal and noise are uncorrelated. □

Example 1.18. **The AWGN channel**. Thermal noise is ubiquitous and WGN is often present in communication models. WGN was briefly introduced in Examples 1.15, 1.17 and Application 1.11. In communications (as discussed in Section **??**), distinct models that share the property of having WGN added at the receiver are called AWGN channel models.

Figure 1.26 illustrates a continuous-time AWGN, where the received signal $r(t) = s(t) + \nu(t)$ is simply the sum of the transmitted signal $s(t)$ and WGN $\nu(t)$.

Because $s(t)$ and WGN $\nu(t)$ are often assumed to be uncorrelated, as discussed in Example 1.17, the power of $r(t)$ is the sum of the powers of $s(t)$ and $\nu(t)$. □

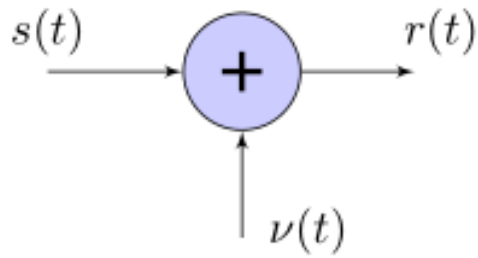


Figure 1.26: *Continuous-time version of the AWGN channel model.*

Having learned about correlation (cross-correlation, etc.), it is possible to study a linear model for quantization, which circumvents dealing with the quantizer non-linearity.

1.14 A Linear Model for Quantization

As discussed, the quantization process is non-linear and the quantization error values depend on the quantizer input. But the following *linear model for the quantization error* has proved to be a good approximation in several applications.

In this model, the error $\mathsf{E}_q = \mathsf{X} - \mathsf{X}_q$ is assumed to be a uniformly distributed random variable with support $[-\Delta/2, \Delta/2]$ and zero-mean. This is sometimes called *quantization noise*. Two assumptions are important:

1. The signal has enough variation to span all levels of the quantizer stair. The model fails, for example, if the input signal is a constant (DC) value.

2. There is no correlation between the error E_q and the signal to be quantized X.

The main assumption is that, for a quantizer with a step of Δ, the quantization noise can be modeled by assuming that $e_q[n]$ is formed by independent and identically distributed (i. i. d.) samples of a random variable E_q with uniform PDF and support $[-\Delta/2, \Delta/2]$. Because it is zero-mean, the power of $e_q[n]$ is solely the variance of the uniform PDF given by Eq. (1.19), which in this case is

$$\sigma^2 = \Delta^2/12. \tag{1.47}$$

Assuming that

$$\Delta = \frac{\hat{X}_{\max} - \hat{X}_{\min}}{2^b},$$

one has that the power $\mathcal{P}_n$ of the quantization noise is

$$\mathcal{P}_n = \sigma^2 = \frac{\Delta^2}{12} = \frac{(\hat{X}_{\max} - \hat{X}_{\min})^2}{2^{2b}12}, \tag{1.48}$$

It is a good idea to practice using the model by calculating the *quantization SNR* for sinusoids and cosines, uniformly and normally distributed random signals. The quantization SNR is obtained by using the quantization noise power in the denominator of the SNR expression (the numerator is the signal power, as usual). The following example illustrates the result for quantizing a Gaussian input signal.

Example 1.19. **Quantization SNR of a Gaussian signal and the 6 dB per bit rule of thumb.** If the signal to be quantized $x(t)$ has samples distributed according to a Gaussian $\mathcal{N}(3, 4)$ with mean $\mu = 3$ V and variance $\sigma^2 = 4$ W, a reasonable alternative (see Eq. (1.15)) is to adopt $\hat{X}_{\min} = \mu - 3\sigma$ and $\hat{X}_{\max} = \mu + 3\sigma$. In this case and using Eq. (1.15),

$$\Delta = \frac{\hat{X}_{\max} - \hat{X}_{\min}}{M-1} = \frac{3+6-(3-6)}{2^b-1} = \frac{12}{2^b-1},$$

where b is the number of bits of the quantizer. Using the linear model of quantization:

$$\text{SNR} = \frac{\mathcal{P}_s}{\mathcal{P}_n} = \frac{3^2+4}{\Delta^2/12} = \frac{13\times 12}{12^2/(2^b-1)^2} = \frac{13(2^b-1)^2}{12} \approx 1.083(2^b-1)^2,$$

where $\mathcal{P}_s$ and $\mathcal{P}_n$ are the signal and noise power, respectively. The SNR_{dB} is

$$\begin{aligned} 10\log_{10}\text{SNR} &= 10[\log_{10}1.0833 + 2\log_{10}(2^b-1)] \\ &\approx 0.3463 + 20b\log_{10}2 \\ &\approx 0.3463 + 6.021b, \end{aligned}$$

where the last step assumed that $2^b \gg 1$.

The result $\text{SNR}_{\text{dB}} \approx 6b + \text{cte.}$ is a well-known "rule of thumb". The constant (cte.) may vary, but the SNR_{dB} typically increases by 6 dB for each extra bit in the quantizer. It is common to use this approximation to suggest, for example, that an ADC of 12 bits has approximately quantization $\text{SNR}_{\text{dB}} = 6\times 12 = 72$ dB, while an ADC of 16 bits has $\text{SNR}_{\text{dB}} = 6\times 16 = 96$ dB. □

Other applications of the main results in this chapter are discussed in the next section.

1.15 Applications

Application 1.1. Recording with a sound board. It is relatively easy to record sound using a microcomputer. However, most softwares that capture sound are not very useful for studying DSP, because they assume the user is not interested in "low-level" details such as the number of bits per sample. But there many alternatives that do provide this kind of information. Two free and open-source (FOSS) softwares for manipulation sound files are Audacity [url1aud] and Sox [url1sox]. While Sox is very useful for converting among file formats and working from a command line, Audacity is adopted here because it has a graphical user interface (GUI) that allows, for example,

to monitor recording and avoid saturating the ADC, which would distort the sound due to clipping of its amplitudes.

Figure 1.27 shows a short segment of audio recorded with the default options of sampling frequency $F_s = 44.1$ kHz and number $b = 32$ bits per sample in floating-point, as indicated by the letter A in the figure. The menu Edit - Preferences - Quality of Audacity allows to change these values. Another option to change F_s is the "Project rate" in letter B of Figure 1.27. The level meters indicated with letter C are activated during recording and playback and, in this case, suggest that the signal amplitude was clipped due to ADC saturation. Alternatively, this can be visualized using menu View - Show Clipping. Each time a new recording starts (by clicking the button indicated by letter D), the audio track has the F_s and b imposed by the current default options. Most sound boards have two channels and can record in stereo but here it is assumed that only one channel is of interest and the files are mono.

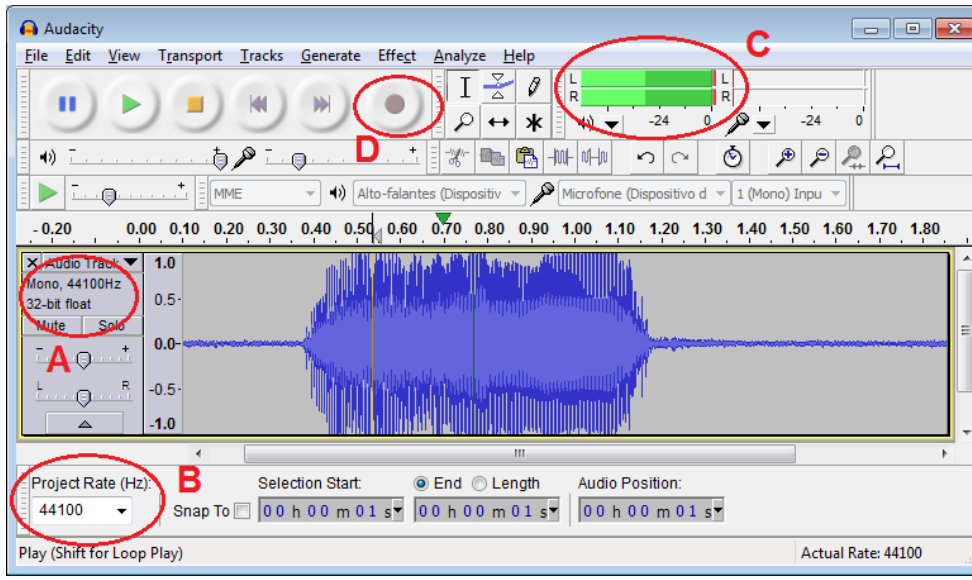


Figure 1.27: *Example of sound recorded at $F_s = 44.1$ kHz with the Audacity sound editor.*

Audacity can save a signal in a variety of file formats, such as MP3, using the menu Export. Our goal is to later read the saved file in another software (Matlab/Octave, etc.), so MP3 should be avoided and of special interest here are "wav" (actually the WAVE format, an instance of Microsoft's RIFF file format) and raw (header-less).

The "wav" format is just a wrapper for many codecs. In other words, within a "wav" file one can find uncompressed data requiring hundreds of kilobits to represent each second (kbps) of audio as well highly compressed data requesting less than five kbps. Unless the file size should be minimized, for increased portability it is better to use an uncompressed "PCM" format. Due to its adoption in digital communications, the result of A/D conversion is sometimes called pulse-coded modulation (PCM). Hence, PCM can be seen as a codec but its output is equivalent to a signal simply sampled at F_s and quantized (or encoded) with b bits/sample. If the adopted quantizer is uniform

(see Eq. (1.14)), the PCM is called linear. The linear PCM is the best format with respect to portability but there are also two popular non-linear PCMs.

Because the probability distribution of long segments of speech signals is approximately Laplacian, not uniform, the quantizer used in digital telephony is non-uniform. These quantizers are based on non-linear curves (approximately logarithmic) called A-law and μ-law. Figure 1.28 shows some options when the user chooses Export - Other compressed files (in "type") and then Options.

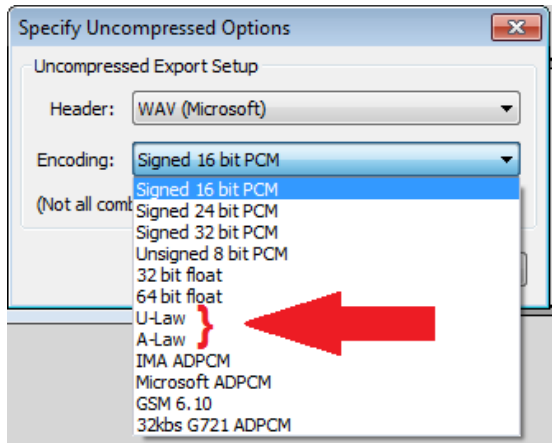


Figure 1.28: *Some Audacity options for saving an uncompressed WAVE file. The two non-linear PCMs are indicated.*

Hence, use Audacity to record some sound with $F_s = 8$ kHz and export it as a file named myvoice.wav in the "WAV (Microsoft) signed 16 bits" format. After that, read it with Matlab/Octave using:

```
[x,Fs,b]=wavread('myvoice.wav');
```

It can be observed that Fs=8000 and b=16, as recorded. Note that by default the wavread function outputs samples in floating-point and normalizes them to be within the range $[-1, 1]$. If the actual integer values are of interest, Matlab allows to use

```
[x2,Fs,b]=wavread('myvoice.wav','native');
```

Using the last two commands and comparing min(x), max(x) and min(x2), max(x2), in the case of a specific audio file, the (native) integer values -8488 (min) and 5877 (max) were normalized to -0.2590 and 0.1794, respectively, when not using the option 'native'. The normalization consists in dividing the native integer values by 2^{b-1}, which takes in account that these values are originally within the range $[-2^{b-1}, 2^b - 1]$. For example, in this case $b = 16$ and $5877/2^{15} \approx 0.1794$.

In case the file had used A-law non-linear PCM, Matlab would give the error message:
Data compression format (CCITT a-law) is not supported.
and Octave:
error: wavread: sample format 0x6 is not supported

Now, it is suggested to get more familiar with headerless files using Audacity to save a sound file as "raw". It may be useful to check Appendix B.5 for more details on how the information is organized in binary files. After recording in Audacity, choose Export - Other compressed files (in "type") as in Figure 1.28, but this time select the header "RAW (header-less)" instead of "WAV (Microsoft)". For the encoding, select "Signed 16 bit-PCM", as before, and name the file 'myvoice_raw.wav'. In this case, it would be wiser to use another file extension and name it 'myvoice.raw', for example. But the purpose of using "wav" is to make clear that the extension by itself cannot guarantee a file format is the expected one.

In this particular example, the file sizes are 29,228 and 29,184 for the WAVE and raw formats, respectively. In fact, in spite of a WAVE possibly having a sophisticated structure with several sections (chunks), most of them have a single chunk and one header consisting of the first 44 bytes, which is the difference between the two sizes given that both have the same $29184/2 = 14592$ samples of 2 bytes each.

Using the command wavread for the raw file would generate error messages in Matlab/Octave. Based on Appendix B.5.4, the following code properly reads the samples:

```
fp=fopen('myvoice_raw.wav','rb'); %open for reading in binary
x=fread(fp,Inf,'int16'); %read all samples as signed 16-bits
fclose(fp);  %close the file
```

As a sanity check, one can read the samples of the WAVE file, skip its header and compare with the result of wavread with Listing 1.14 on Matlab.

Listing 1.14: *MatlabOctaveCodeSnippets/snip_signals_wavread.m*

```
fp=fopen('myvoice.wav','rb'); %open for reading in binary
x=fread(fp,Inf,'int16'); %read all samples as signed 16-bits
fclose(fp);  %close the file
x(1:22)=[]; %eliminate the 44-bytes header
[x2,Fs,b]=wavread('myvoice.wav','native');
x2=double(x2); %convert integer to double for easier manipulation
max(abs(x-x2)) %result is 0, indicating they are identical
```

The advantage of using WAVE is that the header informs F_s, b, whether its mono or stereo, etc. Also, the WAVE format takes care of endianness (see Appendix B.5.3). Not using wavread, write code in Matlab/Octave to open a WAVE file (with only 1 chunk) and extract F_s, b and the samples as integers. This code can be used by Octave users to mimic the option native in Matlab's wavread. It may be useful to read Appendix B.5.5 and use the companion code laps_dump.c, which can be compiled with most C compilers. A short description of the WAVE header is provided at [url1wav]. □

Application 1.2. Recording sound with Matlab. This application discusses how to record sound directly with Matlab, which has several functions to deal with sound recording and playback. You can check soundsc, audiorecorder, wavplay and wavrecord, for example. Some functions work only on Windows.

Octave has functions such as record and sound and its support to sound is more natural on Linux. There are solutions such as [url1rec] to record and play sound on Octave running on Windows, but the installation is not trivial.

The following code was used in Matlab to record 5 seconds of one (mono) channel sound at a sampling rate of 11,025 Hz, using 16 bits to represent each sample:

```
r=audiorecorder(11025,16,1); %create audiorecorder object
record(r,5); %record 5 seconds and store inside object r
```

One can use play(r) to listen the recorded sound or y = getaudiodata(r, 'int16') to obtain the samples from the audiorecorder object. However, if one of these commands immediately follows record(r,5), the error can be generated:
??? Cannot retrieve audio data while recording is in progress.
This means the software was still recording when it tried to execute the second command. An alternative is to use recordblocking as in Listing 1.15.

Listing 1.15: *MatlabOnly/snip_signals_recordblocking.m*

```
r=audiorecorder(11025,16,1); %create audiorecorder object
recordblocking(r,5); %record 5 seconds and store inside r
play(r) %playback the sound
y = getaudiodata(r, 'int16'); %extract samples as int16
plot(y); %show the graph
```

Note that y in the above example is an array with elements of the type int16, i. e., 2 bytes per sample. This saves storage space when compared to the conventional real numbers stored in double (8 bytes) each, but limits the manipulations. For example, the command soundsc(y,11025) generates an error message if y is int16. In such cases, a conversion as y=double(y) can be used before invoking soundsc (use whos y to check that the storage has quadruplicated).

To write y to a 16-bits per sample WAV file and read it back, use Listing 1.16 but the command double(z)./double(y) shows that the normalization used by wavwrite made z approximately three times y. The Voicebox toolbox ([url1voi]) has functions readwav and writewav that are smarter than Matlab's with respect to avoiding the normalization.

Listing 1.16: *MatlabOnly/snip_signals_wavwrite.m*

```
yd=double(y); %convert from int16 (y from getaudiodata)
yd=yd/max(abs(yd)); %need to normalize
wavwrite(yd,11025,16,'somename.wav') %write as 16-bits
z=wavread('somename.wav','native'); %avoid normalization
```

Some sound boards allow full-duplex operation, i. e., recording and playing at the same time. Typically the sampling frequency must be the same for both operations. On Windows one can try the wavplay function with the option "async" as exemplified in Listing 1.17.

Listing 1.17: *MatlabOnly/snip_signals_realtimeLoopback.m*

```
Fs = 11025;   %define sampling rate (Hz)
fc = 1500; %cosine frequency (Hz)
recordingDuration = 1; %duration of recording, in seconds
r=audiorecorder(Fs,16,1);
while 1 %infinite loop, stop with CTRL+C
    recordblocking(r,recordingDuration);
    inputSignal  = getaudiodata(r);
    p=audioplayer(r);
    subplot(211), plot(inputSignal); %graph in time domain
    subplot(212), pwelch(double(inputSignal)); %in frequency domain
    drawnow %Force the graphics to update immediately inside the loop
end
```

Listing 1.17 shows the acquired signal (from the ADC) in both time and frequency domains. In this code, the call to wavplay is non-blocking but samples are lost in the sense that inputSignal is not a perfect cosine. Using a loopback cable, as in Application 1.4, allows to evaluate the system.

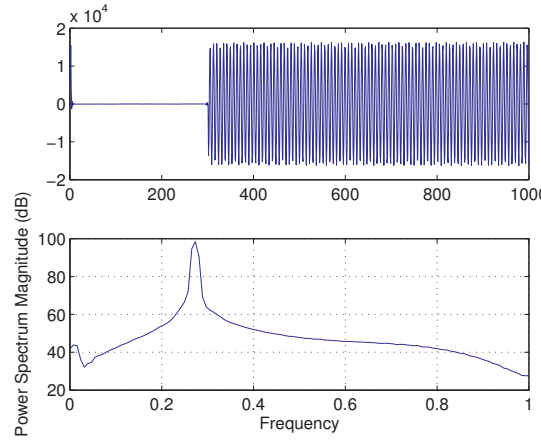


Figure 1.29: *Cosine obtained with Listing 1.17 and a loopback cable connecting the soundboard DAC and ADC.*

Figure 1.29 was obtained with a loopback. Note from the top plot that approximately 300 samples are a transient and after that one can see the cosine at $f_c = 1500$ Hz, which is mapped to $1500/(F_s/2) \approx 0.27$ ($F_s = 11025$ Hz) in the normalized axis of the bottom plot. In order to get this kind of system running, it is important to reduce the volume (DAC gain) to avoid saturation of the signals.

As an exercise, digitize signals at different sampling frequencies and plot them with the axes properly normalized. Another interesting exploration is to obtain a sound signal inputSignal, digitized at a given rate (e. g., $F_s = 11,025$ Hz) and represented by int16. Convert it to double with x=double(inputSignal) in order to easily manipulate the signal and describe what is the result of each of the commands in Listing 1.18.

Listing 1.18: *MatlabOnly/snip_signals_digitize_signals.m*

```
fs=22050; %sampling frequency
r = audiorecorder(fs, 16, 1);%create audiorecorder object
recordblocking(r,5);%record 5 s and store inside object r
y = getaudiodata(r,'int16'); %retrieve samples as int16
x = double(y); %convert from int16 to double
soundsc(x,fs); %play at the sampling frequency
soundsc(x,round(fs/2));%play at half of the sampling freq.
soundsc(x,2*fs); %play at twice the sampling frequency
w=x(1:2:end); %keep only half of the samples
soundsc(w,fs); %play at the original sampling frequency
z=zeros(2*length(x),1); %vector with twice the size of x
z(1:2:end)=x;%copy x into odd elements of z (even are 0)
soundsc(z,fs); %play at the original sampling frequency
```

What should be the sampling frequency for vectors w and z in Listing 1.18 to properly listen the audio? □

Application 1.3. Real time sound processing with Matlab's DSP System Toolbox. Matlab's DSP System Toolbox has extended support to interfacing with the sound board. Listing 1.19 provides a simple example that illustrates recording audio.

Listing 1.19: *MatlabOnly/snip_signals_realtimeWithDspSystem.m*

```
exampleNumber=1; %choose 1 (spectrum analyzer) or 2 (digital filter)
Fs = 8000;   %define sampling rate (Hz)
%create an audio recorder object:
microphone = dsp.AudioRecorder('NumChannels',1,'SampleRate',Fs);
if exampleNumber==1
    specAnalyzer = dsp.SpectrumAnalyzer; %spectrum analyzer object
else
    [B,A]=butter(4,0.05); %4-th order lowpass Butterworth filter
    filterMemory=[]; %initialize the filter's memory
    speaker = dsp.AudioPlayer('SampleRate',Fs); %create audio player
end
disp('Infinite loop, stop with CTRL+C...');
while 1 %infinite loop, stop with CTRL+C
    audio = step(microphone); %record audio
    if exampleNumber==1 %spectrum analyzer
        step(specAnalyzer,audio); %observe audio in frequency domain
    else %perform digital filtering
        [output,filterMemory]=filter(B,A,audio,filterMemory);
        step(speaker, output); %send filtered audio to player
    end
end
```

Listing 1.19 indicates how to plot the signal in frequency domain or to perform digital filtering. The code may drop samples depending on the computer's speed. Matlab's documentation inform how to control the queue and buffer lengths, and also obtain the number of overruns. □

Application 1.4. Estimating latency using the sound board, Audacity and a loopback cable. The goal here is to practice dealing with digital and analog signals, interfacing Audacity and Matlab/Octave. An audio cable and a single computer (with the sound system working) is all that is needed for many interesting experiments. The user is invited to construct or purchase a cable with the proper connectors for his/her computer. In most cases, a "3.5 mm male to 3.5 mm male audio cable" is the required one, as indicated in Figure 1.30. A single channel (mono) cable may suffice but stereo cables have almost the same cost and can be used in more elaborated experiments.[14]

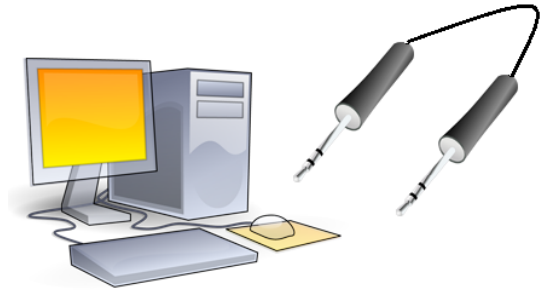


Figure 1.30: *Setup for loopback of the sound system using an audio cable, which is connected to the microphone input (or, alternatively, to the line in) and to the speaker output (or to the line out).*

The task is to estimate the *latency* or *channel delay*, which is the time interval between the signal is transmitted and its arrival at a receiver after passing through a channel. In this specific case, the channel is composed of the sound board hardware (buffers, etc.) and the used device drivers (low-level software that interfaces with the hardware) and application software (Audacity in this case). In sound processing, latency is especially important when *overdubbing*, i. e., recording a track while playing back others. A detailed description of testing the latency with Audacity can be found at [url1lat].

To have a better control of the sound board, it is important to disable all special effects and enhancements for both recording and playback devices, such as automatic gain control for the input ADC signal. Figure 1.31 provides screenshots from Windows but users of other operating systems should be able to find how to choose the best sound options.

After making sure the best configuration for your sound system was chosen, the task now is to generate some samples of a periodic train of impulses. Instead of impulses, the Audacity menu "Generate - Click Track" provides a dialog window with other options. But here the suggestion is to use Matlab/Octave and create a signal with N discrete-time impulses $\delta[n]$. Note that the analog signal corresponding to $\delta[n]$ will never be the theoretical continuous-time $\delta(t)$. For example, assuming zero-order reconstruction (see Figure 1.9), the amplitude of $\delta[n]$ would be held constant during

[14] Note that some computers do not have a sound input connector (microphone or line in) and an external (e. g., USB-based) sound board would be required.

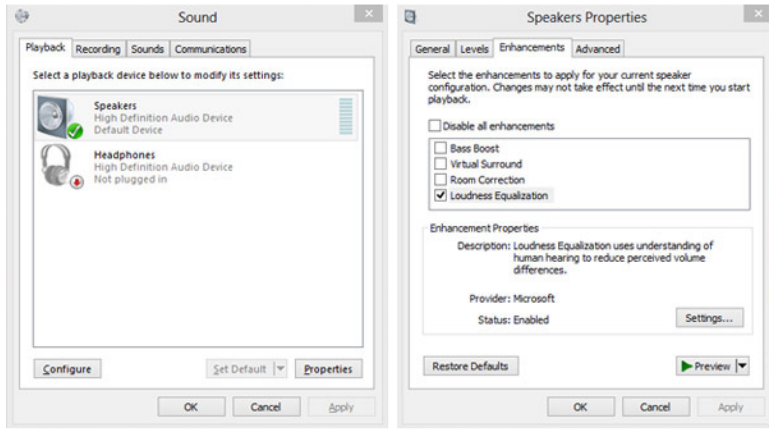


Figure 1.31: *Example of options provided by Windows and the sound board. All the enhancements for both recording and playback devices should be disabled.*

the whole sampling interval T_s. Aware of this limitation, Listing 1.20 generates a train of N discrete-time impulses and saves it to a WAVE file.

Listing 1.20: *MatlabOctaveCodeSnippets/snip_signals_inpulse_train.m*

```
Fs = 44100; %sampling frequency
Ts = 1/Fs; %sampling period
Timpulses = 0.25; %interval between impules in seconds
L=floor(Timpulses/Ts); %number of samples between impulses
N = 4; %number of impulses
impulseTrain=zeros(N*L,1); %allocate space with zeros
b=16; %number of bits per sample
amplitude = 2^(b-1)-1; %impulse amplitude, max signed int
impulseTrain(1:L:end)=amplitude; %generate impulses
wavwrite(impulseTrain,Fs,b,'impulses.wav') %save WAVE RIFF
```

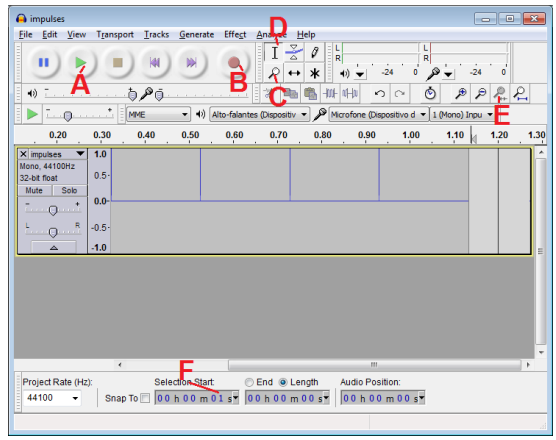


Figure 1.32: *Audacity window after reading in the 'impulses.wav' file.*

Opening the generated file with Audacity should lead to Figure 1.32. Note the amplitudes have been normalized and the first impulse barely appears. In this case, as indicated by letter F in Figure 1.32, the selection region starts approximately at 1 s. The interface is friendly and the letters C and D indicate how to switch between zooming the signal and enabling the cursor, respectively. After a segment is selected, letter E indicates how to easily zoom it to fit the selection. Instead of seconds (in letter F), it is sometimes convenient to use "samples". Using the play button indicated with letter A plays the file.

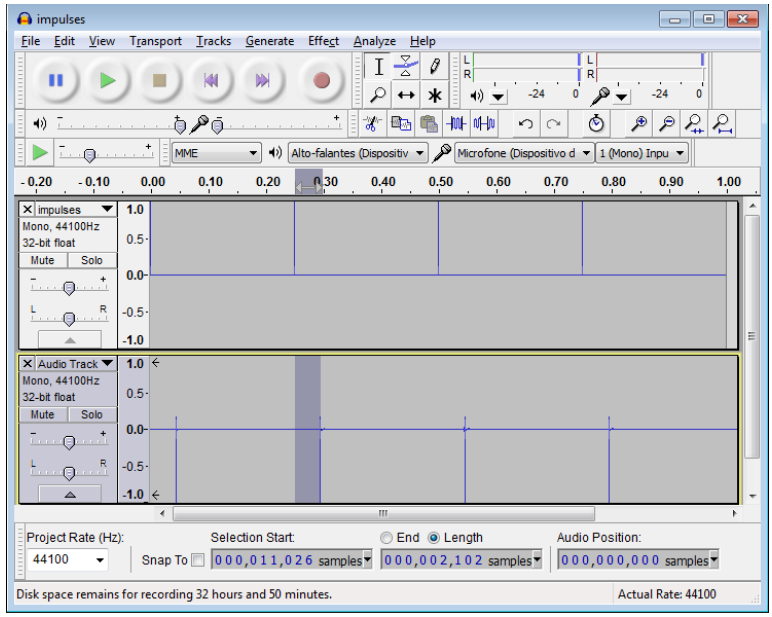


Figure 1.33: *Audacity window after simultaneously recording and playing 'impulses.wav' with a loopback.*

At this point, a feature of Audacity that is useful for overdubbing can be used to simultaneously activate the DAC and ADC: when recording, Audacity also plays all the signals that are "open" (in this case, the "impulses" signal). With the audio cable connected in loopback, start recording (and playback) simply using button (letter B), stopping it after a second. The final situation should be similar to Figure 1.33.

From the code used to generate 'impulses.wav' it can be seen that the impulses are separated by $F_s/4 = 11025$ samples (the first one is at $n = 1$, the second at $n = 11026$ and so on). This information was used to impose the start selection (letter F in Figure 1.32) at sample 11,026 in Figure 1.33 (it is irrelevant here, but recall that the first index in Matlab/Octave is 1 but 0 in Audacity). The end of the selection was located approximately at the start of the second impulse of the recorded signal (bottom plot, identified as "Audio Track"). In this case, the number of samples of this selection indicate that the latency was approximately $2102 \times T_s \approx 47.66$ ms.

At this point it may be useful to export the recorded signal as a WAVE file to be read in Matlab/Octave. First, you can close the window with the "impulses"

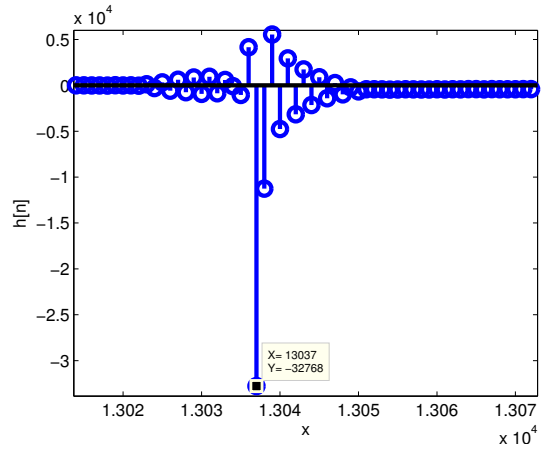


Figure 1.34: *Zoom of the response to the second impulse in Figure 1.32.*

signal (otherwise Audacity will ask if the two files should be merged) and use the "Export" menu. Assuming the output file name was impulseResponses.wav, the command h=wavread('impulseResponses.wav') can be used to generate the zoom of the second impulse response in Figure 1.34. The concept of impulse response is very important, as discussed in Chapter 3.

Because the maximum absolute amplitude occurs at $n = 13037$ in Figure 1.34 and the corresponding impulse is located at $n = 11026$, another estimate of the latency is $(13037 - 11026)T_s \approx 45.6$ ms. A detail is that for creating Figure 1.34, the 'native' option of Matlab's wavread was used to avoid normalization and, consequently, the minimum signal value is $-2^{b-1} = -32768$ not -1. □

Application 1.5. PC sound board quantizer. Given a system with an ADC, typically one has to know beforehand or conduct measurements to obtain the quantizer step size Δ. This is the case when using a personal computer (PC) sound board. For a sound board, the value of Δ depends if the signal was acquired using the microphone input or the line-in input of the sound board. The microphone interface is designed for signals with peak value of $X_{\max} = 10$ to 100 mV while the peak for the line-in is typically 0.2 to 2 V. Note that the voltage ranges of line inputs and microphones vary from card to card. See more details in [url1pcs]. For the sake of this discussion, one can assume a dynamic range of $[-100, 100]$ mV and a ADC of 8 bits per sample, such that $\Delta = 200/(2^8 - 1) \approx 0.78$ mV. The following example illustrates how to approximately recover the analog signal for visualization purposes. Assume the digital dynamic range is [0, 255] and the digital samples are $D = [13, 126, 3, 34, 254]$. If one simply uses stem(D), there is no information about time and amplitude. Listing 1.21 shows the necessary normalizations to visualize the abscissa in seconds and the ordinate in Volts, which in this case corresponds to A=1000*[-89.70, -1.56, -97.50, -73.32, 98.28].

Listing 1.21: *MatlabOctaveCodeSnippets/snip_signals_amplitude_normalization.m*

```
D=[13 126 3 34 254]; %signal as 8-bits unsigned [0, 255]
n=[0:4]; %sample instants in the digital domain
Fs=8000; %sampling frequency
delta=0.78e-3; %step size in Volts
A=(D-128)*delta; %subtract offset=128 and normalize by delta
Ts=1/Fs; %sampling interval in seconds
time=n*Ts; %normalize abscissa
stem(time,A); %compare with stem(n,D)
xlabel('time (s)'); ylabel('amplitude (V)');
```

Now assume a PC computer with a sound board that uses a 16 bits ADC and supports at its input a dynamic range of -185 to 185 mV. The quantizer is similar to the one depicted in Figure 1.11, but the quantization step should be $\Delta = 2 \times 185 \times 10^{-3}/2^{16} \approx 5.6\ \mu$V. It is assumed here that $\Delta = 5.6\ \mu$V and the quantizer is uniform from $-2^{15}\Delta$ to $(2^{15}-1)\Delta$. In this case, the $M = 65,536 = 2^{16}$ levels are organized as 32,767 positive levels, 32,768 negative levels and one level representing zero. The assumed coding scheme is the offset code of Table 1.5 with 32,768 as the offset. Hence, the smallest value $-2^{15}\Delta$ is mapped to the 16-bits codeword "0000 0000 0000 0000", $(-2^{15}+1)\Delta$ to "0000 0000 0000 0001" and so on, with $(2^{15}-1)\Delta$ being coded as "1111 1111 1111 1111".

If at a specific time t_0 the ADC input is $x(t_0) = x = 0.003$ V, the ADC output is $x_i = 536$, which corresponds to $x_q = 0.0030016$ V and leads to a quantization error $e = x - x_q \approx -1.6 \times 10^{-6}$ V. These results can be obtained in Matlab/Octave with

```
delta=5.6e-6, b=16 %define parameters for the quantizer
format long %see numbers with many decimals
x=3e-3; [xq,xi]=ak_quantizer(x,delta,b), error=x-xq
```

Based on similar reasoning, calculate the outputs x_i of the quantizer, their respective x_q values and the quantization error for $x \in \{-300, -100, 0, 20, 180\}$ mV.

If you have access to an oscilloscope and a function generator, try to estimate the value of Δ of your sound board, paying attention to the fact that some software/hardware combination use automatic gain control (AGC). You probably need to disable AGC to better control the acquisition.

It is not trivial, but if you want to learn more about your sound board, try to evaluate its performance according to the procedure described at [url1bau]. □

Application 1.6. Time-reversal of sounds. The operation $y[n] = x[-n]$ corresponds to flipping the signal over the ordinate axis. In Matlab/Octave it can be implemented with `fliplr` (left-right) or `flipud` (up-down) for row and column vectors, respectively. For example, assume that $x[n] = 3\delta[n] + 4\delta[n-2] + 5\delta[n-4]$. One can represent $x[n]$ as the row vector `x=[3 0 4 0 5]` and flip it using `y=fliplr(x)`. It is important to note that, when representing a signal, one vector can store the amplitude values but not their respective time instants. An additional vector is required to store the time information. Listing 1.22 is a snippet (part of the script) used to generate Figure 1.35. It illustrates the care that must be exercised to properly represent the signals $y[n] = x[-n]$ using a computer program. Note that the user must relate the amplitude vector and the "time" vector.

Listing 1.22: *MatlabOctaveCodeSnippets/snip_signals_timereversal.m*

```
x=[3 0 4 0 5]; %some signal samples
y=fliplr(x); %time-reversal
n1=0:4; %the 'time' axis
n2=-4:0; %the 'time-reversed' axis
subplot(211); stem(n1,x); title('x[n]');
subplot(212); stem(n2,y); title('y[n]');
```

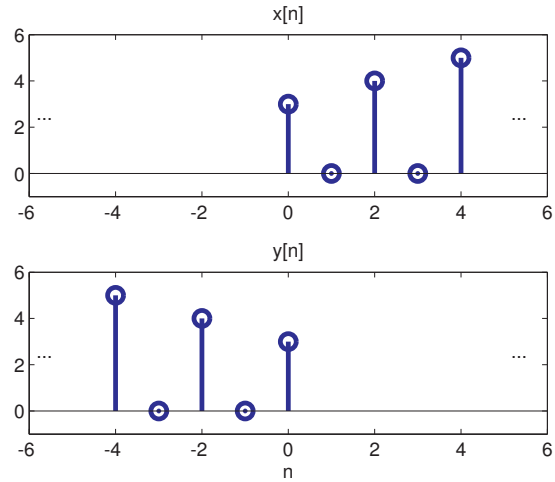


Figure 1.35: *Representation of signals related by* $y[n] = x[-n]$*, where* $x[n] = 3\delta[n] + 4\delta[n-2] + 5\delta[n-4]$.

Figure 1.35 was produced using three dots to indicate that the signals have infinite duration in spite of being represented by finite-length vectors. Another convention is that, when an amplitude value is not explicitly shown, it is assumed to be zero.

Extend the exercise by recording sound and representing it as a signal $x[n]$. You may use palindromes,[15] which are words or phrases which read the same in both directions. For example, record $x[n]$ for the word "deed" and listen to $x[-n]$. Compare the waveforms of $x[n]$ and $x[-n]$. You may find useful the specgram function in Matlab/Octave to obtain an alternate view of the signals. □

Application 1.7. Using rat in Matlab/Octave to find the period of discrete-time sinusoids. The Matlab/Octave function rat for rational fraction approximation can be used for finding m and N. But care must be exercised because rat approximates the input argument within a given tolerance. The code below illustrates how this function can be used to obtain m and N:

```
w=3*pi/5 %define some angular frequency (rad)
[m,N]=rat(w/(2*pi)) %find m and N
```

[15] On the Web, you can find palindromes such as "A Man, A Plan, A Canal: Panama" or "God saw I was dog".

In this case, the result is m=3, N=10, as expected. However, note that w=0.2,[m,N]=rat(w/(2*pi)) returns m=113, N=3550, which is not precise (recall that if $\Omega = 0.2$ the sinusoid is non-periodic). Modifying the previous command to use a smaller tolerance w=0.2,[m,N]=rat(w/(2*pi),1e-300) gives much larger values for m,N, which clearly indicates that the user must be aware that rat uses approximations. Make sure you can generate discrete-time sinusoids with distinct values of m and N and understand the roles played by these two values. □

Application 1.8. Power of the sum of two signals. Assume a signal $z[n] = x[n] + y[n]$ is generated by summing two real signals (similar result can be obtained for complex-valued signals) $x[n]$ and $y[n]$ with power $\mathcal{P}_x$ and $\mathcal{P}_y$. The question is: What is the condition for having $\mathcal{P}_z = \mathcal{P}_x + \mathcal{P}_y$?

Assuming the two signals are random and using expected values (a similar result would hold for deterministic signals):

$$\mathcal{P}_z = \mathbb{E}[\mathsf{Z}^2] = \mathbb{E}[(\mathsf{X}+\mathsf{Y})^2] = \mathcal{P}_x + \mathcal{P}_y + 2\mathbb{E}[\mathsf{XY}]. \tag{1.49}$$

If X and Y are uncorrelated, i. e., $\mathbb{E}[\mathsf{XY}] = \mathbb{E}[\mathsf{X}]\mathbb{E}[\mathsf{Y}]$ and at least one signal is zero-mean, Eq. (1.49) simplifies to

$$\mathcal{P}_z = \mathcal{P}_x + \mathcal{P}_y. \tag{1.50}$$

This is a useful result for analyzing communication channels that model the noise as additive. These models assume the noise is uncorrelated to the transmitted signal and Eq. (1.50) applies. □

Application 1.9. Estimate the PDF of speech signals. Via a normalized histogram, estimate the PDF of a speech signal with a long duration. After this estimation, you should convince yourself that uniform quantizers are not adequate for speech signals. In fact, when using a non-linear quantizer based on the A-law or μ-law, it is possible to use only 8 bits to achieve the subjective quality of a linear PCM with 12 bits. Observe whether or not your histogram approaches a Laplacian density, as suggested by previous research in speech coding. □

Application 1.10. A simple application of correlation analysis. A company produces three distinct beauty creams: A, B and C. The task is the analysis of correlation in three databases, one for each product. The contents of each database can be represented by two vectors $\mathbf{x}$ and $\mathbf{y}$, with 1,000 elements each. Vector $\mathbf{x}$ informs the age of the consumer and $\mathbf{y}$ the number of his/her purchases of the respective cream (A, B or C) during one year, respectively. Figure 1.36 depicts scatter plots corresponding to each product.

The empirical (the one calculated from the available data) covariance matrices and means were approximately the following: $\mathbf{C}_a = \begin{bmatrix} 4.08 & -0.002 \\ -0.002 & 0.98 \end{bmatrix}$ and $\mu_a = [30.0, 6.05]$,

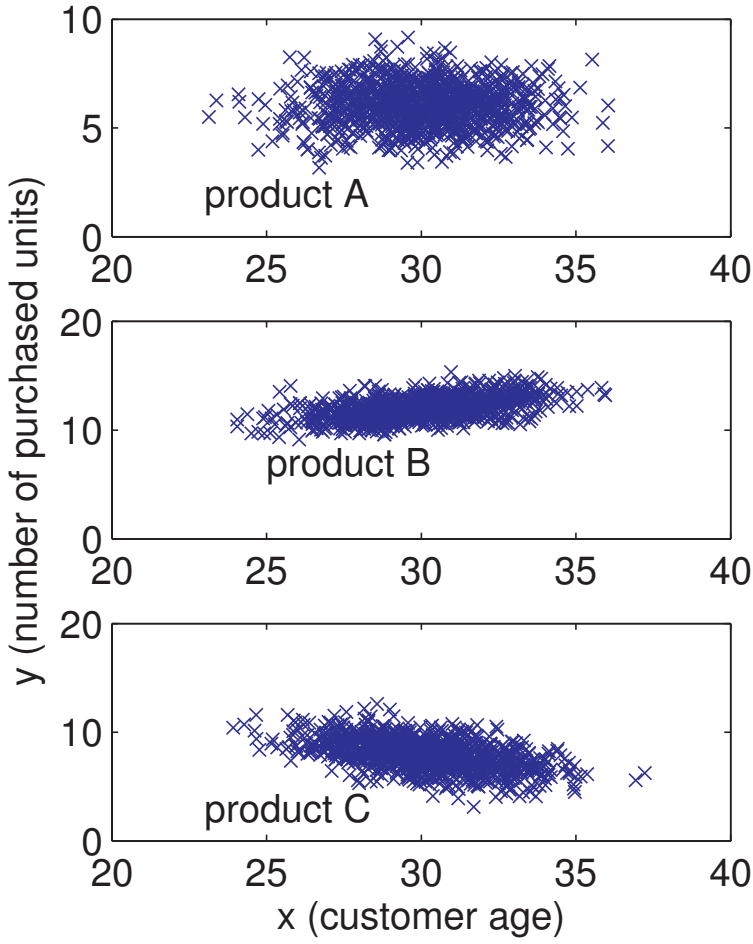


Figure 1.36: *Scatter plot of customer age versus purchased units for three products. These two variables present positive correlation for product B, negative for C and are uncorrelated for product A.*

$\mathbf{C}_b = \begin{bmatrix} 4.00 & 0.99 \\ 0.99 & 1.00 \end{bmatrix}$ and $\mu_b = [30.1, 12.0]$, $\mathbf{C}_c = \begin{bmatrix} 4.16 & -1.46 \\ -1.46 & 2.02 \end{bmatrix}$ and $\mu_c = [30.13, 7.99]$. The correlation coefficients are $\rho_a = -0.0011$, $\rho_b = 0.4924$ and $\rho_c = -0.5031$.

The plots and correlation coefficients indicate that when age increases, the sales of product B also increases (positive correlation). In contrast, the negative correlation of ρ_b indicates that the sales of product C decreases among older people. The sales of product A seem uncorrelated with age. The script figs_signals_correlationcoeff.m allows to study how the data and figures were created. Your task is to learn how to generate two-dimensional Gaussians with arbitrary covariance matrices.

Note that the correlation analysis was performed observing each product sales individually. You can assume the existence of a unique database, where each entry has four fields: age, sales of A, B and C. What kind of analysis do you foresee? For

example, one could try a marketing campaign that combines two product if their sales are correlated. Or even use data mining tools to extract association rules that indicate how to organize the marketing. □

Application 1.11. Playing with the autocorrelation function of white noise and sinusoids. Using randn in Matlab/Octave, generate a vector corresponding to a realization of a WGN process (see Example 1.15): x=randn(1,1000). Check whether or not it is Gaussian by estimating the FDP (use hist). Plot its autocorrelation with the proper axes. Generate a new signal that is uniformly distributed: y=rand(1,1000)-0.5; and plot the same graphs as for the Gaussian signal. What does it happen with the autocorrelation if you add a DC level (add a constant to x and y)? And what if you multiply by a number (a "gain")? Generate a cosine T=0.01; t=0:T:10-T; z=cos(2*pi*10*t); of 10 Hz with a sampling frequency of 100 Hz. Take a look at the autocorrelation for lags from $m = -30, \ldots, 30$ with [c,lags]=xcorr(z,30,'biased');plot(lags,c). Compare this last plot with a zoom of the cosine: plot(z(1:30)). Note that they have the same period. In fact, an autocorrelation R of x incorporates all the periodicity that is found in x as indicated by Eq. (1.44). Make sure you can use Eq. (1.44) to predict the plots you obtain with xcorr when the signal is a sinusoid. □

Application 1.12. Using autocorrelation to estimate the cycle of sunspot activity. The international sunspot number (also known as the Wolfer number) is a quantity that simultaneously measures the number and size of sunspots. A sunspot is a region on the Sun's surface that is visible as dark spots. The number of sunspots correlates with the intensity of solar radiation: more sunspots means a brighter sun. This number has been collected and tabulated by researchers for around 300 years. They have found that sunspot activity is cyclical and reaches its maximum around every 9.5 to 11 years (in average, 10.4883 years).[16] The autocorrelation can provide such estimate as indicated by the script below. Note that we are not interested in $R(0)$, which is always the maximum value of $R(\tau)$. The lag of the largest absolute value of $R(\tau)$ other than $R(0)$ indicates the signal fundamental period. Because theoretically no other value can be larger than $R(0)$, the task of automatically finding the second peak (not the second largest sample), which is the one of interest, is not trivial. The code snippet below simply (not automatically) indicates the position of the second peak for the sunspot data.

Listing 1.23: *MatlabOctaveCodeSnippets/snip_signals_peak_detection.m*

```
load sunspot.dat; %the data file
year=sunspot(:,1); %first column
wolfer=sunspot(:,2); %second column
%plot(year,wolfer); title('Sunspot Data') %plot raw data
x=wolfer-mean(wolfer); %remove mean
[R,lag]=xcorr(x); %calculate autocorrelation
plot(lag,R); hold on;
```

[16] See, e. g., [url1sun] for more information. Note also that Matlab has a script sunspots.m that works in the frequency domain.

```
index=find(lag==11); %we know the 2nd peak is lag=11
plot(lag(index),R(index),'r.', 'MarkerSize',25);
text(lag(index)+10,R(index),['2nd peak at lag=11']);
```

Figure 1.37 shows the graph generated by the companion script figs_signals_correlation.m. It complements the previous code snippet, showing how to extract the second peak automatically (this can be useful in other applications of the ACF). Your task is to study this code and get prepared to work with "pitch" estimation in Application 1.13.

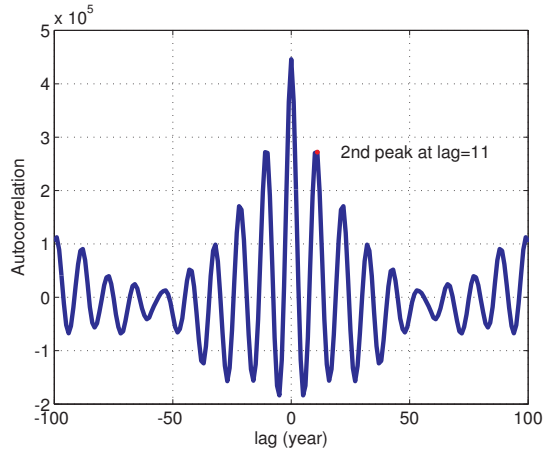


Figure 1.37: *Autocorrelation of the sunspot data.*

An important aspect of the sunspot task is the interpretation of $R(\tau)$. As discussed, when the autocorrelation has a peak, it is an indication of high similarity, i. e., periodicity. In the sunspot application, the interval between two lags was one year. If the ACF is obtained from a signal sampled at F_s Hz, this interval between lags is the sampling period $T_s = 1/F_s$ and it is relatively easy to normalize the lag axis. The next example illustrates the procedure. □

Application 1.13. Using autocorrelation to estimate the "pitch". This application studies a procedure to record speech, estimate the average fundamental frequency $F0$ (also erroneously but commonly called *pitch*) via autocorrelation and play a sinusoid with a frequency proportional to $F0$.

One can estimate the fundamental frequency of a speech signal by looking for a peak in the delay interval corresponding to the normal pitch range in speech.[17] The following script illustrates the procedure.

Listing 1.24: *MatlabOctaveCodeSnippets/snip_signals_fundamental_frequency.m*

[17] Adult male speakers have a pitch range typically in the range [110, 130] Hz, while females have pitch in [200, 230] Hz. You may get an idea if you like Hollywood: it has been said that Sean Connery, Mel Gibson, Barbra Streisand and Julia Roberts have an average pitch of 158, 108, 228 and 171 Hz, respectively.

```
Fs=44100; %sampling frequency
Ts=1/Fs; %sampling interval
minF0Frequency=80; %minimum F0 frequency in Hz
maxF0Frequency=300; %minimum F0 frequency in Hz
minF0Period = 1/minF0Frequency; %correponding F0 (sec)
maxF0Period = 1/maxF0Frequency; %correponding F0 (sec)
Nbegin=round(maxF0Period/Ts);%number of lags for max freq.
Nend=round(minF0Period/Ts); %number of lags for min freq.
if 0 %record sound or test with 300 Hz cosine
    r = audiorecorder(Fs, 16, 1);%object audiorecorder
    disp('Started recording. Say a vowel a, e, i, o or u')
    recordblocking(r,2);%record 2 s and store in object r
    disp('finished recording');
    y=double(getaudiodata(r, 'int16'));%get recorded data
else %test with a cosine
    y=cos(2*pi*300*[0:2*Fs-1]*Ts); %300 Hz, duration 2 secs
end
subplot(211); plot(Ts*[0:length(y)-1],y);
xlabel('time (s)'); ylabel('Signal y(t)')
[R,lags]=xcorr(y,Nend,'biased'); %ACF with max lag Nend
subplot(212); %autocorrelation with normalized abscissa
plot(lags*Ts,R); xlabel('lag (s)');
ylabel('Autocorrelation of y(t)')
firstIndex = find(lags==Nbegin); %find index of lag
Rpartial = R(firstIndex:end); %just the region of interest
[Rmax, relative_index_max]=max(Rpartial);
%Rpartial was just part of R, so recalculate the index:
index_max = firstIndex - 1 + relative_index_max;
lag_max = lags(index_max); %get lag corresponding to index
hold on; %show the point:
plot(lag_max*Ts,Rmax,'xr','markersize',20);
F0 = 1/(lag_max*Ts); %estimated F0 frequency (Hz)
fprintf('Rmax=%g lag_max=%g T=%g (s) Freq.=%g Hz\n',...
    Rmax,lag_max,lag_max*Ts,F0);
t=0:Ts:2; soundsc(cos(2*pi*3*F0*t),Fs); %play freq. 3*F0
```

Figure 1.38 was generated using the previous script with the signal $y(t)$ consisting of a cosine of 300 Hz instead of digitized speech (simply change the logical condition of the "if").

The code outputs the following results:

Rmax=0.49913 lag_max=147 T=0.00333333 (sec) Frequency=300 Hz

Note that the autocorrelation was normalized xcorr(y,Nend,'biased'), which led to $R(0) \approx 0.5$ Watts, coinciding with the sinusoid power $A^2/2$, where $A = 1$ V is the sinusoid amplitude.

As commonly done, in spite of dealing with discrete-time signals, the graphs assume the signals are approximating a continuous-time signal and ACF. Hence, the abscissa is t, not n.

Some PC sound boards heavily attenuate the signals around 100 Hz. Therefore, the last command multiplies the estimated $F0$ by 3, to provide a more audible tone. Modify the last line of the code to use $F0$ instead of $3F0$ and observe the result of varying F0 with your own voice. Then try to improve the code to create your own

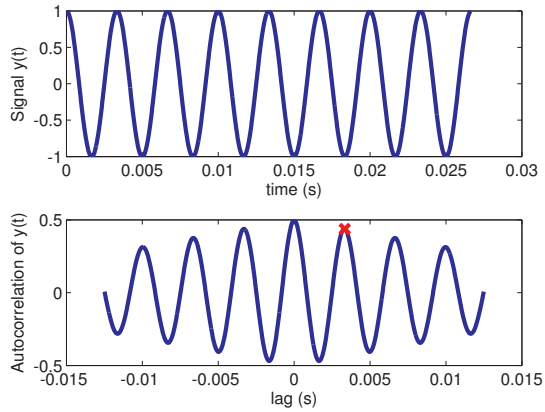


Figure 1.38: *Autocorrelation of a cosine of 300 Hz. The autocorrelation is also periodic. The period in terms of lags is 1/300 s.*

F0 estimation algorithm. Find on the Web a Matlab/Octave F0 (or pitch) estimation algorithm to be the baseline (there is one at [url1pit]) and compare it with your code. Use the same input files for a fair comparison and superimpose the pitch tracks to spectrograms to better compare them. If you enjoy speech processing, try to get familiar with Praat [url1pra] and similar softwares and compare their F0 estimations with yours. □

Application 1.14. Using cross-correlation for synchronization of two signals or time-alignment. Assume a discrete-time sinal $x[n]$ is transmitted through a communication channel and the receiver obtains a delayed and distorted version $y[n]$. The task is to estimate the delay imposed by the channel. The transmitter does not "stamp" the time when the transmission starts, but uses a predefined preamble sequence $p[n]$ that is known by the receiver. The receiver will then guess the beginning of the transmitted message by searching for the preamble sequence in $y[n]$ via cross-correlation. Before trying an example that pretends to be realistic, some simple manipulations can clarify the procedure.

Assume we want to align the signal $x[n] = \delta[n] + 2\delta[n-1] + 3\delta[n-2]$ with $y[n] = 3\delta[n] + 2\delta[n-1] + \delta[n-2] + \delta[n-3] + 2\delta[n-4] + 2\delta[n-5]$. Intuitively, the signal $x[n-3]$ is a good match to $y[n]$. Alternatively, $y[n+3]$ matches $x[n]$. The Matlab/Octave command `xcorr(x,y)` for cross-correlation can help finding the best lag L such that $x[n+L]$ matches $y[n]$. The procedure is illustrated below:

Listing 1.25: *MatlabOctaveCodeSnippets/snip_signals_cross_correlation.m*

```
x=1:3; %some signal
y=[(3:-1:1) x]; %the other signal
[c,lags]=xcorr(x,y); %find cross-correlation
max(c) %show the maximum cross-correlation value
L = lags(find(c==max(c))) %lag for max cross-correlation
stem(lags,c); %plot
```

```
xlabel('lag (samples)'); ylabel('cross-correlation')
```

The result is $L = -3$. If the order is swapped to xcorr(y,x) as below, the result is $L = 3$.

```
[c,lags]=xcorr(y,x);
max(c) %show the maximum cross-correlation value
maxlag=lags(find(c==max(c))) %max cross-correlation y,x lag
```

It should be noticed that the cross-correlation is far from perfect with respect to capturing similarity between waveforms. For example, if $y[n]$ is changed to y=[(4:-1:1) x], the previous commands would indicate the best lag as $L = 1$. The reader is invited to play with simple signals and find more evidence of this limitation. As a rule of thumb, the cross-correlation will work well if one of the signals is a delayed version of the other, without significant distortion. However, in situations such as reverberant rooms where one of the signals is composed by a sum of multi-path (with distinct delays) versions of the other signal, more sophisticated techniques should be used.

Another aspect is that, in some applications, the best similarity measure is the absolute value of the cross-correlation (i. e., L = lags(find(abs(c)==max(abs(c))))) instead of L = lags(find(c==max(c)))). For example, this is the case when $x[n]$ can be compared either to $y[n]$ or $-y[n]$.

Listing 1.26 illustrates the delay estimation between two signals $x[n]$ and $y[n]$. The vector y, representing $y[n]$, is obtained by delaying x and adding Gaussian noise to have a given SNR.

Listing 1.26: *MatlabOctaveCodeSnippets/snip_signals_time_delay.m*

```
Fs=8000; %sampling frequency
Ts=1/Fs; %sampling interval
N=1.5*Fs; %1.5 seconds
t=[0:N-1]*Ts;
if 1
    x = rand(1,N)-0.5; %zero mean uniformly distributed
else
    x = cos(2*pi*100*t); %cosine
end
delayInSamples=2000;
timeDelay = delayInSamples*Ts %delay in seconds
y=[zeros(1,delayInSamples) x(1:end-delayInSamples)];
SNRdb=10; %specified SNR
signalPower=mean(x.^2);
noisePower=signalPower/(10^(SNRdb/10));
noise=sqrt(noisePower)*randn(size(y));
y=y+noise;
subplot(211); plot(t,x,t,y);
[c,lags]=xcorr(x,y); %find crosscorrelation
subplot(212); plot(lags*Ts,c);
%find the lag for maximum absolute crosscorrelation:
L = lags(find(abs(c)==max(abs(c))));
estimatedTimeDelay = L*Ts
```

Figure 1.39 illustrates the result of running the previous code. The random signals have zero mean and uniformly distributed samples. The estimated delay via xcorr(x,y) was -0.25 s. The negative value indicates that x is advanced with respect to y. The command xcorr(y,x) would lead to a positive delay of 0.25 s.

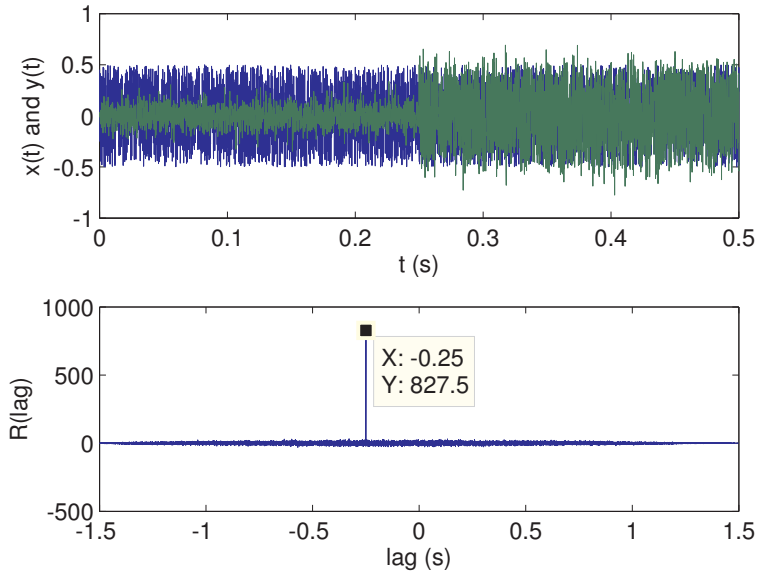


Figure 1.39: *First graph shows signals $x(t)$ and $y(t-0.25)$ contaminated by AWGN at an SNR of 10 dB. The signal $y(t)$ can be identified by a smaller amplitude in the beginning, where its samples are due to noise only. The second graph shows their cross-correlation $R_{XY}(\tau)$ indicating the delay of -0.25 s.*

After running the code as it is, observe what happens if x=rand(1,N), i. e., use a signal with a mean different than zero (0.5, in this case). In this case, the correlation is affected in a way that the peak indicating the delay is less pronounced. Another test is to use a cosine (modify the if) with delayInSamples assuming a small value with respect to the total length N of the vectors. The estimation can fail, indicating the delay to be zero. Another parameter to play with is the SNR. Use values smaller than 10 dB to visualize how the correlation can be useful even with negative SNR.

It is important to address another issue: comparing vectors of different length. Assume two signals $x[n]$ and $y[n]$ should be aligned in time and then compared sample-by-sample, for example to calculate the error $x[n]-y[n]$. There is a small problem for Matlab/Octave if the vectors have a different length. Assuming that xcorr(x,y) indicated the best lag is positive ($L>0$), an useful post-processing for comparing $x[n]$ and $y[n]$ is to delete samples of $x[n]$. If L is negative, the first samples of $y[n]$ can be deleted. Listing 1.27 illustrates the operation and makes sure that the vectors representing the signals have the same length.

Listing 1.27: *MatlabOctaveCodeSnippets/snip_signals_time_aligment.m*

```
x=[1 -2 3 4 5 -1]; %some signal
y=[3 1 -2 -2 1 -4 -3 -5 -10]; %the other signal
[c,lags]=xcorr(x,y); %find crosscorrelation
L = lags(find(abs(c)==max(abs(c)))); %lag for the maximum
if L>0
  x(1:L)=[]; %delete first L samples from x
else
  y(1:-L)=[]; %delete first L samples from y
end
if length(x) > length(y) %make sure lengths are the same
  x=x(1:length(y));
else
  y=y(1:length(x));
end
plot(x-y); title('error between aligned x and y');
```

Elaborate and execute the following experiment: record two utterances of the same word, storing the first in a vector `x` and the second in `y`. Align the two signals via cross-correlation and calculate the mean-squared error (MSE) between them (for the MSE calculation it may be necessary to have vectors of the same length, as discussed). □

1.16 Comments and Further Reading

Signal processing books sometimes target two distinct audiences. The most introductory treatments are typically adopted for "Signals and Systems" courses, while more advanced textbooks target "Digital Signal Processing" courses. This division can be observed in the two classic books [OWH96] and [OS09]. It is also used in the Schaum's Outline Series (well-known for the many fully solved problems). Regarding the term digital or discrete-time: the jargon is such that people call digital signal processing (DSP) many operations that should be considered discrete-time signal processing given the amplitude is not quantized.[18]

There are many great books on DSP and three of them are (always check for new editions): [DdSN10, Lyo10, Mit10].

When reading DSP and telecommunication books, keep in mind that the taxonomy of signals adopted here is not the only one used. For example, books such as [Pee86, Lat78] define digital signals as the ones with quantized amplitudes. Their alternative definition considers that $x_q(t)$, a continuous-time signal, is digital because of its quantized amplitudes.

Topics such as sampling and quantization have been widely investigated. The sampling theorem is part of a broad area. Only periodic sampling was discussed here. There are many other sampling theorems, addressing issues such as non-uniform sampling and non-bandlimited signals. See, e. g., [Mar01, BF01]. In [Pee86] a whole chapter is dedicated to sampling. As a side note: the sampling theorem is related to the work by Harry Nyquist and, therefore, called Nyquist theorem by some authors.

[18] This is a possible reason for the change in the book title "Digital signal processing" [Opp75] to "Discrete-time signal processing" [OS09].

However, Nyquist did not explicitly consider the problem of sampling and reconstruction of continuous signals. The credits for the sampling theorem go to C. Shannon, V. Kotelnikov, E. Whittaker and others. See, for example, [BS92, Luk99], for historical information.

Quantization is also part of a vast area known as source coding. A classical and good book is [JN84]. A modern treatment can be found in [AM07].

Regarding quantization, it is important to warn the experienced reader that in this text, unless otherwise stated (e. g., when discussing PAM decoding), it is assumed the quantizers are uniform and *mid-tread*. Non-uniform and *mid-riser* quantizers (see [JN84]) are not discussed.

Some authors prefer to reserve the unit Hertz (Hz) to only represent the oscillation rate of signals, electrical fields, etc., sticking with its historical use. In this case, for example, an ADC sampling frequency F_s should not be denoted by Hz, but samples per second (SPS) for example. In this text, Hz is "overloaded" to be a unit that can account for everything repeating a given number of times per second and F_s is reported in Hz.

1.17 Review Exercises

If you are familiar with the material, you probably want to skip this section and check the exercises in Section 1.18.

1.1. One has a discrete-time sinusoid with amplitude $A = 10$ Volts and wants to generate discrete-time AWGN with power P such that the SNR equals 30 dB. What is the value of P in Watts? What are the Matlab/Octave commands to generate both signals?

1.2. Calculate by hand and plot the signal $x[n] = Ce^{an}$ for $n = 0, 1, \ldots, 3$, where $C = 3 + j4$ and $a = 2 + j\pi$.

1.3. What is the amplitude of a sinusoid $x(t)$ with 30 dBm of power? And the amplitude of another sinusoid $y(t)$ with 30 dB more power than $x(t)$? Is it correct to say that a signal has 30 dB of power? You may want to read Appendix A.27 for a discussion about dB.

1.4. Assume that X is a discrete random variable (r.v.) with V distinct values. Explain how to calculate its mean $\mathbb{E}[\mathsf{X}]$ from

$$\mathbb{E}[\mathsf{X}] = \sum_{i=1}^{V} p_i x_i,$$

where p_i is the probability of the i-th possible value x_i. Compare this procedure with the alternative calculation that sums N values and divides by N. Interpret the equation $\mathbb{E}[\mathsf{X}]$ when $\mathsf{X} \in \mathbb{R}$, i. e., X is a continuous r.v.

1.5. To review the alternatives to represent complex numbers, let $a = 4e^{j0}$ and $b = 3e^{j\pi/2}$ be two complex numbers and calculate: $c = ab$, $d = a + b$, $|c|$ and $\angle c$. Note

that the following two forms of representing complex numbers in the polar notation are equivalent: $me^{j\Theta}$ and $m\angle\Theta$.

1.6. Describe in details the header and summarize the contents (list all the information you could extract from the header such as number of samples, sampling frequency, etc.) of the following companion files: tidigits_saa7.wav, timit_testdr8mpam0sx199.raw and timit_testdr8mpam0sx199.wav. These are files that store speech waveforms (TIDIGITS and TIMIT are famous corpora distributed by LDC – [url1ldc]). Choose one of these files and analyze the corresponding signal via plots (i. e., show the waveform with the abscissa and ordinate properly labeled, and other graphs of interest such as the histogram and PSD).

1.7. Using your favorite programming language, list the code for reading raw (without header) files storing two kinds of data: a) floats (4 bytes) in little-endian format and b) shorts (2 bytes) in big-endian format.

1.8. Assume the random vector

$$\mathsf{X} = [3; 2; 3; 2; 2; 0; 1; 0; 0; 3; 0; 2; 3; 2; 2; 3; 3; 0; 3; 2; 0].$$

a) Calculate its histogram. b) Estimate its probability mass function (PMF). c) Calculate the moments: mean $\mathbb{E}[\mathsf{X}]$, variance $\mathbb{E}[(\mathsf{X} - \mu_x)^2]$ and $\mathbb{E}[\mathsf{X}^2]$. Estimate its entropy $H = \mathbb{E}[I_i]$, where $I_i = 1/p_i$ is the information associated to the i-th distinct value and p_i the probability of this value. Note that $H = \sum_i p_i I_i$.

1.9. For calculating the variance using its definition $\mathbb{E}[(\mathsf{X} - \mu_x)^2]$ one has to go twice over the data samples. The first loop obtains the mean μ_x and the second loop calculates $\mathbb{E}[(\mathsf{X} - \mu_x)^2]$. Show a code that uses only one pass over the data by adopting the expression $\mathbb{E}[(\mathsf{X} - \mu_x)^2] = \mathbb{E}[\mathsf{X}^2] - \mu_x^2$.

1.10. a) Generate a realization (one waveform) of a Gaussian random process with 100 i.i.d. (independent and identically distributed) samples of a Gaussian with mean 4 and variance 3, that is, $\mathcal{N}(4, 3)$. b) Generate a waveform with independent but not identically distributed samples: the samples with odd indexes are draw from $\mathcal{N}(4, 3)$, while the samples with even indexes are draw from a uniform distribution $\mathcal{U}(5, 7)$ with support $[5, 7]$.

1.11. Practice the generation of two-dimensional random vectors drawn from a PDF $f_{\mathbf{X}_1,\mathbf{X}_2}(x_1, x_2)$ with mean $\boldsymbol{\mu} = (2,\ 3)^T$. Use the Matlab/Octave's function mvnrnd or, if you do not have Matlab's Statistics Toolbox installed, use the companion octave_mvnrnd instead. An example follows:

Listing 1.28: *MatlabOctaveCodeSnippets/snip_signals_2Drandom.m*

```
N = 100000; %number of 2-d vectors
mu=[2 3]; %mean
C=[1 0.5 ; 0.5 10]; %covariance matrix
r = octave_mvnrnd(mu,C,N); %octave_mvnrnd or mvnrnd
numbiny = 30; numbinx = 30; %number of bins for histogram
```

```
Cest=cov(r) %check estimated covariance matrix (should be close to C)
mu_est=mean(r) %estimated mean
R=C+mu'*mu %theoretical correlation matrix
Rest=Cest + mu_est'*mu_est %estimated correlation matrix
[n,xaxis,yaxis]=ak_hist2d(r(:,1),r(:,2),numbinx,numbiny); %histogram
mesh(xaxis,yaxis,n); pause %plot histogram
contour(xaxis,yaxis,n); xlabel('x1'); ylabel('x2'); %and its countour
```

The task is to compare the graphs for the following covariance matrices: a) $\mathbf{C} = \sigma^2\mathbf{I} = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$, $\sigma^2 = 0.5$. b) $\mathbf{C} = \begin{bmatrix} 0.5 & 0 \\ 0 & 4 \end{bmatrix}$. c) $\mathbf{C} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$.

1.18 Exercises

Unless specified otherwise, assume that the signals are in Volts and were obtained over a resistor of 1 Ohm. Always indicate the units in your answers and graphs.

1.1. Depict in plots the even and odd parts of $x[n] = n^2(u[n-3] - u[n-7])$.

1.2. You do not know much about a signal $x[n]$, but it is given that $x[0] = 3$. Can $x[n]$ be an odd signal? Why?

1.3. Let $[3, 4, 5]$ be the amplitudes of the three samples ($n = 0$, 1 and 2, respectively) of a discrete-time $x[n]$ obtained by sampling a speech signal. a) What is the value of $x[n]$ at $n = 1.5$? b) Describe the result of a D/C conversion of $x[n]$ to a sampled signal $x_s(t)$ with sampling period $T_s = 4$ s and draw the graph of $x_s(t)$ specifying the abscissa in seconds. c) What are the amplitude values of $x_s(t)$ at $t = 1.5$ and 8 s?

1.4. a) Manually (do not use a computer) draw the graph of the sampled signal $x_s(t) = \sum_{k=0}^{\infty}(-1)^k\delta(t-k)$ indicating values at both abscissa and ordinate.

1.5. Given $x(t)$ as depicted in Figure 1.40, clearly draw the graphs of a) $x(-t-3)$, b) $x(4t)$ and c) $x(t/2)$.

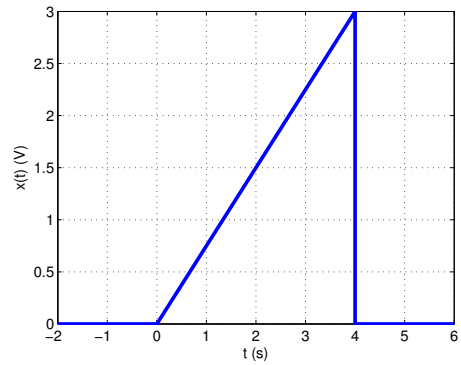


Figure 1.40: *Example of continuous-time signal.*

1.6. a) Manually draw the graph of $x[n] = \delta[n+2] + \delta[n+1] + \delta[n] + \delta[n-1] + \delta[n-2] + 0.4\delta[n+3]$. b) Do the same for the signals: $y[n] = x[n+1]$, $z[n] = x[n+1]\delta[n-2]$ and $w[n] = 3x[n^2]$. c) Convert $x[n]$ to a sampled signal $x_s(t)$ adopting $F_s = 2$ kHz and draw the graph of $x_s(t)$.

1.7. Practice choosing commercial ADCs and DACs. Some companies of interest are Analog Devices, Maxim and Texas Instruments. Assume you have to choose chips for three projects with distinct requirements: 1) the fastest ADC and DAC with at least 16 bits per sample, 2) Low cost chips with 8 bits per sample to work with F_s up to 10 kHz and 3) high precision chips to work with F_s around 100 Hz in medical applications. In your comparison, indicate at least resolution, speed, price, power consumption, supply current and if the data bus is serial or parallel and inform the interface (e. g., SPI). Extra parameters you may include are full scale range (FSR), total harmonic distortion (THD), effective number of bits (ENOB) and offset error.

1.8. Get familiar with digitizing systems and boards, which are sometimes called DAQ (data acquisition) boards. Calculate the storage space and transfer rate for digitized signals (visit [url1bww] for extra information). a) Calculate the total space in megabytes (MB) for storing 30 minutes of a signal sampled at the maximum rate of the following data transfer technologies:

- PCI: 2133 Mbit/s (266.7 MB/s)
- Serial ATA (SATA-300): 3000 Mbit/s (375 MB/s)
- USB 2.0: 480 Mbit/s (60 MB/s)
- Serial RS-232 (max): 0.2304 Mbit/s (0.0288 MB/s)

b) Assume you need to use a 16-bits A/D process to achieve the desired SNR, what is the maximum sampling rate that needs to be supported for each interface above? c) Describe in high-level a digitizer system to sustain a sampling rate of 40 MHz and store 3 hours of a signal. Choose the data transfer technology, total hard disk space, etc.). d) Considering you must use USB 2.0: what is the maximum sampling rate the system could achieve in this case? e) Evaluate a Signatec [url1sig] waveform recording product and indicate what is the maximum throughput that Signatec offers (indicate sampling rate and number of bits per sample) for recording some hours of signal into a hard disk. Note that when operating at maximum sampling rate, most acquisition boards and digital oscilloscopes store the ADC samples in a limited amount of onboard RAM, which is typically capable of storing only few seconds of signal. The discussed recording system must take into account the data transfer from onboard RAM to hard disk.

1.9. For the following signals, calculate the energy E_∞ and power P_∞. Do they have a finite total energy or a finite average power? a) $x(t) = 5\cos(2\pi 300t + \pi/4)$, b) $x(t) = 3e^{-2t}u(t)$ and c) $x(t) = t$.

1.10. What can be said about the total energy and average power of any periodic signal?

1.11. For each of the following signals, determine if it is periodic and its fundamental

period N_0 if positive: a) $x[n] = u[n] + u[-n-1]$, b) $x[n] = \cos(\frac{\pi}{8}n)$, c) $x[n] = \cos(\frac{11}{128}n + \pi/3)$ and d) $x[n] = \cos((\pi/4)n^2)$?

1.12. Classify the following signals as power or energy signals and calculate their autocorrelation $R(\tau)$, using the proper definition. Assume that t is in seconds and indicate the unit of τ: a) $x(t) = \cos(2\pi 100t)$ and b) $x(t) = \exp(-t)u(t)$.

1.13. The following commands were used to estimate the autocorrelation of a cosine: `N=16; n=0:5*N-1; x=cos(2*pi/N*n); R=xcorr(x,'biased');` However, the result did not match the theoretical expression. Can you explain the reason? How could you obtain the proper result? Compare this result with Application 1.13. Why in this case autocorrelation seems to be periodic and match the theoretical result?

1.14. A sinc function $\text{sinc}(\tau)$ (see Section A.12) centered in $\tau = 0$ can be an autocorrelation? What if it was centered in $\tau = 3$ seconds? What is the interpretation to the fact that the autocorrelation achieves its maximum at $\tau = 0$? What is the interpretation for $\text{sinc}(0)$ (the autocorrelation at origin) if the adopted definition were: a) for energy signals and b) for random or power signals?

1.15. A signal $x[n]$ was obtained using the `randn` function in Matlab/Octave, such that it has zero mean and unity variance. What is the power of this signal? How $x[n]$ can be transformed in a signal $y[n]$ with mean equal to four and variance equal to nine? What is the Matlab/Octave command to generate 100 samples of $y[n]$ using `randn`? What is the average power of $y[n]$? Plot the following graphs: autocorrelation and probability density function of both $x[n]$ and $y[n]$.

1.16. The goal is to design a uniform scalar quantizer for a discrete-time signal $x[n]$. It is given that $x[n]$ is approximately uniformly distributed with mean equal to 2 and standard deviation equal to 3. a) Describe a 3-bits quantizer that minimizes the quantization error, explaining your design decisions. b) Calculate the signal to noise ratio (in dB) for the case of a 8 bits quantizer operating with the same input signal.

1.17. A 3-bits uniform quantizer has a step size $\Delta = 2$ V. The input is x and the output x_q. Its minimum output value is $x_q = -8$. Assume four input values $[20, 0.7, -5.4, 6.1]$ V and inform: a) the graph ("stairs") showing $x_q \times x$ for this quantizer, b) the quantization error e_q for each of these input values and c) the power in Watts of e_q considering only the four corresponding samples.

1.18. Assuming the input signal of a quantizer can be modeled by a Gaussian FDP with variance 4 W and mean 2 V. Design a 2-bits uniform quantizer that tries to minimize the quantization error. a) Draw the output versus input for this quantizer and b) calculate the quantization SNR_{dB} assuming only errors in the granular region and that the linear model for quantization is valid.

1.19. In case you have access to the required equipments, estimate and describe in details the quantizer used by the sound system of some personal computer (describe the "stairs": dynamic range and step size). Try to model the DC offset.

1.20. Consider you want to quantize a sinusoid with peak amplitude equals to 10 Volts using $b = 10$ bits. Find general expressions for the quantization SNR in terms of b, using both linear and dB scales. If instead of a sinusoid your signal (to be quantized) had a Gaussian PDF with mean $\mu = 3$ and standard deviation $\sigma = 2$, find similar expressions for the SNR assuming the signal dynamic range is 6σ around the mean.

1.21. The sampling theorem is a strict inequality $F_s > 2f_{\max}$ but some people state it as a non strict inequality, i. e., $F_s \geq 2f_{\max}$, which is incorrect. Provide an example where sampling a sinusoid of 30 Hz with $F_s = 60$ Hz leads to a signal where all samples are zero. For inspiration, take a look at a different approach: assume a cosine

$$x(t) = A\cos\left(2\pi f_0 t + \theta\right), \tag{1.51}$$

where $A = 1/\cos(\theta)$. The sampling frequency is chosen as $F_s = 2f_0$ such that $x(n) = \cos(\pi n) = (-1)^n$ and $x(t)$ cannot be recovered from $x[n]$ (which is the same for any phase θ), as illustrated by Listing 1.29.

Listing 1.29: *MatlabOctaveCodeSnippets/snip_signals_sampling_inequality.m*

```
Fs=20; Ts=1/Fs; %sampling frequency Fs in Hz and period Ts in sec.
f0=Fs/2; %key thing: Fs should  be greater than 2 f0
t=0:Ts:1; %discrete-time axis, from 0 to 1 second
theta1=pi/4 %define an arbitrary angle
A1=1/cos(theta1); %amplitude
x1=A1*cos(2*pi*f0*t+theta1); %not obeying sampling theorem
theta2=0 %define any value distinct from theta1
A2=1/cos(theta2); %amplitude
x2=A2*cos(2*pi*f0*t+theta2); %not obeying sampling theorem
plot(t,x1,'rx',t,x2,'o-b'); title('Cannot distinguish 2 signals!')
```

This ambiguity demonstrates the need for a strict inequality.

1.22. Learn how to manipulate wav files obtained from an audio CD and evaluate their histograms. Were the signals properly digitized? All (or most) quantizer levels were used? In case you find a CDA file, note that these files (of just 44 bytes) are not the actual audio files. They are just pointers to the audio data (similar to shortcut files). In order to copy the files, you need to use a rip software such as, e. g. [url1rip].

2 Transforms and Signal Representation

2.1 To Learn in This Chapter

- Apply basic concepts of linear algebra such as inner products and projections to better understand transforms
- Use inner products to efficiently obtain the transform coefficients when the basis functions are orthogonal
- Interpret transforms (Fourier, Z, Laplace) as obtaining coefficients given by the inner product between the signal to be transformed and the corresponding basis function
- Obtain by inspection the Fourier series coefficients of periodic signals composed by harmonic sinusoids

Transforms are a very important tool in several applications. The continuous-time Fourier transform, for example, provides an alternative "view" $X(f)$ of a signal $x(t)$. Sometimes this extra view is essential for efficiently solving a problem. Few examples of transforms and applications can be found in Table 2.1.

Table 2.1: *Examples of transforms and applications.*

Transform	**Example of application**
Fourier	visualize a signal in frequency "domain" as a sum or integral of sinusoidal components
Z	analyze discrete-time systems by transforming difference equations into polynomials
Discrete cosine transform (DCT)	image coding, where the image details are represented by high-frequency DCT coefficients, which can be discarded without significant loss of perceptual quality

2.2 Matrix multiplication is a linear transform

In linear algebra, any linear transformation[1] (or transform) can be represented by a matrix $\mathbf{A}$. The linear transform operation is given by

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{2.1}$$

[1] See [url2tra].

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output column vectors, respectively. For example, the matrix

$$\mathbf{A} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.2}$$

corresponds to a clockwise rotation of the input vector by an angle θ. Figure 2.1 illustrates the rotation for a vector $\mathbf{x} = [4, 8]^T$ by an angle $\theta = \pi/2$ radians, i. e., $\mathbf{A} = [0, 1; -1, 0]$, resulting in $\mathbf{y} = \mathbf{A}\mathbf{x} = [8, -4]^T$.

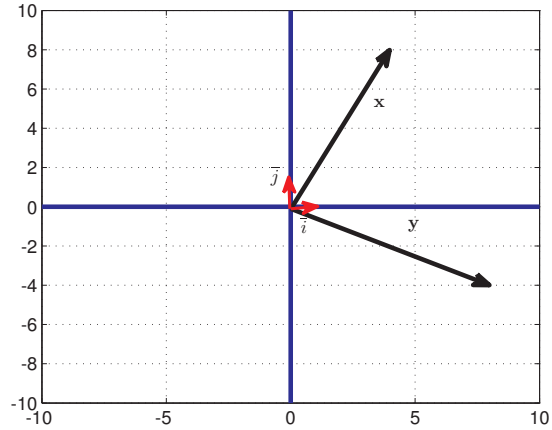


Figure 2.1: *Rotation of a vector* $\boldsymbol{x}$ *by an angle* $\theta = \pi/2$ *radians using* $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ *with* $\boldsymbol{A}$ *given by Eq. (2.2).*

Associating a linear transform to a matrix multiplication is a very important concept. Another one that has origin in linear algebra is the concept of *basis*. Figure 2.1 indicates a pair of orthonormal[2] vectors $\bar{i} = [1, 0]$ and $\bar{j} = [0, 1]$ that span $\mathbb{R}^2$. The vectors $\bar{i}$ and $\bar{j}$ form a *standard basis* and allow to easily represent any vector $\mathbf{y} \in \mathbb{R}^2$, such as $\mathbf{y} = 8\bar{i} - 4\bar{j}$. When the basis functions are organized as the columns of a matrix $\mathbf{A}$, the elements of the input vector $\mathbf{x}$ indicate the coefficients of a linear combination of the basis functions that lead to $\mathbf{y}$. For example, in the case of the standard basis:

$$\mathbf{y} = \begin{bmatrix} 8 \\ -4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 8 \\ -4 \end{bmatrix} = \mathbf{A}\mathbf{x},$$

which is a trivial relation because $\mathbf{A}$ is the identity matrix. More interesting transforms can be used. The previous example of Eq. (2.2) is useful in computer graphics and with $\theta = \pi/2$ radians leads to (see Figure 2.1):

$$\mathbf{y} = \begin{bmatrix} 8 \\ -4 \end{bmatrix} = \begin{bmatrix} \cos(\pi/2) & \sin(\pi/2) \\ -\sin(\pi/2) & \cos(\pi/2) \end{bmatrix} \begin{bmatrix} 4 \\ 8 \end{bmatrix} = \mathbf{A}\mathbf{x}.$$

[2] Vectors that are orthogonal and have norm equal to one.

2.3 Inner Products to Obtain the Transform Coefficients

Linear algebra concepts play an important role in transforms for signal processing (and, of course, many more applications). One crucial concept is that most transforms are designed such that "transforming" the original signal corresponds to the calculation of *inner products*. An inner product is a generalization of the *dot product* $\mathbf{x} \cdot \mathbf{y}$ between two vectors $\mathbf{x}$ and $\mathbf{y}$ of N elements or, equivalently, equal-length sequences (with N samples). The dot product is defined for vectors with real-valued elements as (the notation of an inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ will be used hereafter instead of $\mathbf{x} \cdot \mathbf{y}$):

$$\langle \mathbf{x}, \mathbf{y} \rangle \triangleq \|\mathbf{x}\| \, \|\mathbf{y}\| \cos(\theta), \tag{2.3}$$

where θ is the angle between $\mathbf{y}$ and $\mathbf{x}$. Alternatively, this inner product can also be calculated as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{N} x_i \; y_i = x_1 y_1 + x_2 y_2 + \cdots + x_N y_N, \tag{2.4}$$

where x_i and y_i are the i-th elements of $\mathbf{x}$ and $\mathbf{y}$, respectively. For example, if $\mathbf{x} = [3, -1, 0]$ and $\mathbf{y} = [1, 3, 2]$, their inner product is $\langle \mathbf{x}, \mathbf{y} \rangle = 0$. In this case, from Eq. (2.3) and observing that the norms of both vectors are non-zero, the inner product is zero necessarily because $\cos(\theta) = 0$ and consequently, $\theta = \pi/2$ or $-\pi/2$. In general, when two vectors are *orthogonal* their inner product is zero.

There are many distinct definitions of inner products. But an operation must obey specific properties such as *linearity* to be "valid" as an inner product and, consequently, define an *inner product space*, where concepts such as *norm* and *orthogonality* are natural extensions of the ones with geometric interpretations provided by Eq. (2.3). This geometric interpretation is highly beneficial when interpreting the inner products used in transforms. Table 2.2 illustrates alternative definitions of inner products that are discussed in the sequel.

Table 2.2: *Examples of inner product definitions.*

Equation	**Used for**	**Number**
$\sum_{i=1}^{N} x_i \; y_i$	finite-length real-valued vectors or sequences	(2.4)
$\sum_{i=-\infty}^{\infty} x[n] \; y^*[n]$	infinite-length complex-valued vectors or sequences	(2.5)
$\int_{-\infty}^{\infty} x(t) y^*(t) dt$	continuous-time complex-valued signals	(2.6)
$\int_{\langle T \rangle} x(t) y^*(t) dt$	continuous-time complex-valued signals with duration T	(2.7)

As indicated in Table 2.2, it is also possible to define an inner product for infinite duration signals. For example, consider the complex-valued signal $y[n] = e^{j\Omega n}$, where $n = -\infty, \ldots, -1, 0, 1, \ldots, \infty$. It is a common operation in Fourier transform to calculate

inner products among such signals using the definition

$$\langle x[n], y[n] \rangle \triangleq \sum_{n=-\infty}^{\infty} x[n] y^*[n]. \tag{2.5}$$

Note that when applied to complex-valued signals, the inner product is defined using a complex conjugation.

When dealing with continuous-time signals, a convenient definition is obtained by changing the summation by an integral:

$$\langle x(t), y(t) \rangle = \int_{-\infty}^{\infty} x(t) y^*(t) dt. \tag{2.6}$$

Note that the inner product of finite-duration signals with support T is simplified to

$$\langle x(t), y(t) \rangle = \int_{\langle T \rangle} x(t) y^*(t) dt, \tag{2.7}$$

where $\langle T \rangle$ represents a range of T such as $[0, T]$ or $[-T/2, T/2]$.

When using inner products between continuous-time signals, it is possible to make useful analogies to vectors in the Euclidean space and benefit from geometrical interpretations. For example, similar to vectors, the squared norm $\langle x(t), x(t) \rangle = \|x(t)\|^2$ is the signal energy.[3] Another analogy: the signals $x(t) = 1$ and $y(t) = 2u(t) - 1$ can be interpreted as orthogonal because $\langle x(t), y(t) \rangle = 0$. This inner product can be obtained by noting that $x(t)y(t)$ is -1 for $t < 0$ and 1 for $t \geq 0$, leading to a zero integral. Noting that orthogonal signals are similar to orthogonal vectors will be useful.[4]

At this point, the reader may eventually benefit from Appendix A.14, which provides a review of linear algebra applied to transforms. As mentioned, the goal is to interpret, for example, the Fourier transform

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt$$

as the inner product $\langle x(t), e^{j\omega t} \rangle$ between the signal $x(t)$ and the basis function $e^{j\omega t}$.

As discussed in Section A.14.3, if a signal $x(t)$ can be represented as a linear combination $x(t) = \sum_{d=1}^{D} m_d \varphi_d(t)$, where the D functions $\varphi_d(t)$ compose a set $\{\varphi_j(t)\}$ of orthonormal basis functions, then the values (or *coefficients*) m_d can be recovered using inner products:

$$m_d = \langle x(t), \varphi_d(t) \rangle. \tag{2.8}$$

This result is discussed in Section A.14.3 via examples with vectors.

Before dealing with infinite duration signals, the next section discusses transforms that operate on blocks of samples.

[3] Unless otherwise stated, $\|\cdot\|^2$ denotes the Euclidean or L^2 norm.

[4] Note that orthogonal is a generalization of perpendicular.

2.4 Block Transforms

Many algorithms segment the signal in blocks of samples and process each block individually. For example, when writing Matlab/Octave code for digital communications, it is tempting to organize the information in one vector, which is then processed over a channel and converted to another vector, etc. This works well when the number of samples is relatively small. However, many situations require sending billions of samples, bits or vectors, and the corresponding long arrays would eventually not fit in memory anymore. Hence, in many cases the best strategy is to write DSP code using block-processing.

This section discusses block transforms as a specific block-oriented signal processing, including DCT, DFT, and Haar transforms, which together with the KLT (or PCA) already presented, are among the most used transforms.

When processing a discrete-time "infinite" duration signal $x[n]$ in blocks (also called *frames*), $x[n]$ is iteratively segmented and represented by finite dimensional vectors $\mathbf{x}_m$ of dimension N. Assuming that $x[n]$ is a right-sided signal ($n = 0$ corresponds to the first non-zero sample, $n = 1$ to the second and so on), the first block $\mathbf{x}_0$ is formed by the samples

$$\mathbf{x}_0 = [x[0], x[1], \ldots, x[N-2], x[N-1]]^T.$$

In general, the m-th block is represented by the column vector

$$\mathbf{x}_m = [x[mN], x[mN+1], \ldots, x[mN+N-2], x[mN+N-1]]^T. \tag{2.9}$$

To avoid too many indexes, the dependence on m will be omitted hereafter.

Example 2.1. **Segmenting a signal into blocks**. Suppose the task is to segment a very long signal into blocks and then calculate the power of each block. Listing 2.1 illustrates how to perform this task in Matlab/Octave.

Listing 2.1: *MatlabOctaveCodeSnippets/snip_transforms_segmentation.m*

```
S=10000; %some arbitrary number of samples
x=rand(1,S); %create some very very long vector
N=50; %number of samples per frame (or block)
M=floor(S/N); %number of blocks, floor may discard last block samples
powerPerBlock=zeros(M,1); %pre-allocate space
for m=0:M-1 %following the book convention
    beginIndex = m*N+1; %index of the m-th block start
    endIndex = beginIndex+N-1; %m-th block end index
    xm=x(beginIndex:endIndex) %the samples of m-th block
    powerPerBlock(m+1)=mean(abs(xm).^2); %estimate power of block xm
end
clf; subplot(211); plot(x); subplot(212); plot(powerPerBlock) %plot
```

Note that Eq. (2.9) assumes the first index is 0, which is adopted in C, Java and other languages. However, in Matlab/Octave the first index is 1 and this required using beginIndex = m*N+1 instead of beginIndex = m*N as in Eq. (2.9).

The reader is invited to use a speech signal instead of the random signal and observe how the signal power varies over time, comparing it for vowels and consonants. □

As detailed in Appendix A.14, in many signal processing tasks it is useful to have two related transformations called a *transform pair*. They are typically defined by a pair of $N \times N$ square matrices $\mathbf{A}$ and $\mathbf{A}^{-1}$ and the jargon *N-point transform* indicates their dimension. The inverse matrix $\mathbf{A}^{-1}$ is assumed to exist and can "undo" the transformation $\mathbf{A}$ (and vice-versa). Here, the *forward* (or direct) transformation is denoted as

$$\mathbf{X} = \mathbf{A}\mathbf{x},$$

while the *inverse* transformation is denoted here as

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{X}. \tag{2.10}$$

The vector $\mathbf{X}$ is called the transform of $\mathbf{x}$ and the elements of $\mathbf{X}$ are called *coefficients*. The columns of $\mathbf{A}^{-1}$ are called the *basis functions* (or basis vectors). In this text, vectors are represented by bold lower case letters, but the vector with coefficients will be denoted by capital letters to be consistent with the jargon (unfortunately, vectors such as $\mathbf{X}$ can be confused with matrices, but the context will distinguish them).

A matrix $\mathbf{A}$ with orthonormal columns is called *unitary*. Unitary matrices are widely used in transforms because their inverse is simply the conjugate transpose. In other words, unitary matrices have the following property:

$$\mathbf{A}^{-1} = \mathbf{A}^{H} = (\mathbf{A}^{*})^{T},$$

where H denotes the Hermitian (conjugate transposition). In order to see that, first consider the elements of $\mathbf{A}$ are real numbers, such that $\mathbf{A}^{H} = \mathbf{A}^{T}$. The result of the product $\mathbf{A}^{T}\ \mathbf{A} = \mathbf{I}$ is the identity matrix because the inner product between the columns of $\mathbf{A}^{T}$ (the basis functions) with themselves (rows of $\mathbf{A}$) is one when they coincide (main diagonal of $\mathbf{I}$) and zero otherwise. In a more general case of a complex $\mathbf{A}$, one has $\mathbf{A}^{-1} = \mathbf{A}^{H}$ via a similar reasoning.

Example 2.2. **Coding and frequency analysis: two applications of block transforms**. As detailed in Appendix A.14, the block linear transform corresponds to a matrix multiplication. Section 2.10 discusses examples of applications, including coding. One advantage of adopting block transforms in coding applications is the distinct importance of coefficients. In the original domain, all samples (or pixels in image processing) have the same importance, but in the transform domain, coefficients typically have distinct importance. Hence, the coding scheme can concentrate on representing the most important coefficients and even discard the non-important ones.

Transforms are also useful in frequency analysis applications in which the goal is to evaluate what frequencies compose the input signal. A single frequency (or tone) is equivalent to a sinusoidal signal and the typical task is to find the most relevant frequencies. For example, the signal $x[n] = 2\cos(2\pi/3\ \ n) + 3\sin(2\pi/5\ \ n) + 7\cos(2\pi/11\ \ n)$ can be considered to have frequency components with coefficients values

2, 3 and 7, corresponding to the angular frequencies of $\Omega = 2\pi/3$, $2\pi/5$ and $2\pi/11$ radians, respectively. □

The next paragraphs present the DCT transform, which can be used for both frequency analysis and coding.

2.4.1 DCT transform

An example of a unitary matrix transform very useful for coding is the discrete cosine transform (DCT). When $N = 4$, the corresponding matrices are

$$\mathbf{A} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1.307 & 0.541 & -0.541 & -1.307 \\ 1 & -1 & -1 & 1 \\ 0.541 & -1.307 & 1.307 & -0.541 \end{bmatrix}$$

and

$$\mathbf{A}^{-1} = \mathbf{A}^H = \mathbf{A}^T = \frac{1}{2}\begin{bmatrix} 1 & 1.307 & 1 & 0.541 \\ 1 & 0.541 & -1 & -1.307 \\ 1 & -0.541 & -1 & 1.307 \\ 1 & -1.307 & 1 & -0.541 \end{bmatrix}$$

for the direct and inverse transforms, respectively.

Considering that the first element is $a_{0,0}$, an element $a_{n,k}$ of the N-point inverse DCT matrix $\mathbf{A}^{-1} = \{a_{n,k}\}$ can be obtained by

$$a_{n,k} = w_k \cos\left(\frac{\pi(2n+1)k}{2N}\right),$$

where w_k is a scaling factor that enforces the basis vectors to have unit norm, i. e., $w_k = \frac{1}{\sqrt{N}}$ for $k = 0$ and $w_k = \sqrt{\frac{2}{N}}$ for $k = 1, 2, \ldots, N-1$.

Example 2.3. **DCT calculation in Matlab/Octave**. Listing 2.2 illustrates how the DCT matrices can be obtained in Matlab/Octave.

Listing 2.2: *MatlabOctaveFunctions/ak_dctmtx.m*

```
function [A, Ai] = ak_dctmtx(N)
% function [A, Ai] = ak_dctmtx(N)
%Calculate the DCT-II matrix of dimension N x N.
%A and Ai are the direct and inverse transform matrices, respectively
Ai=zeros(N,N); %pre-allocate space
scalingFactor = sqrt(2/N); %make base functions to have norm = 1
for n=0:N-1 %a loop helps to clarify obtaining Ai (inverse) matrix
    for k=0:N-1 %first array element is 1, so use A(n+1,k+1):
       Ai(n+1,k+1)=scalingFactor*cos((pi*(2*n+1)*k)/(2*N));
```

```
    end
end
Ai(1:N,1)=Ai(1:N,1)/sqrt(2); %different scaling factor, k=0
%unitary transform, so the direct is the Hermitian:
A = Ai'; %command ' is Hermitian. Matrix is real so, just transpose
```

The matrices obtained with ak_dctmtx.m can be used to perform the transformations but this has only pedagogical value. There are algorithms for computing the DCT that are faster than a plain matrix multiplication. Check the functions dct and idct in Matlab/Octave. □

Example 2.4. **The DCT basis functions are cosines of distinct frequencies**. Figure 2.2 shows four basis functions of a 32-points DCT transform.

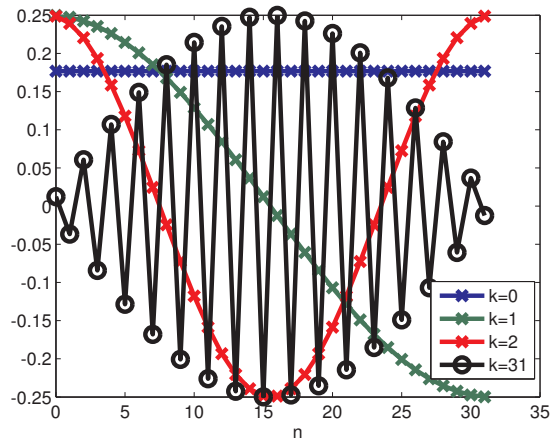


Figure 2.2: *The first three ($k = 0, 1, 2$) and the last ($k = 31$) basis functions for a 32-points DCT. Note that the frequency increases with k.*

Figure 2.2 indicates that, in order to represent signals composed by "low frequencies", DCT coefficients of low order (small values of k can be used), while higher order coefficients are more useful for signals composed by "high frequencies". For example, the commands:

```
N=32;k=3;n=0:N-1;x=7*cos(k*(pi*(2*n+1)/(2*N)));stem(x); X=dct(x)
```

return a vector X with all elements equal to zero but X(4)=28, which corresponds to $k = 3$ (recall the first index in Matlab/Octave is 1, not 0). Using a larger k will increase the frequency and the order of the corresponding DCT coefficient. □

Example 2.5. **Example of a DCT transformation**. For example, assuming a 4-points DCT and $\mathbf{x} = [1, 2, 3, 4]^T$, the *forward* transform can be obtained in this case

with

$$\mathbf{X} = \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \mathbf{A}\mathbf{x}$$

$$\approx \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1.307 & 0.541 & -0.541 & 1.307 \\ 1 & -1 & -1 & 1 \\ 0.541 & -1.307 & 1.307 & -0.541 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\approx \begin{bmatrix} 5 \\ -2.230 \\ 0 \\ -0.158 \end{bmatrix}.$$

In this case,

$$X(0) \;=\; \langle \mathbf{x}, \mathbf{A}(:,0)) \rangle \;=\; \langle [1,2,3,4]^T, 0.5[1,1,1,1]^T \rangle \;= 5,$$

where $\mathbf{A}(:,0)$ represents the first (0-th) column of matrix $\mathbf{A}$. Similarly, $X(2)$ is given by

$$X(2) \;=\; \langle \mathbf{x}, \mathbf{A}(:,2)) \rangle \;=\; \langle [1,2,3,4]^T, 0.5[1,-1,-1,1]^T \rangle \;= 0,$$

and so on.

The previous expressions provide intuition on the direct transform. In the inverse transform, when reconstructing $\mathbf{x}$, the coefficient $X(k)$ is the scaling factor that multiplies the k-th basis function in the linear combination $\mathbf{x} = \mathbf{A}^{-1}\mathbf{X}$. Still considering the 4-points DCT, the inverse corresponds to

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1.307 & 1 & 0.541 \\ 1 & 0.541 & -1 & -1.307 \\ 1 & -0.541 & -1 & 1.307 \\ 1 & -1.307 & 1 & -0.541 \end{bmatrix} \begin{bmatrix} 5 \\ -2.230 \\ 0 \\ -0.158 \end{bmatrix}$$

$$= \frac{1}{2} \left\{ 5 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - 2.230 \begin{bmatrix} 1.307 \\ 0.541 \\ -0.541 \\ -1.307 \end{bmatrix} \right.$$

$$+0 \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} - 0.158 \begin{bmatrix} 0.541 \\ -1.307 \\ 1.307 \\ -0.541 \end{bmatrix} \Bigg\}.$$

Note that $X(2) = 0$ and, consequently, the basis function $[1, -1, -1, 1]^T$ is not used to reconstruct $\mathbf{x}$. The reason is that this specific basis function is orthogonal to $\mathbf{x}$ and does not contribute to its construction. □

Alternatively, the matrix multiplication can be described by a transform equation. For example, the DCT coefficients can be calculated by

$$X[k] = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} x[n] \cos\left(\frac{\pi(2n+1)k}{2N}\right), k = 1, \ldots, N-1$$

and

$$X[0] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n].$$

As mentioned, the k-th element (or coefficient) $X(k)$ of $\mathbf{X}$ can be calculated as the inner product of $\mathbf{x}$ with the complex conjugate of the k-th basis function. This can be done because the DCT basis functions are orthogonal among themselves, as discussed in Section A.14.3. The factors $\sqrt{2/N}$ and $1/\sqrt{N}$ are used to have basis vectors with unitary norms. The k-th basis vector is a cosine with frequency $(k\pi)/N$ and phase $(k\pi)/(2N)$.

2.4.2 DFT transform

As the DCT, the discrete Fourier transform (DFT) is a very useful tool to accomplish frequency analysis, where the goal is to estimate the coefficients for basis functions that are distinguished by the their frequencies. The DFT is related to the discrete-time Fourier series, which also uses cosines $\cos(\frac{2\pi nk}{N})$ and sines $\sin(\frac{2\pi nk}{N})$, $k = 0, 1, \ldots, N-1$, as basis functions, and will be discussed in this chapter. While the DCT uses cosines and its matrices are real, the DFT uses complex exponentials as basis functions.

Using Euler's formula, Eq. (A.1), complex numbers provide a more concise representation of sines and cosines and the k-th *DFT basis function* is given by

$$\frac{1}{N} e^{\frac{j2\pi nk}{N}} = \frac{1}{N} \left(\cos(\frac{2\pi nk}{N}) + j \sin(\frac{2\pi nk}{N})\right), \tag{2.11}$$

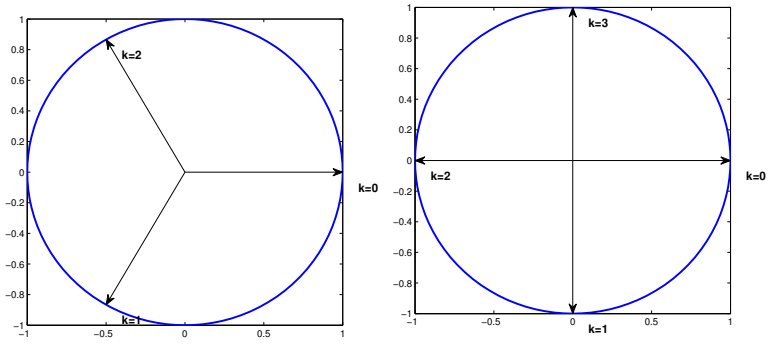
where $n = 0, 1, \ldots, N-1$ expresses time evolution, as for the DCT. Varying k, one obtains different frequencies. The DFT scaling factor $1/N$ must be changed to $1/\sqrt{N}$ if one wants to have basis functions with unitary norm. In this case the transform is called *unitary DFT*.

For convenience, the *twiddle factor* W_N is defined as

$$W_N = e^{-\frac{j2\pi}{N}} \tag{2.12}$$

such that the k-th basis is $(1/N)\,(W_N)^{-nk}$ for the DFT and $(1/\sqrt{N})\,(W_N)^{-nk}$ for the unitary DFT. Twiddle means to lightly turn over or around and is used because the complex number W_N has unitary magnitude and changes only the angle of a complex number that is multiplied by it.

Figure 2.3 illustrates the complex numbers W_N as vectors for different values of N. Because $|W_N| = 1$, the twiddle factor is located on the unit circle of the complex plane and effectively informs an angle. For example, the three angles used by a DFT of $N = 3$ points are 0, 120 and 240 degrees, while a 4-points DFT uses 0, 90, 180 and 270.



(a) $N = 3$ (b) $N = 4$

(c) $N = 5$ (d) $N = 6$

Figure 2.3: *The angles corresponding to W_N in the unity circle, for $N = 3, 4, 5, 6$.*

For $N = 4$, the DFT pair is given by

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \tag{2.13}$$

and

$$\mathbf{A}^{-1} = \frac{1}{N} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^{-1} & W_4^{-2} & W_4^{-3} \\ 1 & W_4^{-2} & W_4^{-4} & W_4^{-6} \\ 1 & W_4^{-3} & W_4^{-6} & W_4^{-9} \end{bmatrix}.$$

Note that $\mathbf{A}^{-1} \neq \mathbf{A}^H$ because the basis functions have norm equal to $1/\sqrt{N}$ (the DFT matrix $\mathbf{A}$ is not unitary). In this case,

$$\mathbf{A}^{-1} = N\mathbf{A}^H. \tag{2.14}$$

Also note that the reason to have W_N defined as a negative exponent in Eq. (2.12) is that the direct transform $\mathbf{A}$ uses positive powers of W_N. Another important fact is that $(W_N)^{aN+b} = (W_N)^b$ for any $a \in \mathbb{Z}$. This can be seen by noting that W_N^n has a period of N samples and $(W_N)^{aN} = 1$. Hence, the 4-points direct DFT matrix of Eq. (2.13) can be written as

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & 1 & W_4^2 \\ 1 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}.$$

It can be seen that the matrix $\mathbf{A}$ (and also $\mathbf{A}^{-1}$) has several symmetries that can be explored to save computations. There is a large set of algorithms for efficiently computing the DFT. These are collectively called *fast Fourier transform* (FFT) algorithms. Apart from numerical errors, both DFT (the plain matrix multiplication) and FFT algorithms lead to the same result. Therefore, the DFT is the name of the transform and FFT is a fast algorithm (from a large collection) used to calculate the DFT. Implementing an FFT routine is not trivial but they are typically available in softwares such as Matlab/Octave.

There are FFT algorithms specialized in the case where the FFT size N is a power of two (i. e., $N = 2^a,\ a \in \mathbb{N}$). Other FFT algorithms do not present this restriction but typically are less efficient. The computational costs of DFT and FFT are quite different when N grows large. The DFT matrix multiplication requires computing

N inner products, so the order of this computation is $\mathcal{O}(N^2)$ while FFT algorithms achieve $\mathcal{O}(N \log_2 N)$. Figure 2.4 illustrates this comparison.

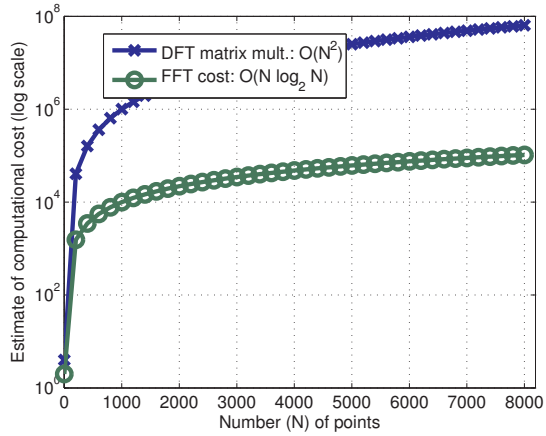


Figure 2.4: *Computational cost of the DFT calculated via matrix multiplication versus an FFT algorithm. Note that $N = 4096$ are currently used in standards such as VDSL2, for example, and it is clearly unreasonable to use matrix multiplication.*

In general, the element in row k and column n of the forward DFT matrix $\mathbf{A}$ is W_N^{nk}. The element in row n and column k of an inverse DFT $\mathbf{A}^{-1}$ corresponds to the complex conjugate W_N^{-nk} normalized by N.

An alternative to matrix notation is to use transform equations. The N-point DFT pair (forward and inverse, respectively) is:

$$X[k] = \sum_{n=0}^{N-1} x[n] \left(W_N\right)^{nk} \tag{2.15}$$

and

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \left(W_N\right)^{-nk}. \tag{2.16}$$

For the unitary DFT pair, the normalization factors should be changed to:

$$X[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] \left(W_N\right)^{nk} \tag{2.17}$$

and

$$x[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X[k] \left(W_N\right)^{-nk}. \tag{2.18}$$

2.4.3 Haar transform

In signal processing, the Haar transform is intimately related to *wavelets*. This section will not explore this relation nor detail the generation law of Haar matrices. The goal

here is to motivate the study of wavelets by indicating how basis functions with support shorter than N can be useful.

The Haar transform is unitary, such that $\mathbf{A}^{-1} = \mathbf{A}^H$. The reader can use the script MatlabOctaveThirdPartyFunctions/haarmtx.m with A=haarmtx(N) to obtain the forward matrix $\mathbf{A}$. For $N = 2$, the Haar, DCT and unitary DFT forward matrices are all the same:

$$\mathbf{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

For $N = 4$, the Haar forward matrix is

$$\mathbf{A} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}.$$

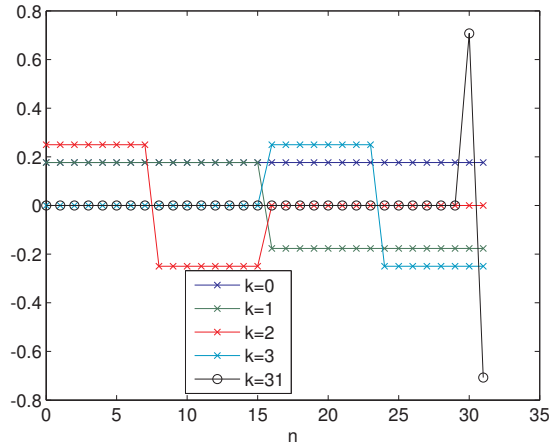


Figure 2.5: *The first four ($k = 0, 1, 2, 3$) and the last ($k = 31$) basis functions for a 32-points Haar. Note that the frequency increases with k.*

Figure 2.5 and Figure 2.6 depict some Haar basis functions. The most interesting aspect is that, in contrast to DCT and DFT, some Haar functions are non-zero only at specific intervals. This characteristic allows the Haar coefficients to provide information not only about "frequency", but also about "time" (or localization). For example, it is well-known in signal processing that Fourier transforms are better suited for stationary signals while wavelets can better represent transient signals in a time-frequency plan. This kind of behavior can already be visualized by comparing DFT (and DCT) basis with Haar's and is explored in Application 2.2.

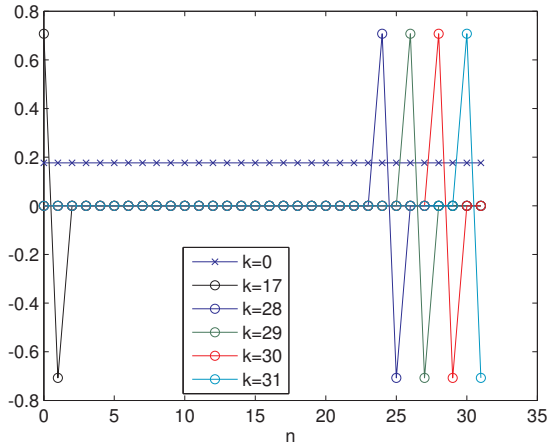


Figure 2.6: *Basis functions $k = 0, 17$ and the last four ($k = 28, 29, 30, 31$) for a 32-points Haar. The support (range of non-zero samples) of the last 15 functions ($k \geq 17$) is just two samples.*

2.4.4 Unitary matrices lead to energy conservation

As commented, for transforms with unitary matrices, $X(k)$ is the inner product between $\mathbf{x}$ and the complex conjugate of the k-th basis function. In the inverse transform, the coefficient $X(k)$ is the scaling factor that multiplies the k-th basis function. This interpretation is not valid for a general non-unitary transform pair $\mathbf{A}$ and $\mathbf{A}^{-1}$.

Another interesting property of a unitary matrix (or transform) $\mathbf{A}$ is that

$$||\mathbf{X}|| = ||\mathbf{A}\mathbf{x}|| = ||\mathbf{x}||.$$

As indicated in Eq. (1.25), the squared norm of a vector corresponds to its energy and, consequently, a unitary matrix *conserves energy* when $\mathbf{x}$ is transformed into $\mathbf{X}$. The following example summarizes this result.

Theorem 2. Parseval theorem for block transforms. If $\mathbf{A}$ is unitary, the transform $\mathbf{X} = \mathbf{A}\mathbf{x}$ conserves energy ($||\mathbf{X}||^2 = ||\mathbf{x}||^2$) or, equivalently, $||\mathbf{X}|| = ||\mathbf{x}||$.

Proof sketch: First, recall that

$$\begin{aligned}\|\mathbf{a}+\mathbf{b}\|^2 &= \langle \mathbf{a}+\mathbf{b}, \mathbf{a}+\mathbf{b}\rangle = \langle \mathbf{a},\mathbf{a}\rangle + \langle \mathbf{a},\mathbf{b}\rangle + \langle \mathbf{b},\mathbf{a}\rangle + \langle \mathbf{b},\mathbf{b}\rangle \\ &= \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2 + 2\langle \mathbf{a},\mathbf{b}\rangle.\end{aligned}$$

To visualize that $\|\mathbf{X}\| = \|\mathbf{x}\|$, for simplicity, assume that $N = 2$ and $\mathbf{x} = [\alpha,\ \beta]^T$. Hence, $\mathbf{X} = \alpha\mathbf{A}(:,0) + \beta\mathbf{A}(:,1)$ and, because $\mathbf{A}$ is unitary, $||\mathbf{A}(:,0)|| = ||\mathbf{A}(:,1)|| = 1$ and $\langle \mathbf{A}(:,0), \mathbf{A}(:,1)\rangle = 0$. Therefore, the squared norm is $\|\mathbf{X}\|^2 = \alpha^2\|\mathbf{A}(:,0)\|^2 + \beta^2\|\mathbf{A}(:,1)\|^2 + 2\langle \mathbf{A}(:,0), \mathbf{A}(:,1)\rangle = \alpha^2 + \beta^2 = \|\mathbf{x}\|^2$. Same reasoning applies for $N > 2$ as indicated by Eq. (A.32). □

Note that in a coding application that uses a unitary transform, the error energy in time and "frequency" domains are the same. In other words, assume that $\mathbf{A}$ is a unitary

matrix and the original vector $\mathbf{x} = \mathbf{A}^H\mathbf{X}$ is obtained from the coefficients $\mathbf{X}$. If the original coefficients $\mathbf{X}$ are represented by $\hat{\mathbf{X}}$, the coefficients error vector $\mathbf{e}_f = \mathbf{X} - \hat{\mathbf{X}}$ has the same norm $||\mathbf{e}_f|| = ||\mathbf{e}_t||$ as the error vector in time domain $\mathbf{e}_t = \mathbf{x} - \hat{\mathbf{x}}$. The proof uses the reasoning adopted in Eq. (A.32):

$$\|\mathbf{e}_f\| = \|\mathbf{X} - \hat{\mathbf{X}}\| = \|\mathbf{A}(\mathbf{x} - \hat{\mathbf{x}})\| = \|\mathbf{A}\mathbf{e}_t\| = \sqrt{(\mathbf{A}\mathbf{e}_t)^H\mathbf{A}\mathbf{e}_t} = \|\mathbf{e}_t\|. \tag{2.19}$$

2.4.5 Orthogonal but not unitary also allows easy inversion

Orthogonal but not unitary matrices also lead to useful transforms. They do not lead to energy conservation but the coefficients are conveniently obtained by inner products, as for unitary matrices. An important detail is that when used in transforms, the inner products with the basis vectors of a orthogonal matrix must be normalized by the norms, as discussed for the DFT. This aspect is further discussed in the sequel. The term "energy" is used here instead of vector "norm" because the results are valid not only for vectors, but also for continuous-time signals, for example.

An orthogonal matrix $\mathbf{B}$ can be written as $\mathbf{B} = \mathbf{A}\mathbf{D}$, where $\mathbf{A}$ is unitary and $\mathbf{D} = \text{diag}[\sqrt{E_1}, \ldots, \sqrt{E_N}]$ is a diagonal matrix with the norm of the i-th column of $\mathbf{B}$, or square root of its energy E_i, composing the main diagonal. The inverse of a matrix with orthogonal columns is

$$\mathbf{B}^{-1} = \text{diag}[1/\sqrt{E_1}, \ldots, 1/\sqrt{E_N}]\mathbf{A}^H = \text{diag}[1/E_1, \ldots, 1/E_N]\mathbf{B}^H.$$

Example 2.6. **Inversion of orthogonal but not unitary matrix**. For example, $[3, 3]^T$ and $[-1, 1]^T$ form a basis for $\mathbb{R}^2$ with energies $E_1 = 18$ and $E_2 = 2$, respectively. The matrix $\mathbf{B} = [3, -1; 3, 1]$ with orthogonal columns can be written as $\mathbf{B} = \mathbf{A}[\sqrt{18}, 0; 0, \sqrt{2}]$, where $\mathbf{A} = [3/\sqrt{E_1}, -1/\sqrt{E_2}; 3/\sqrt{E_1}, 1/\sqrt{E_2}]$ is orthonormal. The inverse of $\mathbf{B}$ is $\mathbf{B}^{-1} = [1/\sqrt{E_1}, 0; 0, 1/\sqrt{E_2}]A^H$ or, equivalently, $\mathbf{B}^{-1} = [1/E_1, 0; 0, 1/E_2]B^H$. □

If all columns of an orthogonal matrix $\mathbf{B}$ have the same energy $E_i = E, \forall i$, its inverse is

$$\mathbf{B}^{-1} = \frac{1}{E}\mathbf{B}^H. \tag{2.20}$$

For example, in Eq. (2.11) or, equivalently, Eq. (2.15), the DFT transform was defined by an orthogonal matrix with the same energy $E = 1/N$ for all basis functions. Therefore, Eq. (2.20) leads to $\mathbf{B}^{-1} = N\mathbf{B}$, which is equivalent to Eq. (2.14).

The next section focuses in Fourier transforms and series. The connections with block transforms are many. For example, extending Eq. (2.20) for continuous-time signals, the basis functions for Fourier series, other than the one for $k = 0$, have energy $E = T_0$ over the duration of a fundamental period T_0. Hence, the inverse transform in Eq. (2.21) has the factor $1/T_0$.

2.5 Fourier Transforms and Series

The transforms in this section adopt eternal sinusoids as basis functions and complement the DFT, which uses finite-length basis functions. For mathematical convenience, complex exponentials are also used given that they conveniently represent sinusoids.

Using sinusoids as basis functions is very useful in many applications. Depending on the type of signal to be analyzed, there are four pairs of analysis and synthesis transform equations that adopt eternal sinusoids as basis functions. These four pairs can be collectively called *Fourier analysis tools* and are discussed in the sequel.

The reason for not having only one transform pair when dealing with Fourier analysis is that two properties of the signal to be analyzed can (or must) be taken in account: the signal is continuous or discrete in time and periodic or not periodic. The four pairs in Table 2.3, which shows the equations to conduct Fourier analysis with eternal sinusoids, cover all possible combinations.

Table 2.3: *The four pair os equations for Fourier analysis with eternal sinusoids and the description of their spectra: c_k, $X(f)$ (or $X(\omega)$), $X[k]$ and $X(e^{j\Omega})$.*

	Continuous-time	**Discrete-time**
Periodic	Fourier **series** $c_k = \frac{1}{T_0}\int_{\langle T_0\rangle} x(t)e^{-j2\pi k f_0 t}dt$ $x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k f_0 t}$	Discrete-time Fourier series (**DTFS**) $X[k] = \frac{1}{N}\sum_{n=\langle N\rangle} x[n]W_N^{nk}$ $x[n] = \sum_{k=\langle N\rangle} X[k]W_N^{-nk}$
Non-periodic	Fourier **transform** $X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt$ $x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft}\mathrm{d}f$ or $X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$ $x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega$	Discrete-time Fourier transform (**DTFT**) $X(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n}$ $x[n] = \frac{1}{2\pi}\int_{\langle 2\pi\rangle} X(e^{j\Omega})e^{j\Omega n}d\Omega$

It can be seen from Table 2.3 that the terminology *series* is used when the signal to be analyzed is periodic. In this case the spectrum is discrete in frequency and represented by *coefficients* c_k or $X[k]$. Obtaining a Fourier series is a special case of the general procedure of representing a function (in this case, a periodic signal) via a series expansion such as Taylor's, Laurent's, etc. In contrast, the spectrum of non-periodic signals is continuous in frequency and the tools to analyze non-periodic signals are called *transforms*. Note this nomenclature is not 100% consistent with block transforms in the sense that the DFT, which has a discrete spectrum, is called "transform".

As highlighted in Table 2.4, in Fourier analysis, there is an interesting and maybe not evident duality between the time and frequency domains: periodicity in one domain leads to a discrete function in the other domain, while non-periodicity leads to a continuous function. For example, Table 2.4 shows that the spectrum of a discrete-time signal $x[n]$ is always periodic.

Table 2.4 and Table 2.3 indicate that the DTFS is the only pair that has discrete

Table 2.4: *Duality of periodicity and discreteness in Fourier analysis.*

	Continuous-time $x(t)$	**Discrete-time** $x[n]$
Periodic	Fourier **series** Basis: $e^{j\omega_0 kt}$ c_k non-periodic, k discrete	Discrete-time Fourier series (**DTFS**) Basis: $e^{j\frac{2\pi}{N}kn}$ $X[k]$ periodic, k discrete
Non-periodic	Fourier **transform** Basis: $e^{j\omega t}$ $X(\omega)$ non-periodic, ω continuous	Discrete-time Fourier transform (**DTFT**) Basis: $e^{j\Omega n}$ $X(e^{j\Omega})$ periodic, Ω continuous

signals in both domains, time and frequency. This indicates that the DTFS is related to the DFT. In fact, the DFT and DTFS are mathematically equivalent with the exception of the normalization factor $1/N$ in Eq. (2.11). But their interpretation has a remarkable distinction: The basis functions of the block transform DFT in Eq. (2.11) are finite-duration sequences or, equivalently, vectors, with dimension N, while the basis functions of the DTFS are infinite-duration complex exponentials $e^{j\frac{2\pi}{N}kn}$ of period N. Hence, the DTFS and DFT are more naturally interpreted and used for periodic and finite-duration signals, respectively.

Another reason for keeping their names different is that the DFT (via FFT algorithms) is often used in computers, digital oscilloscopes, spectrum analyzers, etc., to analyze non-periodic and even continuous-time signals that were digitized. Hence, there are aspects that must be studied to apply DFT in these cases, such as the relation between an original spectrum $X(\omega)$ of a continuous-time signal and $X[k]$, the one obtained by the DFT of its discrete-time version. Therefore, in spite of the operations in DTFS and DFT being mathematically equivalent apart from a normalization constant, it is convenient to restrict the discussion of DTFS to the analysis of periodic and discrete-time signals and call DFT the tool for finite-duration signals, which in practice is used for analyzing any digitized signal.

Besides their relation to the DFT, there are many other similarities and relations within the four Fourier pairs in Table 2.4 themselves. For example, transforms are meant for non-periodic signals but, as discussed in Appendix A.29.2, impulses can be used to also represent periodic signals via a transform (instead of a series). The next sections will discuss some of these relations.

One point that will not be explored in this text is the important aspect of convergence. Similar to the fact that a vector outside the span of a given basis set cannot be perfectly represented by the given basis vectors, there are signals that cannot be represented by Fourier transforms or series. In other words, the transform/series may not converge to a perfect representation even when using an infinite number of basis functions. A related aspect is the well-known *Gibbs phenomenon*: when the signal $x(t)$ has discontinuities (such as $u(t)$ at $t = 0$), the Fourier representation has to use an infinite number of

basis functions. Any truncation of this number (i. e., using a finite number of basis functions) leads to ripples in the reconstructed signal. Given the adopted emphasis in the engineering application of transforms, this text assumes the signals are well-behaved and the transforms and series properly converge.

Complementing Table 2.3, Table 2.5 indicates the assumed units when the signals in time domain are given in Volts and is useful to observe the difference for continuous and discrete spectra.

Table 2.5: *Units for each pair of Fourier equations in Table 2.3.*

	Continuous-time	**Discrete-time**
	x(t) in Volts	x[n] in Volts
Periodic	Fourier series	Discrete-time Fourier series (DTFS)
	c_k (Volts)	$X[k]$ (Volts)
Non-periodic	Fourier transform	Discrete-time Fourier transform (DTFT)
	$X(f)$ (Volts/Hz)	$X(e^{j\Omega})$ (Volts/(normalized frequency))

2.5.1 Fourier series for continuous-time signals

The Fourier series for a continuous-time signal uses an infinite number of complex *harmonic* sinusoids $e^{jk\omega_0 t}, k \in \mathbb{Z}$, as basis functions. These functions allow to represent any periodic signal $x(t)$ with period T_0 (i. e., $x(t) = x(t+T_0), \forall t$), where $\omega_0 = \frac{2\pi}{T_0} = 2\pi f_0$ (rad/s). The frequency f_0 in Hz (or ω_0 in rad/s) is called the *fundamental frequency*.

The following result is useful for proving Fourier pairs.

Example 2.7. **Eternal (infinite-duration) sinusoids at different frequencies are orthogonal**. If $\omega_1 \neq \omega_2$, then

$$\int_{-\infty}^{\infty} A_1 \sin(\omega_1 t + \phi_1) A_2 \sin(\omega_2 t + \phi_2) dt = 0.$$

Proof: To simplify notation, let $\omega_1 t + \phi_1 = \alpha$ and $\omega_2 t + \phi_2 = \beta$. From Eq. (A.10):

$$A_1 \sin(\alpha) A_2 \sin(\beta) = \frac{1}{2}[\cos(\alpha - \beta) - \cos(\alpha + \beta)].$$

The integral from $-\infty$ to ∞ of any sinusoid is zero, therefore:

$$\int_{-\infty}^{\infty} [A_1 \sin(\alpha) A_2 \sin(\beta)] dt = \frac{1}{2}[\int_{-\infty}^{\infty} \cos(\alpha - \beta) dt - \int_{-\infty}^{\infty} \cos(\alpha + \beta) dt] = 0,$$

unless $\alpha = \beta$ (in this case $\cos(\alpha - \beta) = 1$). □

Example 2.8. **Cosine and sine at the same frequency are orthogonal**. Similar to Example 2.7, it can be shown that:

$$\int_{-\infty}^{\infty} A_1 \cos(\omega_0 t) A_2 \sin(\omega_0 t) dt = A_1 A_2/2 \int_{-\infty}^{\infty} \sin(2\omega_0 t) dt = 0,$$

which used Eq. (A.5). □

The following result is useful when the integration interval is the fundamental period T_0 or a multiple of T_0.

Example 2.9. **Harmonic sinusoids are orthogonal when the inner product (integral) is over a time interval multiple of the fundamental period**. If $\omega_0 = 2\pi/T_0$ is the fundamental frequency in rad/s and $k \neq m$, then

$$\int_{<T_0>} A_1 \sin(k\omega_0 t + \phi_1) A_2 \sin(m\omega_0 t + \phi_2) dt = 0.$$

Proof: Note that the sinusoids are assumed here to be eternal, but the result is also valid in case they have a finite-duration coinciding with the time duration of the integral. From Eq. (A.10):

$$\int_{<T_0>} A_1 \sin(k\omega_0 t + \phi_1) A_2 \sin(m\omega_0 t + \phi_2) dt =$$
$$\frac{1}{2}\left[\int_{<T_0>} \cos((k-m)\omega_0 t + \phi_1 - \phi_2) dt - \int_{<T_0>} \cos((k+m)\omega_0 t + \phi_1 + \phi_2) dt\right] = 0.$$

The cosine with angular frequency $(k+m)\omega_0$ of the second parcel has a period $T = T_0/(k+m)$ and its integral over T_0 is zero, because T_0 is an interval corresponding to an integer number of periods T. The same reasoning can be applied to the cosine with angular frequency $(k-m)\omega_0$ with period $T_0/|k-m|$, which is especially easy to observe when $k > m$. If $k < m$, because $\cos(x) = \cos(-x)$, the cosine argument of the first parcel can be changed to $(m-k)\omega_0 t + \phi_2 - \phi_1$ to help concluding that this integral over T_0 is also zero. □

Example 2.9 proves that the Fourier series basis functions are orthogonal. They are not orthonormal and the energy over a duration T_0 is

$$E = \int_{\langle T_0 \rangle} \left| e^{jk\omega_0 t} \right|^2 dt = \int_0^{T_0} dt = T_0,$$

which is the normalization factor that will appear in the Fourier series equations, similar to the result for block transforms in Eq. (2.20).

The basis $b(t) = 1, \forall t$, corresponding to $k = 0$ in a Fourier series, is responsible for representing the DC level of $x(t)$. All other basis functions have a frequency

$$f_k = k f_0.$$

When $k > 1$, frequencies higher than f_0 (considering absolute values) are generated and, consequently, a period $T_k = T_0/k$ that is smaller than T_0. Therefore, all basis functions but the one for $k = 0$ are periodic in T_0. The frequencies f_k that obey a relation $f_k = kf_0$ with respect to a fundamental frequency f_0 are called *harmonics*. The frequency $f_2 = 2f_0$ is called the second harmonic, $3f_0$ is the third harmonic and so on.

It seems intuitive that Fourier series should not be used to represent non-periodic signals. In fact, the basis functions even depend on the period T_0 of the signal to be analyzed. On the other hand, because the signal is periodic, it suffices to find coefficients that represent the signal $x(t)$ during a single period. The trick is then to use these coefficients (obtained with inner products of duration T_0) to multiply eternal complex exponentials and properly represent the periodic (and consequently infinite-duration) $x(t)$. Hence, the Fourier series pair is:

$$\begin{cases} c_k &= \frac{1}{T_0}\int_{\langle T_0\rangle} x(t)e^{-j2\pi kf_0t}dt \quad , k = -\infty, \ldots, -1, 0, 1, \ldots, \infty \\ \\ x(t) &= \sum_{k=-\infty}^{\infty} c_k e^{j2\pi kf_0t} \qquad\qquad , \forall t. \end{cases} \tag{2.21}$$

One can use the reasoning in Theorem 3 (page 170) to prove the Fourier series equations. Writing the synthesis equation

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi kf_0t}$$

is similar to $\mathbf{x} = \sum_{i=1}^{N} \alpha_i \mathbf{b}_i$. Calculating $\langle \mathbf{x}, \mathbf{b}_i \rangle$ is therefore equivalent to

$$\langle x(t), e^{j2\pi mf_0t}\rangle = \int_{\langle T_0\rangle} x(t)e^{-j2\pi mf_0t}dt = \int_{\langle T_0\rangle} \left[\sum_{k=-\infty}^{\infty} c_k e^{j2\pi kf_0t}\right] e^{-j2\pi mf_0t}dt.$$

(recall from Table 2.2 that $\langle x(t), y(t)\rangle = \int x(t)y^*(t)dt$ when the signals are complex). Due to the properties of the inner product one can write

$$\langle x(t), e^{j2\pi mf_0t}\rangle = \sum_{k=-\infty}^{\infty} \left[\int_{\langle T_0\rangle} c_k e^{j2\pi(k-m)f_0t}dt\right] = T_0c_m. \tag{2.22}$$

The last step is due to the orthogonality of the basis functions:

$$\langle e^{j2\pi kf_0t}, e^{j2\pi mf_0t}\rangle = \int_{\langle T_0\rangle} e^{j2\pi kf_0t}e^{-j2\pi mf_0t}dt = \begin{cases} T_0, & k = m \\ 0, & k \neq m. \end{cases}$$

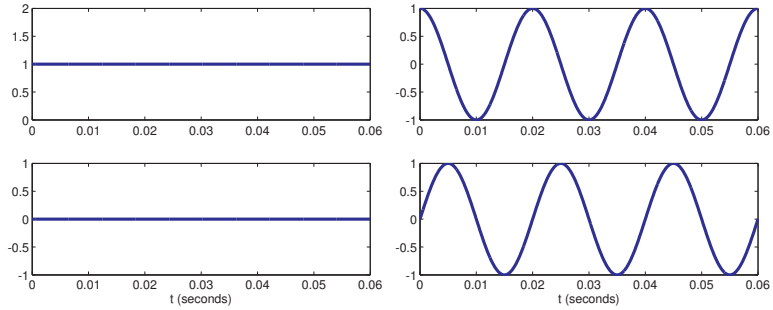
In summary, the previous steps used the basis functions orthogonality to prove that

the analysis equation

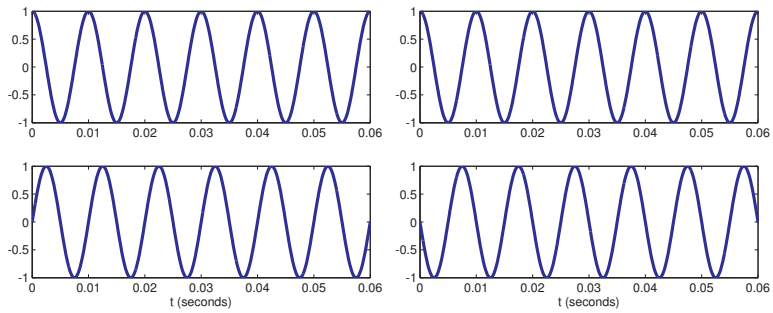
$$c_k = \frac{1}{T_0}\langle x(t), e^{j2\pi k f_0 t}\rangle = \frac{1}{T_0}\int_{\langle T_0\rangle} x(t)e^{-j2\pi k f_0 t}dt$$

corresponds to using the inner product as in Theorem 3 (page 170).

The negative frequencies $f_k, k = -\infty, \ldots, -2, -1$ exist for mathematical convenience. They allow, for example, to represent a cosine as a sum of complex exponentials with "positive" and "negative" frequencies as in Eq. (A.2).



(a) $k = 0$ (DC) (b) $k = 1$ (fundamental)



(c) $k = 2$ (d) $k = -2$

Figure 2.7: *Fourier series basis functions for analyzing signals with period $T_0 = 1/50$ seconds. Because the basis functions are complex-valued signals, the plots show their real (top) and imaginary (bottom) parts.*

Figure 2.7 shows four basis functions for analyzing signals with period $T_0 = 1/50 = 0.02$ seconds. Note that the basis for $k = 0$ is a real signal while the others are complex. The basis for $k = 2$ has the same real part of the one for $k = -2$ and the negative of the imaginary part. This is due to the fact that cosines are even functions $\cos(x) = \cos(-x)$ while sines are odd functions $\sin(x) = -\sin(-x)$ (see Section A.1).

When the Fourier series is used to analyze a real signal $x(t)$, this symmetry of the basis functions leads to interesting properties for the coefficients: the real part of c_k and their magnitude compose even sequences while the imaginary part and their phase compose odd sequences. In summary, for real signals $x(t)$: $|c_k| = |c_{-k}|$, $\mathcal{R}eal\{c_k\} = \mathcal{R}eal\{c_{-k}\}$,

$\angle c_k = -\angle c_{-k}$ and $\mathcal{I}mag\{c_k\} = -\mathcal{I}mag\{c_{-k}\}$. These properties can be written compactly as

$$c_{-k} = c_k^*, \tag{2.23}$$

which corresponds to *Hermitian symmetry*.

Another aspect due to the symmetry is often invoked: because any signal $x(t)$ can be decomposed as the sum $x(t) = x_e(t) + x_o(t)$ of an even part $x_e(t)$ and an odd part $x_o(t)$ (see Section A.1), the cosines are in charge of representing $x_e(t)$, while the sines represent $x_o(t)$. This leads to conclusions such as: if $x(t)$ is real and even, then $\angle c_k = \mathcal{I}mag\{c_k\} = 0, \forall k$.

The properties based on the symmetry of the basis functions and, eventually, also of the signal to be analyzed ($x(t)$ in this case), are valid not only for Eq. (2.21) but for other Fourier pairs.

The graphs in Figure 2.28 are called the *spectrum* of the signal. Because the coefficients c_k are in general complex numbers, a spectrum is represented using the polar or rectangular form of complex numbers. Figure 2.28 uses the polar format. One can notice that, because $x(t)$ is real, $c_{-k} = c_k^*$, i. e., the spectrum presents the Hermitian symmetry of Eq. (2.23).

Trigonometric Fourier series

Instead of complex exponentials $e^{jk\omega_0 t}$ with negative frequencies as in Eq. (2.21), the Fourier series can be written in terms of cosines and sines with positive frequencies only. In this case, one has a set of coefficients for the cosines and another set for the sines that are usually called a_k and b_k, respectively, and related as follows:

$$\begin{cases} a_k = \frac{2}{T_0}\int_{\langle T_0\rangle} x(t)\cos(2\pi k f_0 t)dt \quad , k = 1, 2, \ldots, \infty \\ b_k = \frac{2}{T_0}\int_{\langle T_0\rangle} x(t)\sin(2\pi k f_0 t)dt \quad , k = 1, 2, \ldots, \infty \\ a_0 = \frac{1}{T_0}\int_{\langle T_0\rangle} x(t)dt \\ \\ x(t) = a_0 + \sum_{k=1}^{\infty}\left(a_k\cos(2\pi k f_0 t) + b_k\sin(2\pi k f_0 t)\right). \end{cases} \tag{2.24}$$

Comparing the expression for $x(t)$ in Eq. (2.21) and Eq. (2.24), they are related via Euler's formula (see Eq. (A.1))

$$e^{jk\omega_0 t} = \cos(k\omega_0 t) + j\sin(k\omega_0 t).$$

This allows to write $c_0 = a_0$ and, for $k > 0$:

$$c_k = \frac{1}{2}(a_k - jb_k) \quad \text{and} \quad c_{-k} = \frac{1}{2}(a_k + jb_k).$$

Eq. (2.24) is called the trigonometric Fourier series while Eq. (2.21) is the exponential version.

2.5.2 Discrete-time Fourier series (DTFS)

The Fourier series for discrete-time signals is used to analyze a periodic signal $x[n]$ with fundamental period N. Similar to the continuous-time case, the basis functions consist of a set of complex exponentials formed by a fundamental frequency and its harmonics. The basis corresponding to the fundamental frequency is $e^{j\Omega_0 n}$, where $\Omega_0 = \frac{2\pi}{N}$ radians, and the harmonics are $\Omega_k = k\Omega_0$. A major distinction between the DTFS and the Fourier series is that there are only N distinct frequencies (in fact, angles) for the DTFS because

$$(k+N)\Omega_0 = (k+N)\frac{2\pi}{N} = k\frac{2\pi}{N} + 2\pi = k\Omega_0.$$

Hence, the DTFS pair is:

$$\begin{cases} X[k] &= \frac{1}{N}\sum_{n=\langle N\rangle} x[n]e^{-jk\Omega_0 n} \quad , k = -\infty, \ldots, -1, 0, 1, \ldots, \infty \\ \\ x[n] &= \sum_{k=\langle N\rangle} X[k]e^{jk\Omega_0 n} \qquad , n = -\infty, \ldots, -1, 0, 1, \ldots, \infty. \end{cases} \tag{2.25}$$

If convenient, Eq. (2.25) can be represented using the twiddle factor $W_N = e^{-\frac{j2\pi}{N}}$ as in Table 2.3. The notation $n = \langle N\rangle$ and $k = \langle N\rangle$ represent any interval of N consecutive integers, such as $0, 1, \ldots, N-1$ or $-N, -(N-1), \ldots, -1$. Note the basis functions depend on the period N of the signal to be analyzed.

Because $x[n]$ is periodic, it suffices to specify its samples in an interval $\langle N\rangle$ but Eq. (2.25) indicates that $x[n]$ has an infinite duration (in contrast to the finite-duration sequence obtained by an inverse DFT). As indicated in Eq. (2.25), only $k = \langle N\rangle$ values of $X[k]$ are necessary to represent the periodic $x[n]$, and along k the values $X[k]$ are periodic, i. e., $X[k] = X[k+N]$. The reason is that the angles repeat $(k+N)\Omega_0 = k\Omega_0$ after N samples (as illustrated by the divided unity circle in Figure 2.3) and, consequently, their corresponding basis functions and coefficients $X[k]$.

Similar to Example 2.7, which obtained the Fourier series of a continuous-time signal, when $x[n]$ is composed by a sum of sinusoids, it is relatively easy to find the DTFS coefficients $X[k]$ by inspection, as discussed next.

Calculating the DTFS via the DFT

As mentioned, mathematically the DFT and DTFS differ only by the scaling factor $1/N$, as indicated in Eq. (2.15) and Eq. (2.25), respectively. Using the proper scaling factor will not alter the spectrum shape but it is important in case the numerical values should be interpreted with the correct units. For example, if a cosine is generated and its FFT calculated in Matlab/Octave, the numerical values will be influenced by both the cosine amplitude and value of N. Only after the normalization by $1/N$ one can interpret the spectrum of a periodic signal in Volts.

As mentioned, having a DFT routine, typically implemented as an FFT, one can obtain the forward DTFS by dividing the DFT spectrum by N and, consequently, interpreting as the spectrum of a periodic signal.

Example 2.10. **DTFS using an FFT routine**. Listing 2.3 illustrates how to calculate the DTFS in Matlab/Octave using both their built-in fft function and the companion ak_fftmtx.m, with X and X2 being the same apart from numerical errors around 10^{-13}. Figure 2.8 shows the resulting spectrum.

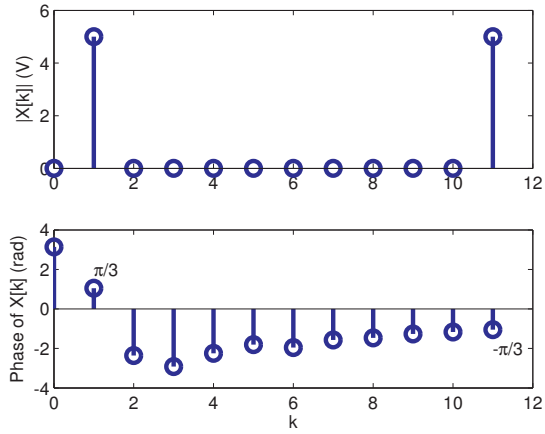


Figure 2.8: *DTFS / DFT of* $x[n] = 10\cos(\frac{\pi}{6}n + \pi/3)$ *calculated with* $N = 12$. *The plot on top is the magnitude obtained with* abs(X) *and the bottom is the phase obtained with* angle(X) *(note some random phases when the magnitude is close to zero).*

Listing 2.3: *MatlabOctaveCodeSnippets/snip_transforms_DTFS.m*

```
N=12; %period in samples
n=transpose(0:N-1); %column vector representing abscissa
Vp=10; x=Vp*cos(pi/6*n + pi/3); %cosine with amplitude 10 V
X=(1/N)*fft(x);%calculate DFT and normalize to obtain DTFS
A=ak_fftmtx(N,2); %DFT matrix with the DTFS normalization
X2=A*x; %calculate the DTFS via the normalized DFT matrix
```

While DFT graphs typically show only the chosen range of N coefficients as in Figure 2.8 and Figure 2.30, sometimes the periodicity of DTFS / DFT must be represented. Therefore, the strict representation of the spectrum of $x[n]$ is given in Figure 2.9, which does not limit the abscissa to N values of k.

Note in the example of Figure 2.9 the value $N = 12$ was chosen to coincide with the period of the cosine and the non-zero coefficients were $k = \pm 1$. Choosing $N = 24$ would move the non-zero coefficients to $k = \pm 2$. If the number of points of the DFT / DTFS was not a multiple of 12, $x[n]$ would still be interpreted as a periodic signal but not a single cosine. In this case the spectrum would have many non-zero coefficients. □

As mentioned, all four pairs of Fourier representations are related. The two pairs of series have been discussed. One can obtain the expressions for transforms from expressions for series by using the limit when the period (N or T_0) goes to infinite. This allows to create an aperiodic signal from a periodic one, and also illustrates that the discrete components of the spectrum (c_k or $X[k]$) get so close to each other that

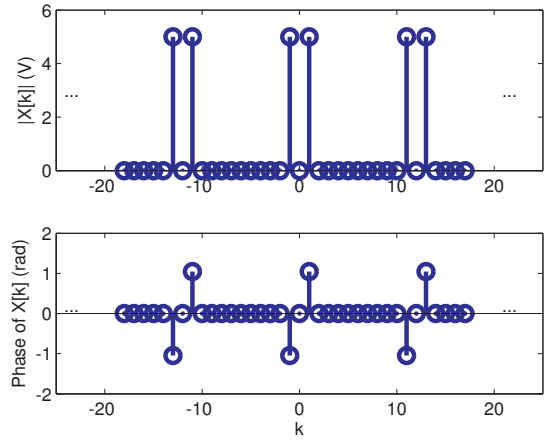


Figure 2.9: *Complete representation of the DTFS / DFT of* $x[n] = 10\cos(\frac{\pi}{6}n + \pi/3)$ *indicating the periodicity* $X[k] = X[k+N]$.

create a continuum of frequencies ($X(f)$ or $X(e^{j\Omega})$). The next sections discuss Fourier transforms.

2.5.3 Continuous-time Fourier transform using frequency in Hertz

The continuous-time Fourier transform[5] uses an infinite number of exponentials $e^{j\omega t}$ as basis functions, where ω is the angular frequency in radians per second and varies from $-\infty$ to ∞. Recall that two exponentials $e^{j\omega_0 t}$ and $e^{j\omega_1 t}$, $\omega_0 \neq \omega_1$, are orthogonal. Because $\omega = 2\pi f$, where f is the frequency in Hertz, the continuous-time Fourier equations are given by

$$\begin{cases} X(f) &= \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt \\ \\ x(t) &= \int_{-\infty}^{\infty} X(f) e^{j2\pi f t} \mathrm{d}f. \end{cases} \tag{2.26}$$

One should interpret the coefficient $X(f_0)$ for a given frequency f_0 as the inner product $X(f_0) = \langle x(t), e^{j2\pi f_0 t} \rangle$. In other words, $X(f_0)$ is the norm of the projection of $x(t)$ on basis $e^{-j2\pi f_0 t}$ (scaled by the norm of the basis).

As anticipated in Table 2.5, if $x(t)$ is given in Volts, the unit of $X(f)$ is Volts $\times$ seconds or, equivalently, Volts/Hz. Similarly, for $x(t)$ in ampere, $X(f)$ is in ampere/Hz.

[5] Check the URLs at [url2jav].

2.5.4 Continuous-time Fourier transform using frequency in rad/s

If one prefers to use rad/s instead of Hz, a change of variable $f = \frac{\omega}{2\pi}$ leads to the alternative definition

$$\begin{cases} X(\omega) &= \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \\ \\ x(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega. \end{cases} \tag{2.27}$$

given that $\mathrm{d}f = \frac{\mathrm{d}\omega}{2\pi}$.

Note that $X(\omega)$ has unit of Volts/Hz and *not* Volts/(rad/s). This is because $X(\omega)$ assumes the values of $X(f)$ but with the abscissa given in rad/s. The factor $\frac{1}{2\pi}$ in Eq. (2.27) is then responsible for having integrals using any of the functions, leading to the same result. For example, for $t = 0$

$$x(t)|_{t=0} = \int_{-\infty}^{\infty} X(f)\mathrm{d}f = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)\mathrm{d}\omega. \tag{2.28}$$

Hence, a plot of $X(f)$ without impulses $\delta(f)$ can be converted to $X(\omega)$ by simply multiplying the abscissa by 2π.

However, when $X(f)$ contains impulses, the scaling property of an impulse has to be taken into account and each area A of an impulse $A\delta(f)$ in $X(f)$ is multiplied by 2π to become $2\pi A\delta(\omega)$ in $X(\omega)$. For example, the transform of $x(t) = \cos(2\pi f_0 t)$ is $X(f) = 0.5[\delta(f + f_0) + \delta(f - f_0)]$ or, equivalently, $X(\omega) = \pi[\delta(\omega + 2\pi f_0) + \delta(\omega - 2\pi f_0)]$.

2.5.5 Discrete-time Fourier transform (DTFT)

The discrete-time Fourier transform uses an infinite number of exponentials $e^{j\Omega n}$ as basis functions, where Ω is the angular frequency in radians The equations are given by

$$\begin{cases} X(e^{j\Omega}) &= \sum_{-\infty}^{\infty} x[n]e^{-j\Omega n} \\ \\ x[n] &= \frac{1}{2\pi}\int_{\langle 2\pi\rangle} X(e^{j\Omega})e^{j\Omega n}d\Omega. \end{cases} \tag{2.29}$$

If $x[n]$ is given in Volts, the unit of $X(e^{j\Omega})$ is Volts per normalized frequency $\Omega/(2\pi)$. The unit of $X(e^{j\Omega})$ is *not* Volts/rad and the division by 2π in the inverse transform of Eq. (2.29) reminds that.

One should interpret the coefficient $X(\Omega_0)$ for a given frequency Ω_0 as the inner product $X(\Omega_0) = \langle x[n], e^{j\Omega_0 n}\rangle$.

The notation is $X(e^{j\Omega})$ because Ω only appears in the imaginary part of an exponent, i. e., Ω is always the angle of a complex exponential, unless Ω is the argument of an impulse $\delta(\Omega)$. For example, $X(e^{j\Omega}) = \frac{1+\Omega}{3+\Omega}$ will never be a "valid" discrete-time Fourier transform. In contrast, a "valid" expression is $X(e^{j\Omega}) = \frac{1+e^{j2\Omega}}{3+e^{-j4\Omega}} + \delta(\Omega)$. One can also

note that

$$X(e^{j\Omega}) = X(e^{j(\Omega+m2\pi)}), m \in \mathbb{Z},$$

which indicates that $X(e^{j\Omega})$ is always periodic in 2π (for any $x[n]$). This discreteness-periodicity duality had been already summarized in Table 2.4.

Calculating the DTFT via the DFT

The DFT corresponds to sampling the DTFT in the frequency domain such that

$$X[k] = X(e^{j\Omega})\Big|_{\Omega=\frac{2\pi}{N}k} \tag{2.30}$$

with N being the DFT dimension. In other words, the DFT uses a frequency increment of $\Delta\Omega = 2\pi/N$ and calculates the value of the DTFT $X(e^{j\Omega})$ at the angles $0, \Delta\Omega, 2\Delta\Omega, \ldots, (N-1)\Delta\Omega$ rad.

The value N is typically chosen to be larger than the duration of $x[n]$. When one wants to sample the DTFT using a high-resolution grid, *zero-padding* can be used, which corresponds to increasing the size of a sequence with zero elements.

2.6 Relating discrete and analog frequencies

If $x[n]$ is obtained by sampling $x(t)$, it is convenient to relate their spectra. This is a followup of Section 1.8. Recall from Eq. (1.10) that $\omega = \Omega F_s$. Hence, F_s can be used to relate the abscissas of graphs of $X(e^{j\Omega})$ and $X(\omega)$. But the periodicity of $X(e^{j\Omega})$ should be taken into account as follows.

After a sampling frequency F_s is specified (e. g., $F_s = 8$ kHz), the value $F_s/2$ represents the frequency f that will be mapped to the angle π rad and its multiples in 2π. This can be seen by

$$\Omega = \omega/F_s = 2\pi f/F_s = 2\pi 4000/8000 = \pi. \tag{2.31}$$

In discrete-time (or "digital") signal processing, the value of $F_s/2$ is called *Nyquist frequency* or *folding frequency*. If the spectrum $X(f)$ is zero[6] for $f \in [F_s/2, \infty[$, then the values of $X(\omega)$ in the range $2\pi \times [-F_s/2, F_s/2]$ coincides with the values of $X(e^{j\Omega})$ normalized by F_s in the corresponding range of $[-\pi, \pi]$. In other words:

$$X(e^{j\Omega}) = F_s X(\omega),$$

which will be proved later on. Note that $X(e^{j\Omega}) = F_s X(\omega)$ only holds if the sampling theorem is obeyed, otherwise the components of signal $x(t)$ with frequencies in the range $]-\infty, F_s/2[$ and $]F_s/2, \infty[$ will be mapped (or folded, which is the reason for calling $F_s/2$ the folding frequency) to components of $x[n]$ with angular frequencies in the range $]-\pi, \pi[$, eventually distorting the discrete-time representation of the

[6] More strictly, only the magnitude matters: the condition is $|X(f)| = 0, f \geq F_s/2$ because the phase can be discarded when the magnitude is zero.

original $x(t)$. This phenomenon is called *aliasing* because a folded component from $X(\omega)$ appears in $X(e^{j\Omega})$ as an "alias" or impostor.

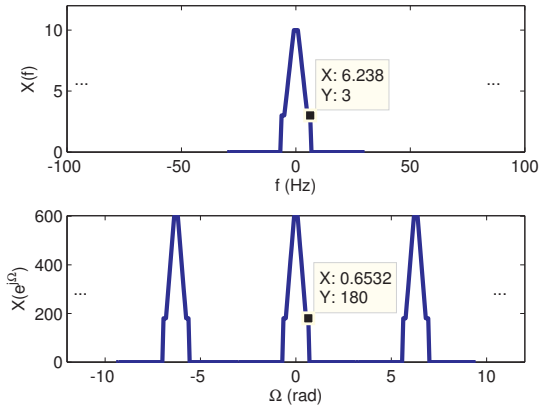


Figure 2.10: *Spectrum $X(f)$ (top) and $X(e^{j\Omega})$ when $F_s = 60$ Hz. The two indicated points are related by $\omega = \Omega F_s$ and the frequency $f = 6.238$ Hz is mapped into $\Omega = 0.6532$ rad. Notice three of the infinite number of replicas of $X(f)$ centered at $\Omega \in [-2\pi, 0, 2\pi]$. In this case there was no aliasing because the sampling theorem was obeyed.*

For the sake of illustration, Figure 2.10 depicts the spectrum $X(f)$ of a continuous-time signal $x(t)$. It also shows the spectrum $X(e^{j\Omega})$ of $x[n]$, obtained by sampling $x(t)$ with $F_s = 60$ Hz. In this case, $\omega = 2\pi \times 6.238$ rad/s is mapped to 0.6532 rad, with the magnitude being scaled by 60, according to $X(e^{j\Omega}) = F_s X(\omega)$. This discussion aims at illustrating how the Fourier transform $X(f)$ of $x(t)$ and the DTFT of the corresponding $x[n]$ (obtained from $x(t)$ via a C/D conversion) are related. The next paragraphs complement Eq. (2.30) and discuss how to interpret the DFT in Hz.

2.7 Summary of equations for DFT / FFT Usage

The DFT resolution in radians is $\Delta\Omega = 2\pi/N$. Using $\omega = \Omega F_s$, one can note that the continuous-time angular frequency $\omega = 2\pi F_s$ rad/s corresponds to the angle $\Omega = 2\pi$ rad. Hence, the DFT frequency spacing is

$$\Delta f = \frac{F_s}{N}. \tag{2.32}$$

For example, assuming $F_s = 100$ Hz, a DFT of $N = 256$ points has a resolution of $\Delta f = 0.3906$ Hz. In radians, this resolution corresponds to $\Delta\Omega = 2\pi/256 \approx 0.0245$ rad.

Note that Δf can also be written as $\Delta f = F_s/N = 1/(NT_s)$, where NT_s is the total duration T_{total} of the signal being analyzed, such that

$$\Delta f = \frac{1}{T_{\text{total}}}. \tag{2.33}$$

Hence, a spectral analysis with resolution of at least Δf Hz requires a data record of

length

$$T_{\text{total}} \geq \frac{1}{\Delta f}. \tag{2.34}$$

The following example illustrates the interpretation in Hz of a DFT result. The DFT performs a sampling operation on the frequency domain Ω according Eq. (2.30). As illustrated in Figure 2.3, a DFT with $N = 3$ points (or 3-DFT) uses three basis functions $e^{j\Omega}$ with the angles $\Omega = 0, 2\pi/3, 4\pi/3$ rad, i. e., a counter-clockwise rotation using a step of $\Delta w = 2\pi/3$. These angles correspond to $k = 0, 1, 2$, respectively, as indicated by Eq. (2.11). Note that in Figure 2.3, incrementing k corresponds to a clockwise rotation, given that Eq. (2.12) defines the twiddle factor with a negative exponent.

The three angles of a 3-DFT correspond to $0, F_s/3, -F_s/3$ Hz, respectively, as indicated by $\omega = \Omega F_s$ or directly observing that the DFT uses a grid of $\Delta f = F_s/N$. For $N = 4$, the four angles correspond to the frequencies $0, F_s/4, F_s/2 - F_s/4$ Hz. The angle corresponding to $k = 0$ is always 0 rad, which is called DC because corresponds to 0 Hz. Note that for $k > N/2$ the DFT values correspond to negative frequencies. The angular frequency corresponding to the k-th angle is

$$\Omega_k = k\Delta\Omega = k\frac{2\pi}{N}, \tag{2.35}$$

which corresponds to the frequency

$$f_k = k\Delta f = k\frac{F_s}{N}. \tag{2.36}$$

The values representing the largest frequency when N is even is

$$k = N/2, \tag{2.37}$$

which corresponds to π rad and $F_s/2$ Hz. When N is odd, the value at

$$k = (N-1)/2 \tag{2.38}$$

is the one representing the largest frequency $F_s(N-1)/(2N)$ Hz. Table 2.6 summarizes some equations typically used with FFT algorithms.

As discussed, the following three transforms differ only on the normalization factor: DFT, unitary DFT and DTFS. Generalizing Eq. (2.15) and Eq. (2.16), the "core" DFT transform equations can be written as follows:

$$X[k] = \alpha \sum_{n=0}^{N-1} x[n] \left(W_N\right)^{nk} \tag{2.39}$$

and

$$x[n] = \beta \sum_{k=0}^{N-1} X[k] \left(W_N\right)^{-nk}. \tag{2.40}$$

Table 2.6: *Summary of equations useful for signal processing with FFT.*

Expression	Reference	Comment
$\Delta f = F_s/N$	Eq. (2.32)	Frequency spacing (Hz)
$\Delta f = 1/T_{\text{total}}$	Eq. (2.33)	Dependence on signal duration (Hz)
$\Delta\Omega = (2\pi)/N$	Eq. (2.50)	Frequency spacing (rad)
$f_k = k\ \Delta f$	Eq. (2.36)	Frequency of k-th tone (Hz)
$\Omega_k = k\ \Delta\Omega$	Eq. (2.35)	Frequency of k-th tone (rad)
$F_s/2$	Eq. (2.37)	Highest frequency (Hz) for even N
$F_s(N-1)/(2N)$	Eq. (2.38)	Highest frequency (Hz) for odd N

As indicated in Eq. (2.20), the only requirement to have a valid pair is that

$$\alpha\beta = \frac{1}{N}. \tag{2.41}$$

In summary, the three popular options for choosing α and β:

- **DFT (equivalent to DTFT at given tones)**: $\alpha = 1$ and $\beta = 1/N$. Matlab/Octave implements this transform in its `fft` and `ifft` routines. It allows to estimate the DTFT at specific values of Ω using Eq. (2.30). Also, it is the fastest in the direct transform because there is no need for multiplication by a factor.
- **Unitary DFT**: $\alpha = \beta = 1/\sqrt{N}$. Because the transform is unitary, Theorem 2 applies and the energy `sum(abs(X).^2)` in frequency and time `sum(abs(x).^2)` domains coincide. To implement the unitary DFT when using Matlab/Octave one can call `X=fft(x)/sqrt(N)` and `x=ifft(X)*sqrt(N)` for the direct and inverse transforms, respectively.
- **DTFS**: $\alpha = 1/N$ and $\beta = 1$. The spectrum coefficients can be interpreted in Volts, as obtained from the Fourier series of the input signal (assumed to be in Volts). To implement the DTFS when using Matlab/Octave one can call `X=fft(x)/N` and `x=ifft(X)*N` for the direct and inverse transforms, respectively.

The next two sections present the Laplace and Z transforms.[7] Both have two parameters that identifies each basis functions, in contrast to Fourier equations that have only the frequency. This extra degree of freedom, represented by sets of basis functions that are more powerful than the ones used in Fourier analysis, has the advantage of allowing the representation of a larger class of signals (and systems). A disadvantage is that the inverse transform is defined as a *contour integration* in the complex plane, which is an area of *complex analysis* that is out of the scope of this text. But it is rarely necessary to calculate the inverse transform using complex analysis. The Laplace and Z inverse transforms will be calculated here only by an alternative method called partial fraction expansion, which is discussed in the Appendix, Section A.10.

[7] There are lots of materials on the Web. For example, check: [url2pol], [url2lap] and [url2vid].

The Laplace and Z transforms are widely used to represent the action of linear and time-invariant systems, as will be detailed in Chapter 3. The Laplace transform has a number of properties that make it useful for analyzing linear systems. The most prominent advantage is that differentiation and integration become multiplication and division, respectively, by s. For example, this converts differential equations into polynomial equations, which are much easier to solve and, once solved, the inverse Laplace transform can provide the time domain formula. Similarly, Z transforms are useful to analyze discrete-time systems.

2.8 Laplace Transform

Some signals do not have a representation using Fourier analysis. For example, the signal $x(t) = e^{3t}u(t)$, does not have a Fourier transform because the integral

$$\begin{aligned} X(\omega) &= \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt = \int_{0}^{\infty} e^{(3-j\omega)t}dt = \frac{1}{3-j\omega}e^{(3-j\omega)t}\Big|_0^{\infty} \\ &= \frac{1}{3-j\omega}\lim_{t\to\infty}\frac{e^{3t}}{e^{j\omega t}} = \infty \end{aligned}$$

does not converge. Note that one should evaluate

$$\lim_{t\to\infty}\frac{e^{3t}}{e^{j\omega t}}$$

considering that $|e^{j\omega t}| = 1$ (this limit is not ∞/∞ and L'Hospital's rule is not appropriate).

This limitation in the representational power of Fourier analysis can be circumvented with the trick of pre-multiplying the input signal by an exponential. Assuming the continuous-time domain, this corresponds to multiplying $x(t)$ by an exponential $e^{-\sigma t}$, $\sigma \in \mathbb{R}$, and then taking the Fourier transform of $x(t)e^{-\sigma t}$. For example, if $x(t) = e^{3t}u(t)$, then pre-multiplying by e^{-4t} leads to the following transform for $z(t) = x(t)e^{-4t} = e^{-t}u(t)$:

$$Z(\sigma,\omega) = \int_{-\infty}^{\infty} z(t)e^{-j\omega t}dt = \int_{0}^{\infty} e^{(-1-j\omega)t}dt = \frac{1}{1+j\omega}.$$

The signal $x(t)$ could be recovered from the inverse transform of $Z(\sigma,\omega)$ and a post-multiplication by e^{4t}. This trick can be adopted with several different values of σ. For example, $z(t) = x(t)e^{-5t}$ would also work in the previous example, as well as all values of $\sigma < -3$.

Based on this motivation, the Laplace transform $X(s)$, where $s = \sigma + j\omega$ is a complex number, can be interpreted as the Fourier transform of the signal multiplied by a generic exponential, as follows:

$$X(s) = \mathcal{F}\{x(t)e^{-\sigma t}\} = \int_{-\infty}^{\infty} x(t)e^{-st}dt.$$

Hence, from another perspective, the Laplace transform uses basis functions $e^{(-\sigma+j\omega)t}$ with two parameters σ and ω. Instead of choosing a specific value of σ, when dealing with the Laplace transform, one considers all possible values. The values for which the associated Fourier transform $\mathcal{F}\{x(t)e^{-\sigma t}\}$ converges compose the so-called *region of convergence* (ROC) of the Laplace transform. For the previous example $x(t) = e^{3t}u(t)$, the ROC is $\sigma < -3$.

It is intuitive that the Fourier basis functions, which are limited in amplitude, are not the most adequate to represent a signal such as the $x(t) = e^{3t}u(t)$, which has magnitude increasing with t for $t > 0$. Consider picking a very large value $X(f')$ for the coefficient corresponding to the basis function $e^{j2\pi f't}$, this value $X(f')$ would scale the basis along all the time axis, eventually failing to provide the necessary increase in magnitude. The Laplace transform multiplies $x(t)$ by an exponential weighting function $e^{\sigma t}$, where $\sigma \in \mathbb{R}$ can control the amplitude of the exponential, which increases and decreases with t for $\sigma > 0$ and $\sigma < 0$, respectively. The step is equivalent to adding a new basis function, which is an exponential sinusoid $e^{j\omega t}$ multiplied by $e^{\sigma t}$, i. e., $e^{(\sigma+j\omega)t}$.

The left graph in Figure 2.11 was obtained with Listing 2.4 and the second graph used sigma = -0.3, with both showing the real part of e^{st}, with $s = \pm 0.3 + j10\pi$. The envelope $e^{\pm 0.3t}$ imposes the peak amplitude of the sinusoid $e^{j10\pi t}$ of 5 Hz.

Listing 2.4: *MatlabOctaveCodeSnippets/snip_transforms_laplace_basis.m*

```
sigma = 0.3; w = 2*pi*5; %frequency of 5 Hz
s=sigma+j*w; %define the complex variable s
t=linspace(0,3,1000); %interval from [0, 3] sec.
x=exp(s*t); %the signal
envelope = exp(sigma*t); %the signal envelope
subplot(121);plot(t,real(x));hold on,plot(t,envelope,':r')
```

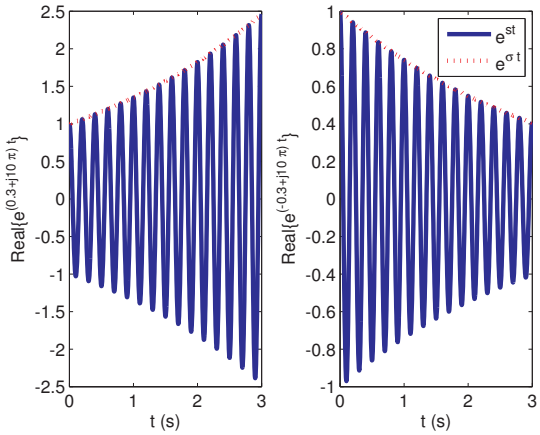


Figure 2.11: *Real part of $e^{\sigma+j10\pi}$. The values of σ are 0.3 and -0.3 for the first (left) and second graphs, respectively. The envelope $e^{\pm 0.3t}$ imposes the peak amplitude of the sinusoid $e^{j10\pi t}$ of 5 Hz.*

The basis functions e^{st} have some peculiarities. If $\sigma > 0$ their amplitudes reach ∞ when $t \to \infty$. If $\sigma < 0$ their amplitudes reach ∞ when $t \to -\infty$. Therefore, the Laplace transform is more useful in the analysis of one-sided signals, such as the ones that incorporate $u(t)$ (right-sided) or $u(-t)$ (left-sided). In these cases, the basis function do not reach infinite because the ROC is chosen in a way that the signal itself is zero at the problematic values $t = \infty$ or $-\infty$.

In fact, the choice of e^{st} and the corresponding popularity of the Laplace transform is due to their use as a tool for analyzing systems (see Chapter 3) not signals. Make a comparison with the Fourier transform: if the signal coincides with a given basis function (e. g., $x(t) = \cos(\omega_0 t)$), a signal impulse represents the signal in the transform domain ($X(\omega) = \pi\delta(\omega - \omega_0)$ for the given example). Such situations do not occur when the Laplace transform is used. The transform domain $X(s)$ does not use impulses and there is no signal $x(t)$ that is represented by a unique basis function.

The pair of Laplace equations is:

$$\begin{cases} X(s) &= \int_{-\infty}^{\infty} x(t)e^{-st}dt \\ \\ x(t) &= \frac{1}{2\pi j}\oint_{\gamma-j\infty}^{\gamma+j\infty} X(s)e^{st}ds, \end{cases}$$

where γ is a real number so that the contour path of integration is in the region of convergence of $X(s)$.

Because there are two parameters, the values that s can assume compose a plane while in Fourier analysis it was a line (frequency in the abscissa). For example, assuming that $x(t) = \frac{1}{5}\left[e^{-t}(3\cos(2t) + \sin(2t)) - 3e^{-2t}\right] u(t)$, the Laplace transform is

$$X(s) = \frac{s-1}{(s+2)(s+1-j2)(s+1+j2)} \tag{2.42}$$

with ROC $\sigma > -1$. Figure 2.12 and Figure 2.13 show the magnitude (in dB) and phase of $X(s)$, respectively.

The values for which $X(s) = 0$ are called *zeros* and the ones that lead to $X(s) = \infty$ are called *poles*. Figure 2.12 illustrates the zero at $s = 1$ and the tree poles. The locations of zeros appear as "valleys" while the poles are located at "volcanoes". It is intuitive that the ROC cannot include poles because they are the positions for which $X(s) = \infty$. Because the convergence depends on σ, the ROCs are regions formed at the left or right of a given value σ_0. When $x(t)$ is right-sided, the ROC is the area at the right of σ_0, while for left-sided $x(t)$ the ROC is the area at the left of σ_0. For the example in Figure 2.12, the ROC is $\sigma > -1$ because $x(t)$ is right-sided and the right-most poles (assuming the orientation of the σ-axis is from $-\infty$ to $+\infty$) are at $\sigma_0 = -1$.

Figure 2.13 shows the phase of $X(s)$. In most cases the phase is less instructive than the magnitude. Figure 2.13 emphasizes that the poles are conventionally signalized with "x" marks while zeros are represented by "o" marks.

The Laplace transform has redundancy and $x(t)$ can be recovered from $X(s)|_{\sigma=\sigma_1}$ if

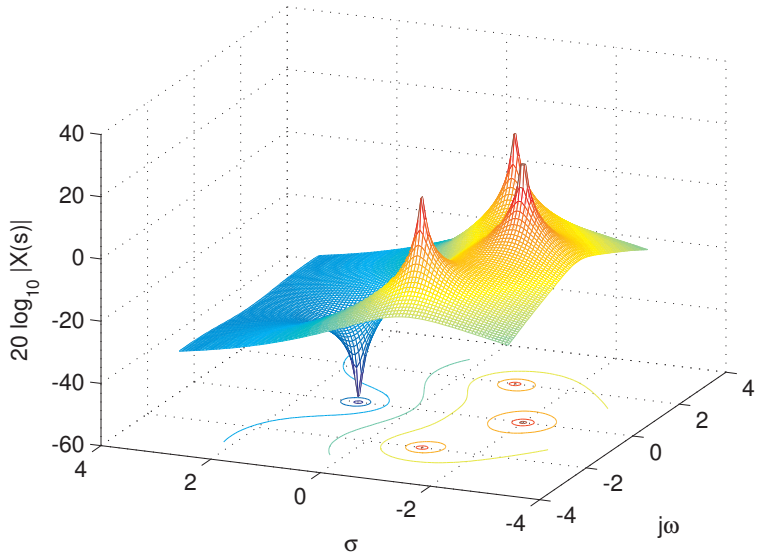


Figure 2.12: *Magnitude (in dB) of Eq. (2.42).*

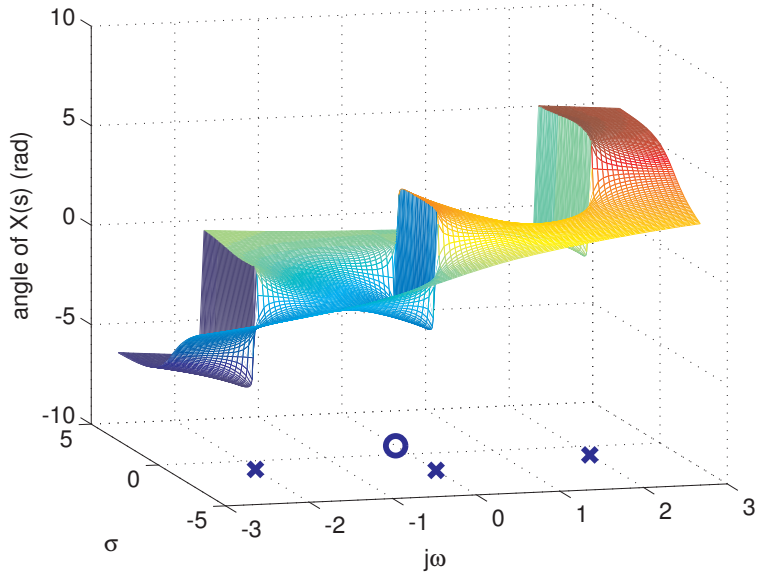


Figure 2.13: *Phase (in rad) of Eq. (2.42).*

σ_1 is in the ROC. An alternative view is that $x(t)$ can be recovered from the knowledge of $X(s)$ and its associated ROC. Note that knowing $X(s)$ does not suffice to to uniquely identify $x(t)$: the ROC must be known too. Distinct signals $x(t)$ can have the same $X(s)$, differing only in the ROCs. The following example illustrates this point.

Example 2.11. **Distinct signals may have the same Laplace transform, and only the ROC can disambiguate.** Given the Laplace transform $X(s) = 1/(s+2)$,

one should find the corresponding signal $x(t)$. In fact, there are two possible signals, depending on the ROC. The pole is at $s = -2$ and assuming the ROC is $\mathcal{R}eal\{s\} > -2$, then $x(t) = e^{-2t}u(t)$. If the ROC is $\mathcal{R}eal\{s\} < -2$, then $x(t) = -e^{-2t}u(-t)$. □

The Fourier transform basis function is equivalent to the Laplace's when $\sigma = 0$. Knowing that $X(s) = \mathcal{F}\{x(t)e^{-\sigma t}\}$ the Fourier transform of $x(t)$ can be derived from its Laplace transform, i. e.,

$$X(\omega) = X(s)|_{\sigma=0} = X(s)|_{s=j\omega} \tag{2.43}$$

in case $\sigma = 0$ is in the ROC of $X(s)$ and, consequently, $x(t)$ has a Fourier transform. Repeating to emphasize: Eq. (2.43) is valid only if the signal $x(t)$ has a Fourier transform. Using the previous example, note that the axis $j\omega$ is part of the ROC of $e^{-2t}u(t)$, which then has Fourier transform while $-e^{-2t}u(-t)$ does not have.

When the signal has both Fourier and Laplace transforms, a graph of $X(s)$ incorporates the corresponding graph of $X(\omega)$. Figure 2.14 shows the magnitude values for $s = j\omega$ (the $j\omega$ axis) as a curve in black, superimposed to Figure 2.12. Alternatively, Figure 2.15 shows only the values of $X(s)$ at the $s = j\omega$ axis.

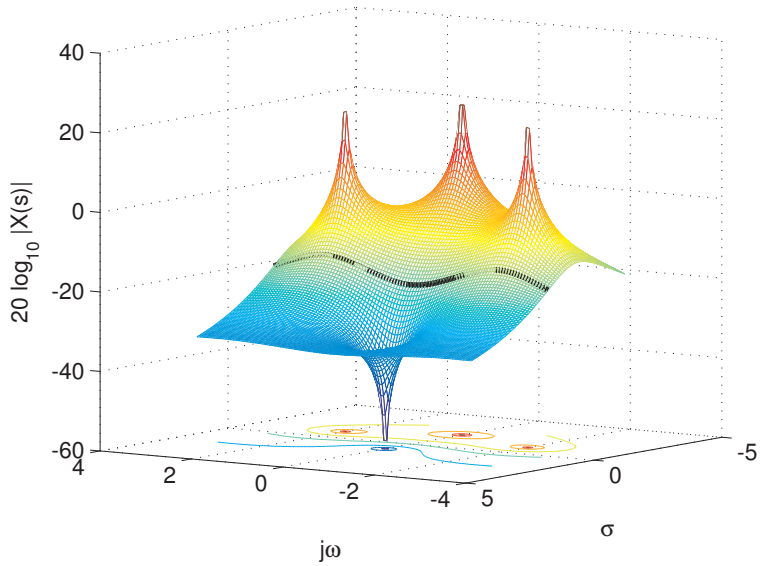


Figure 2.14: *Graph of Figure 2.12 with the identification of the corresponding values of the Fourier transform (magnitude).*

Both representations in Figure 2.14 and Figure 2.15 are three-dimensional and difficult to work with. Figure 2.16 depicts the frequency response in the conventional way, which is much easier to interpret than looking at Figure 2.15, for example. However, it is interesting to compare the figures and note that the peaks in Figure 2.16 are related to the position of the poles and draw conclusions such as that, the closer the pole is to the $\sigma = 0$ axis, the more evident is the corresponding peak at the Fourier transform.

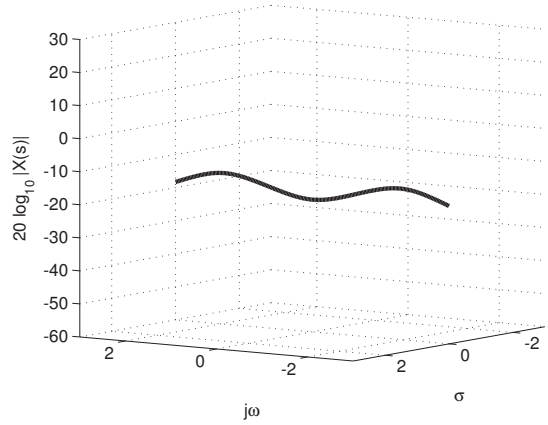


Figure 2.15: *The values of the magnitude of the Fourier transform corresponding to Figure 2.14.*

In filter design, it is sometimes necessary to locate poles in the vicinity of the ω axis because this conducts to filters with high *quality factor*.

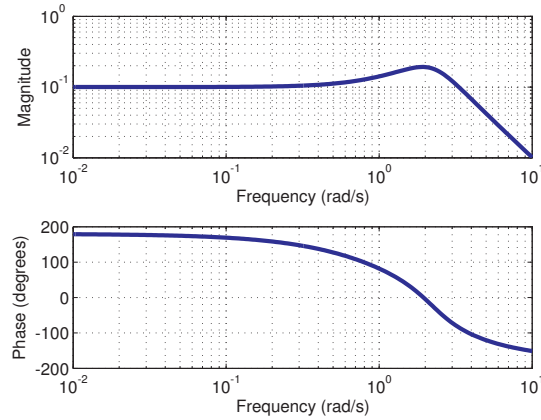


Figure 2.16: *Two dimensional representation of Figure 2.15 obtained with the command* freqs *in Matlab/Octave showing the peak at $\omega = 2$ rad/s due to the respective pole. Because the signal $x(t)$ is real and $X(\omega)$ exhibits Hermitian symmetry, only the graphs for $\omega \geq 0$ are shown.*

The need to use the Laplace as an alternative to the Fourier transform would be even greater if impulses were not allowed in Fourier analysis. The use of impulses in both the time and frequency domains allows a much larger class of signals to be represented with Fourier equations. For example, the *ramp* signal $x(t) = t\ u(t)$, strictly does not have a Fourier transform because the following integral does not converge:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt = \int_{0}^{\infty} te^{-j\omega t}dt = \frac{1 + j\omega t}{\omega^2 e^{j\omega t}}\bigg|_0^{\infty} = \lim_{t\to\infty} \frac{1 + j\omega t}{\omega^2 e^{j\omega t}}.$$

However, using impulses, it is possible to use Fourier representations of signals such as $x(t) = t\ u(t)$, $x(t) = u(t)$ and $x(t) = \cos(\omega_0 t)$. This fact influences the main applications

of the Laplace transform to be the representation of systems, not signals. For example, in terms of signals, while $x(t) = \cos(\omega_0 t)u(t)$ has the transform $X(s) = s/(s^2 + \omega_0^2)$, an infinite-duration sinusoid is not represented in the Laplace transform domain.

Because most signals that are analyzed with the Laplace transform are right-sided, sometimes the adopted definition is

$$X(s) = \int_0^\infty x(t)e^{-st}dt, \tag{2.44}$$

which is called the *unilateral Laplace transform* (in contrast to the bilateral definition of Eq. (2.8)). Both coincide if the signal is right-sided (e. g., by the action of $u(t)$).

The Laplace transform properties and pairs are similar to the Fourier ones as indicated by the following example.

Example 2.12. **A time-domain complex exponential leads to a rational function in s.** Consider the following Laplace transform pair

$$e^{at}u(t) \Leftrightarrow \frac{1}{s-a}, \quad \text{converges for: } \mathcal{R}eal\{s\} > a$$

that is proved as follows:

$$\begin{aligned} X(s) &= \int_{-\infty}^{\infty} x(t)e^{-st}dt = \int_0^\infty e^{(a-s)t}dt = \frac{1}{a-s}e^{(a-s)t}\Big|_0^\infty \\ &= \frac{1}{a-s}\left[(\lim_{t\to\infty} e^{(a-s)t}) - 1\right]. \end{aligned}$$

Given that $s = \sigma + j\omega$, the limit can be obtained as

$$\lim_{t\to\infty} e^{(a-s)t} = \lim_{t\to\infty} \frac{e^{(a-\sigma)t}}{e^{j\omega t}} = 0$$

when $(a - \sigma) < 0$ (note $|e^{j\omega t}| = 1$ and only the numerator defines the convergence). □

2.9 Z Transform

The Z transform is the counterpart of the Laplace transform for discrete-time signals. The pair of equations is given by

$$\begin{cases} X(z) &= \sum_{n=-\infty}^{\infty} x[n]z^{-n} \\ \\ x[n] &= \frac{1}{2\pi j}\oint_{\mathcal{C}} X(z)z^{n-1}dz, \end{cases}$$

where $\mathcal{C}$ is a counterclockwise closed path encircling the origin and entirely in the region of convergence (ROC). The contour or path, $\mathcal{C}$, must encircle all of the poles of $X(z)$.

The Laplace and Z transforms are related. When the Laplace transform is performed on a sampled signal $x_s(t)$ and a C/D is used, the result is the Z transform[8] of a discrete-time sequence $x[n]$ where

$$z = e^{sT_s} \tag{2.45}$$

and T_s is the sampling period.

Eq. (2.45) is used in the *matched Z-transform* method for converting $H(s)$ in s into $H(z)$ in z, and is further discussed in Section **??**.

2.9.1 Some pairs and properties of the Z-transform

A very useful pair and property are $\delta[n] \Leftrightarrow 1$ and $x[n-n_0] \Leftrightarrow X(z)z^{-n_0}$, respectively. Putting them together leads to $\delta[n-n_0] \Leftrightarrow z^{-n_0}$. Using linearity one can write, e. g.,

$$3\delta[n+4] + 2\delta[n] - 2.5\delta[n-3] \Leftrightarrow 3z^4 + 2 - 2.5z^{-3}.$$

In some cases $X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$ can be written as a geometric series and Eq. (A.17) used to obtain $X(z)$. For example, the Z transform of $x[n] = a^n u[n]$ is obtained as follows:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} = \sum_{n=-\infty}^{\infty} a^n u[n]z^{-n} = \sum_{n=0}^{\infty} a^n z^{-n} = \sum_{n=0}^{\infty} (a/z)^n.$$

Using Eq. (A.17) with a scale factor $\alpha = 1$, ratio $r = az^{-1}$ and $|a/z| < 1$ leads to

$$a^n u[n] \Leftrightarrow \frac{1}{1-az^{-1}} = \frac{z}{z-a}, |z| > |a|\,. \tag{2.46}$$

Example 2.13. **Converting $x(t)$ to discrete-time and finding the corresponding Z-transform**. As in Example 1.2, assume a continuous-time signal $x(t) = 4e^{-2t}u(t)$ should be transformed to a discrete-time $x[n]$ with a sampling period T_s, and then have its Z-transform $X(z) = \mathcal{Z}\{x[n]\}$ calculated. From Eq. (1.9):

$$X(z) = \mathcal{Z}\left\{4e^{-2nT_s}u[n]\right\} = 4\mathcal{Z}\left\{\left(e^{-2T_s}\right)^n u[n]\right\} = \frac{4}{1-e^{-2T_s}z^{-1}}, \tag{2.47}$$

where the last step used Eq. (2.46). □

2.9.2 Z-transform region of convergence

Similar to the Laplace transform, the values of z for which the transform exists are called the region of convergence (ROC). The ROC of Z transforms are annular regions

[8] Some textbooks (e. g. [Cio10]) that deal with coding theory use the D instead of the Z transform, where $D = z^{-1}$. For example, $H(z) = 1-0.9z^{-1}$ corresponds to $H(D) = 1-0.9D$ and $H(z) = 1-0.5z$ corresponds to $H(D) = 1-0.5D^{-1}$.

of the form $|z| > m$ (for right-sided sequences), $|z| < m$ (for left-sided sequences) or $m < |z| < p$ (for finite duration sequences), where $m, p \in \mathbb{R}_+$.

To recover $x[n]$ from its transform $X(z)$, it is essential to know the ROC. For example, the Z transform of $x[n] = -a^n u[-n-1]$ is:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n} = -\sum_{n=-\infty}^{-1} a^n z^{-n} = -\sum_{n=1}^{\infty} a^{-n} z^n = -\sum_{n=1}^{\infty} (z/a)^n.$$

In order to use Eq. (A.17) with factor $\alpha = 1$ and ratio $r = z/a$ one can modify the summation interval

$$X(z) = -\sum_{n=1}^{\infty} (z/a)^n = 1 - \sum_{n=0}^{\infty} (z/a)^n = 1 - \frac{1}{1 - z/a} = \frac{z}{z-a},$$

with the ROC $|z| < |a|$ (because Eq. (A.17) requires $|z/a| < 1$). In summary, both $a^n u[n]$ and $-a^n u[-n-1]$ have $X(z) = z/(z-a)$ and only the ROC can disambiguate them when calculating the inverse Z transform.

The inverse transform of rational functions can be obtained by following the steps:

1. make the rational function to have only non-negative[9] powers of z,
2. find the poles and expand $X(z)$ in partial fractions as discussed in Section A.10,
3. eventually multiply by z to create $Y(z) = zX(z)$ and force the appearance of terms term $z/(z-a)$,
4. convert each parcel of $Y(z)$ to the time domain,
5. rearrange the terms, especially the ones corresponding to complex conjugate poles and
6. because $Y(z) = zX(z)$, then $x[n] = y[n-1]$. Find the final result substituting n by $n-1$ in $y[n]$.

For example, to obtain the inverse transform of

$$X(z) = \frac{z^{-2} + 0.9z^{-3}}{1 - 1.8z^{-1} + 1.41z^{-2} - 0.488z^{-3}},$$

with ROC $|z| > 0.8$, one can multiply numerator and denominator by z^3 and obtain their roots:

$$X(z) = \frac{z + 0.9}{z^3 - 1.8z^2 + 1.41z - 0.488} = \frac{z + 0.9}{(z - 0.8)(z - 0.5 \pm j0.6)}.$$

[9] This is not mandatory, but allows to use the same procedure for partial fraction expansion as for the Laplace transform.

The partial fraction expansion is

$$X(z) = \frac{3.78}{z-0.8} + \frac{-1.89+j0.11}{z-0.5+j0.6} + \frac{-1.89-j0.11}{z-0.5-j0.6}.$$

Multiplying both sides by z leads to

$$Y(z) = zX(z) = \frac{3.78z}{z-0.8} + \frac{(-1.89+j0.11)z}{z-0.5+j0.6} + \frac{(-1.89-j0.11)z}{z-0.5-j0.6},$$

which can be rewritten by converting the complex numbers from Cartesian to polar form

$$Y(z) = \frac{3.78z}{z-0.8} + \frac{(1.89e^{j3.08})z}{z-0.78e^{j2.26}} + \frac{(1.89e^{-j3.08})z}{z-0.78e^{-j2.26}},$$

Because in this case the ROC is for a right-sided sequence, each term $z/(z-a)$ corresponds to $a^n u[n]$ and the time domain signal is

$$\begin{aligned} y[n] &= [3.78(0.8)^n + 1.89e^{j3.08}(0.78e^{j2.26})^n + 1.89e^{-j3.08}(0.78e^{-j2.26})^n]u[n] \\ &= [3.78(0.8)^n + 1.89(0.78)^n(e^{j(3.08+2.26n)} + e^{-j(3.08+2.26n)})]u[n] \\ &= [3.78(0.8)^n + 2 \times 1.89(0.78)^n \cos(2.26n + 3.08)]u[n] \end{aligned}$$

which leads to

$$x[n] = [3.78(0.8)^{n-1} + 2 \times 1.89(0.78)^{n-1} \cos(2.26(n-1) + 3.08)]u[n-1].$$

Note that a pair of complex conjugate poles have complex conjugate residues. Let $r = be^{j\alpha}$ and $r^* = be^{-j\alpha}$ be the residues for poles $p = ae^{j\theta}$ and $p^* = ae^{-j\theta}$, respectively, both with multiplicity one. With a ROC corresponding to right-sided signals, the two terms $rz/(z-p)$ and $r^*z/(z-p^*)$ in the partial fraction expansion can be rearranged in time domain to compose the general expression

$$2ba^n \cos(\theta n + \alpha)u[n].$$

If the ROC corresponds to left-sided signals, the same terms correspond to

$$-2ba^n \cos(\theta n + \alpha)u[-n-1].$$

Different approaches to obtain $x[n]$ can lead to distinct expressions, but these expressions must correspond to the same values of $x[n], \forall n$. For example, some people prefer to obtain the partial fraction expansion of $X(z)/z$ instead of using the suggested steps 3) and 6). Expanding $X(z)/z$ allows to multiply the obtained partial fractions by z to get $z/(z-a)$ factors. An example better illustrates the equivalence of both procedures and the reason for suggesting ours.

Assume the task is to find the inverse $x[n]$ of $X(z) = (8z-19)/[(z-2)(z-3)]$ knowing that $x[n]$ is right-sided. Using the alternative procedure of expanding $X(z)/z$,

one has

$$\frac{X(z)}{z} = (8z - 19)/[z(z-2)(z-3)] = -\frac{19/6}{z} + \frac{3/2}{z-2} + \frac{5/3}{z-3},$$

which can be conveniently multiplied by z to obtain parcels in the form $z/(z-a)$:

$$X(z) = -\frac{19}{6} + 3/2\frac{z}{z-2} + 5/3\frac{z}{z-3},$$

that leads to the inverse

$$x[n] = -\frac{19}{6}\delta[n] + \left(\frac{3}{2}2^n + \frac{5}{3}3^n\right)u[n]. \tag{2.48}$$

Using the steps of the suggested procedure, which expands $X(z)$ in partial fractions instead of $X(z)/z$, the result is

$$x[n] = [5(3^{n-1}) + 3(2^{n-1})]u[n-1],$$

which seems different than Eq. (2.48). However, a closer inspection indicates that, for both expressions, the sample values $x[0] = 0, x[1] = 8$, etc., are the same, i. e., the procedures led to the same signal, as expected.

The Web has several tables of properties and pairs related to the Z transform.[10] An interesting result is the *initial value theorem*, which is valid for signals for which $x[n] = 0$ for $n < 0$ (right-sided) and states that

$$x[0] = \lim_{z\to\infty} X(z).$$

Using this theorem, one can anticipate that $x[0] = 0$ when $X(z)$ has a denominator with degree larger than the numerator, such as in $X(z) = (8z - 19)/[(z-2)(z-3)]$.

Similar to Figure 2.12 and Figure 2.13, which are for Laplace, Figure 2.17 and Figure 2.18 depicts the magnitude and phase, respectively, for

$$X(z) = \frac{z + 0.9}{(z - 0.8)(z - 0.5 - j0.6)(z - 0.5 - j0.6)}, \tag{2.49}$$

which has one finite zero and three poles as indicated in Figure 2.19.

The relation between the Z transform and the DTFT is similar to the one between Laplace and Fourier transform. Figure 2.20 and Figure 2.21 provide an example using Eq. (2.49).

Sometimes it is not convenient to deal with 3-d plots such as Figure 2.21. An alternative is to represent the DTFT using a figure similar to Figure 2.22, which shows the magnitude and phase with the angle as independent variable. As discussed in Section 1.8.2, for convenience, the abscissa is normalized by π, such that "1" corresponds

[10] Such as the ones at [url2ztr].

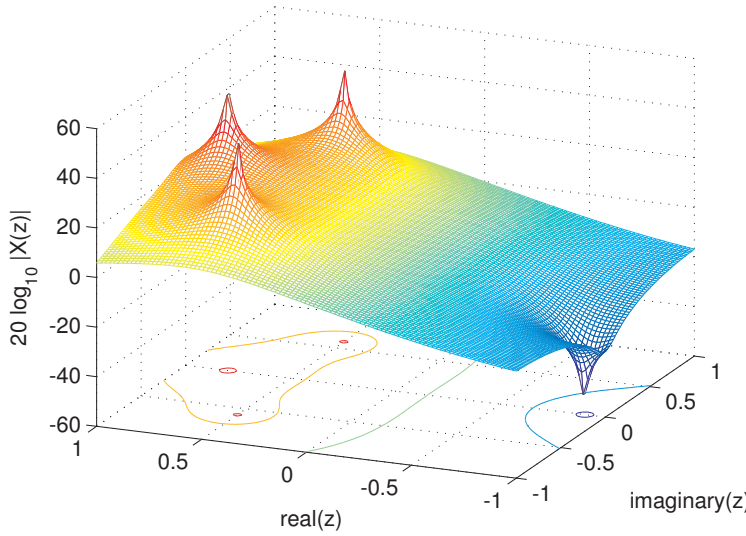


Figure 2.17: *Magnitude (in dB) of Eq. (2.49).*

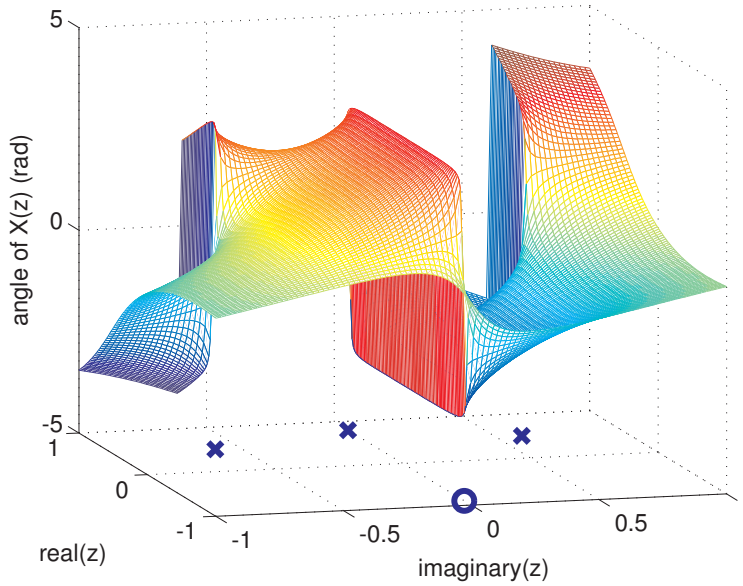


Figure 2.18: *Phase (in rad) of Eq. (2.49).*

to π rad. Due to the symmetry of $X(z)$ when $x[n]$ is real, it is also common to represent the abscissa in the range $[0, \pi]$ (instead of $[0, 2\pi[$ as in Figure 2.22).

2.10 Applications

This section will briefly discuss some applications of transforms.

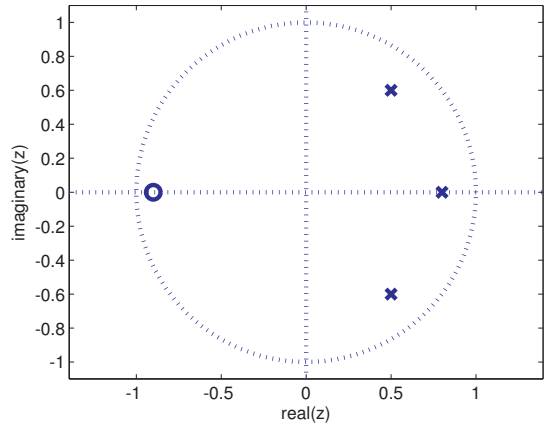


Figure 2.19: *Pole / zero diagram for Eq. (2.49).*

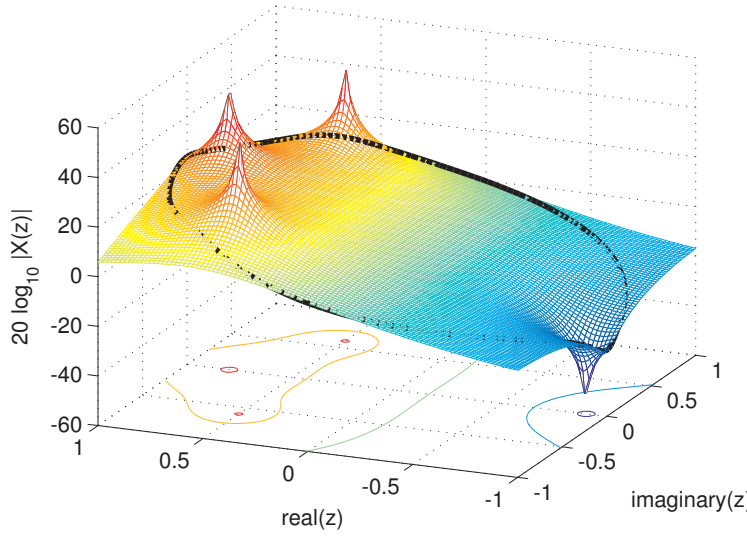


Figure 2.20: *Graph of Figure 2.17 with the identification of the corresponding values of the DTFT (unity circle $|z| = 1$).*

Application 2.1. Example of Gram-Schmidt transform. As an example of the Gram-Schmidt procedure, assume that Listing A.4 is invoked with the commands

```
x=[0,-1,-1,0;  0,2,2,0;  0,1,0,1;  1,1,1,1;  -2,2,2,1] %row vectors
[Ah,A]=ak_gram_schmidt(x) %perform the orthonormalization procedure
X=Ah*transpose(x(1,:)) %coefficients corresponding to first vector
x1=A*X; %reconstruction of first vector x(1,:) (as a column vector)
```

where $M = 5$ vectors and $D = 4$ is the space dimension. In this case, the number of orthonormal basis functions is $N = 4$, such that both matrices Ah and A have dimension 4×4. The basis functions correspond to the columns of matrix A. The first element of x1, corresponding to the first basis function, is the only non-zero element and has value $\sqrt{2}$ which is the coefficient that must multiply the first basis function

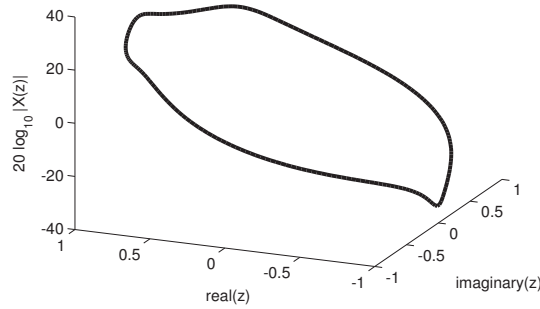


Figure 2.21: *The values of the magnitude of the DTFT corresponding to Figure 2.20.*

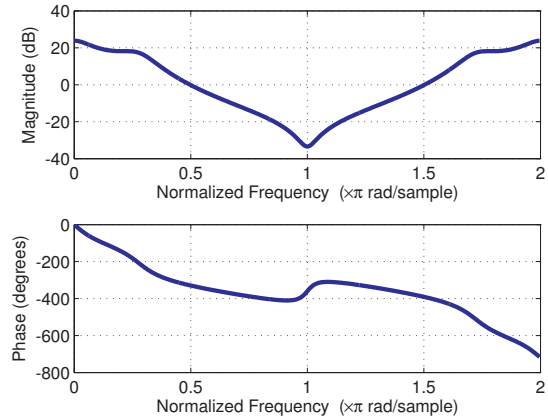


Figure 2.22: *Magnitude (top) and phase (bottom) of the DTFT corresponding to Eq. (2.49). These plots can be obtained with the Matlab/Octave command* freqz *and are a more convenient representation than, e. g., Figure 2.21.*

$[0, -\sqrt{2}/2, \sqrt{2}/2, 0]$ to reconstruct the first vector $[0, -1, -1, 0]$. The reason is that ak_gram_schmidt chooses the first basis as a normalized (unitary-norm) version of the first vector in x.

Debugging the execution of the code ak_gram_schmidt, step-by-step, allows to observe more details as listed in Listing 2.5.

Listing 2.5: *MatlabOctaveCodeSnippets/snip_transforms_granschmidt_debug.m*

```
tol = 1.1102e-015  %calculated (default) tolerance
%first basis in y is [0  -0.7071  -0.7071  0], numBasis=1
k = 2, m = 1
projectionOverBasis = [0 2.0 2.0 0] %2nd input vector
errorVector = 1.0e-015 * [0    0.4441    0.4441  0]
magErrorVector = 6.2804e-016 %do NOT add error vector to basis set
%2nd vector is already represented, go to next iteration
k = 3, m = 1
```

```
projectionOverBasis = [0 0.5 0.5 0] %3rd input vector
%errorVector below is orthogonal to [0 -0.7071 -0.7071 0]
errorVector = [0 0.5 -0.5 1]
magErrorVector = 1.2247 %add normalized error vector to basis set
%second basis is [0 0.4082 -0.4082 0.8165], numBasis=2
k = 4, m = 1
projectionOverBasis = [0 1.0 1.0  0] %4th input vector
errorVector = [1.0 0.0 0.0 1] %using 1st basis
k = 4, m = 2
projectionOverBasis = [0.0  0.3333   -0.3333    0.6667]
errorVector = [1.0 -0.3333 0.3333 0.3333] %using 2 basis vectors
magErrorVector = 1.1547 %add normalized error vector to basis set
%3rd basis is [0.8660 -0.2887 0.2887 0.2887], numBasis=3
k = 5, m = 1
projectionOverBasis = [0 2.0 2.0 0] %5th input vector
errorVector = [-2.0 0.0 0.0 1.0] %using only 1st basis
k = 5, m = 2
projectionOverBasis = [0 0.3333 -0.3333 0.6667]
errorVector = [-2.0 -0.3333 0.3333 0.3333] % using 2 basis vectors
k = 5, m = 3
projectionOverBasis = [-1.250 0.4167 -0.4167 -0.4167]
errorVector = [-0.75 -0.75 0.75 0.75] %using 3 basis vectors
magErrorVector = 1.5000 %add normalized error vector to basis set
%4th basis is [-0.5 -0.5 0.5 0.5], numBasis=4
%abort because (numBasis >= N)
```

The reader can perform a similar analysis for another set of vectors and compare with the results obtained via calculating by hand. □

Application 2.2. Time-localization property of Haar coefficients.

A very simple experiment will help illustrating the time-localization property of Haar functions. Consider the signal to be analyzed in the transform domain is a single impulse $x[n] = \delta[n - n_0], n_0 = 0, \ldots, 31$, which will be represented by a vector $\mathbf{x}$ will zeroed elements but the one corresponding to the n_0 position. The script MatlabOctaveBookExamples/ex_transforms_dcthaar_example.m can be used to observe the behavior of the transforms. Figure 2.23 shows the output of the script when the signal is $x[n] = \delta[n - 11]$. In this case, many Haar coefficients are zero and the one with largest magnitude ($k = 21$) has a corresponding basis function (bottom of the left column) that helps to localize the occurrence of the impulse in $x[n]$. In contrast, most DCT coefficients have relatively large values and the basis functions do not help in time-localization. The mentioned script allows to investigate this aspect more deeply. Compare the results of DCT and Haar transforms when the signal is a cosine summed to an impulse. □

Application 2.3. Inverse Laplace transforms with Matlab's Symbolic Math Toolbox. Keeping in mind that Matlab adopts the unilateral Laplace transform, its Symbolic Math Toolbox can be used to calculate Laplace transforms and their inverses. For example, the unilateral Laplace transform $X(s) = 1/s$ of $u(t)$ can be obtained with the commands syms t; laplace(t / t). Similarly, the commands syms

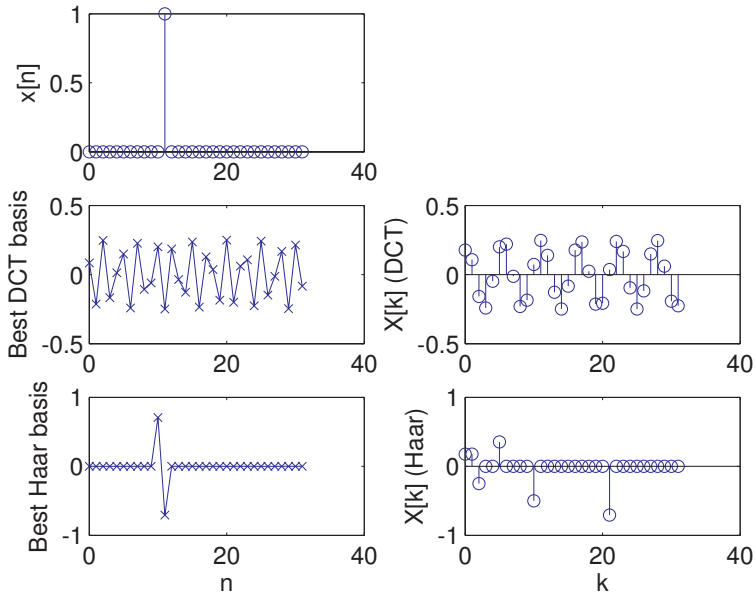


Figure 2.23: *Signal $x[n] = \delta[n-11]$ analyzed by 32-points DCT and Haar transforms. The right column shows the transform coefficients (abscissa is k). The left column shows time-domain plots. Below the signal itself, one can visualize the "best" (the one corresponding to the coefficient with largest absolute value) basis functions for each transform.*

w0,t; laplace(sin(w0*t)) indicate that $X(s) = \omega_0/(s^2 + \omega_0^2)$ for $x(t) = \sin(\omega_0 t)u(t)$. The following commands illustrate the inverse transform of $X(s)$ given by Eq. (2.42):

Listing 2.6: *MatlabOnly/snip_transforms_ilaplace.m*

```
syms s %defines s as a symbolic variable
a=1; b=-2; c=-1+j*2; %choose poles and zeros
X=(s-a)/((s-b)*(s-c)*(s-conj(c))); %define X(s)
ilaplace(X) %Matlab's inverse unilateral Laplace transform
```

The command pretty (e. g., pretty(ilaplace(X))) can be used to beautify the output, which in this case is - 3/5 exp(-2 t) + 1/5 exp(-t) (3 cos(2 t) + sin(2 t)). Find the inverse Laplace transform for other signals, such as $x(t) = \cos(\omega_0 t)u(t)$. □

Application 2.4. ECG transform coding. Because a block transform is completely specified by an invertible matrix $\mathbf{A}$, going from one domain to another ($\mathbf{x}$ to $\mathbf{X}$ or vice-versa) is *loss-less*, which means there is no loss of information. In many applications, such as image coding, where the goal is to minimize the number of bits to represent an image, it is useful to compress the signal using a lossy algorithm. Transform coding can be a lossy algorithm when it discards or quantizes coefficients in the transform domain. This is effective in many applications because, while all samples in $\mathbf{x}$ have the same "importance", in the transform domain $\mathbf{X}$ the coefficients can be organized according

to some hierarchy or rank. In transform coding the most "important" coefficients are quantized more carefully than the unimportant ones.

The ECG data is from the MIT-BIH Arrhythmia Database.[11] Figure 2.24 depicts a segment (first 1,500 samples) of the first channel of file 12531_04.dat. The function ak_rddata.m indicates that the DC offset of the ADC chip was calibrated to be zero. It also indicates that the ECG signals were digitized with a sampling frequency $F_s = 250$ Hz, 12 bits per sample, and a quantizer with step $\Delta_{\text{AD}} = 1/400$ mV. The (empirical) signal power is 3.07 mW and, following the conventional quantization model, Eq. (1.19) suggests the quantization noise power was 5.2×10^{-10} mW. Hence, the SNR corresponding to the quantization stage alone is approximately 97.7 dB.

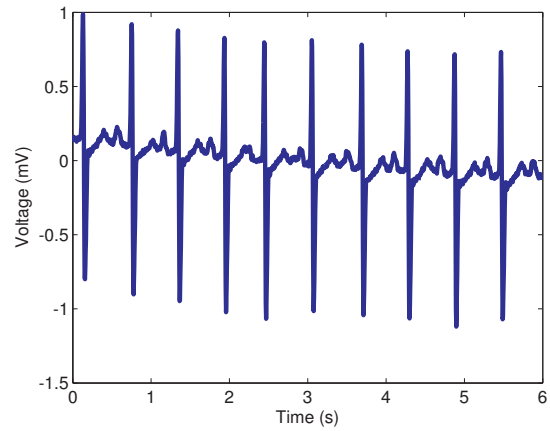


Figure 2.24: *A segment of one channel of the original ECG data.*

A very simple example of a transform coding system is provided as Matlab/Octave code in the companion software (in directory Applications/ECGTransformCoding). The idea is to simply discard the higher-order coefficients in the transform domain. For example, assume that N is the dimension of $\mathbf{x}$ and $\mathbf{X}$, which are related by a $N \times N$ transform matrix $\mathbf{A}$. In the encoding stage, a discrete-time input signal $x[n]$ is segmented into blocks of dimension N, composing a set of M input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$, where $N \times M$ is the available number of samples of $x[n]$. The coding scheme converts each $\mathbf{x}_i$ into $\mathbf{X}_i = A^H \mathbf{x}_i$ and keeps in a new vector $\hat{\mathbf{X}}_i$ only the first $K \leq N$ elements of $\mathbf{X}_i$ (the remaining $N - K$ are discarded). The M vectors $\hat{\mathbf{X}}_i$ can be concatenated to create an encoded output signal $\hat{X}[n]$ with $M \times K$ samples.

In the decoding stage, in order to reconstruct a signal $x'[n]$ from $\hat{X}[n]$ (if there were no losses, $x'[n] = x[n]$), an inverse procedure is adopted. The signal $\hat{X}[n]$ is blocked into vectors of dimension K and $N - K$ zero elements are inserted to create vectors $\mathbf{X}'_i$ of dimension N (this operation is called *zero-padding*). Each of these N-dimensional vectors is converted to time-domain vectors $\mathbf{x}'_i = \mathbf{A}\mathbf{X}'_i$, which are then concatenated to form $x'[n]$.

Figure 2.25 was obtained using a DCT matrix $\mathbf{A}$ of $N = 32$ points and discarding

[11] Available from [url2ecg].

26 (high-frequency) coefficients. It can be seen that keeping only $K = 6$ out of 32 coefficients is enough to provide a rough representation of $x[n]$ but the error is significant in high-frequency regions. Note that the DCT operation is performed in blocks of $N = 32$ samples and Figure 2.25 is the result of processing many of these blocks.

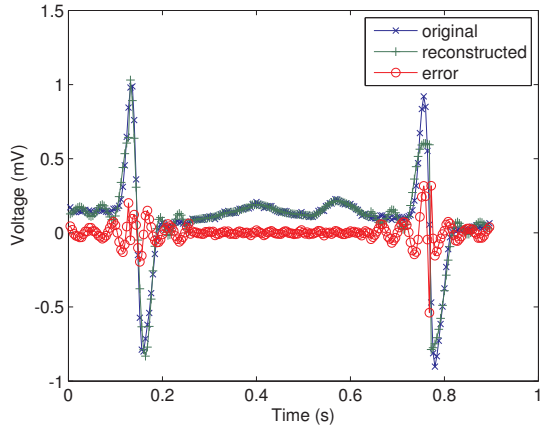


Figure 2.25: *Original and reconstructed ECG signals with DCT of $N = 32$ points and discarding 26 (high-frequency) coefficients. Therefore, the error is predominant in high-frequency regions as indicated in the plot.*

In practice, a quantization scheme should be adopted to represent $\hat{X}[n]$ with a small number of bits per sample. The compression ratio is the number of bits to represent $x[n], n = 0, \ldots, M \times N$ divided by the number of bits to represent $\hat{X}[n], n = 0, \ldots, M \times K$. For simplicity, the proposed example does not involve quantization and the compression ratio is evaluated by N/K. This corresponds to assuming that each sample of both $x[n]$ and $\hat{X}[n]$ is represented with the same number of bits.

Figure 2.26 shows the percentage of kept coefficients K/N in the abscissa and $10 \log_{10}$ MSE in the ordinate, where

$$\text{MSE} = \frac{1}{M \times N} \sum_{n=0}^{M \times N - 1} (x[n] - x'[n])^2$$

is the mean-squared error, which is equivalent to the power of the error signal. If $K = N$ (100% in the abscissa) the system is lossless and MSE $= 0$ ($-\infty$ in log scale), so the graphs do not show these points. The larger N, the better the coding performance but the higher the computational cost.

Assuming $N = 128$ and $K = 10$ as indicated in Figure 2.26, the abscissa is $K/N = 10/128 \approx 7.81\%$ and the corresponding error is MSE $= 1.4824 \times 10^{-8}$ W (-78.29 dBW) or, equivalently, -48.29 dBm. Given that the signal power was estimated as 3.07 mW (4.87 dBm), the $\text{SNR}_{\text{dB}} \approx 4.87 + 48.3 = 53.16$ dB when considering only the distortion incurred by this specific encoding procedure.

It should be noted that the dataset used is formed by abnormal ECG signals and, eventually, better results could be obtained when using another dataset.

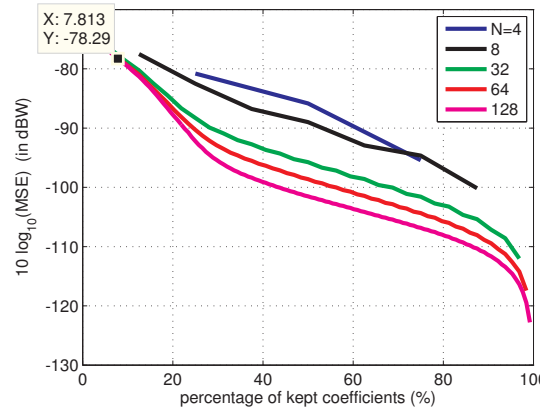


Figure 2.26: *Performance of five DCT-based ECG coding schemes. The number of points is varied* $N \in \{4, 8, 32, 64, 128\}$ *and* $K = 1, 2, \ldots, M-1$. *The larger* N, *the better the coding performance but the higher the computational cost.*

Write an encoder that uses DCT coding to generate files of a relatively small size and a decoder to "decompress" such files and recover an ECG signal (an approximation to the original one). To do that, it is necessary to quantize the DCT coefficients and pack them in a way that the resulting file has a small size. Consider using scalar or *vector quantization*. The advantage of vector quantization is that you need to store only the index to the codebook entry (codeword) with a vector that will represent the DCT. In case you use a codebook with $2^8 = 256$ codewords, each entry can be conveniently written as an unsigned char of 8 bits. To design the codebook, besides the Mathwork's code in the DSP System Toolbox, there are many algorithms such as the *K-means* with implementations available on the Web.

Another interesting exercise is to apply KLT and other transforms to this problem, trying to get better results than with the DCT. □

Application 2.5. DCT coding of image.

The usefulness of DCT in coding is illustrated by the JPEG image coding standard.[12] Before describing an example, two dimensional transforms are briefly discussed.

Linearly-separable two dimensional (2-D) transforms of a block $\mathbf{x}$ (e. g., a matrix with pixel values) is obtained by first using the transform along the rows (or columns) and then transforming this result along the columns (or rows). Alternatively, one can use matrix notation: $\mathbf{X} = \mathbf{A}^H \mathbf{x} \mathbf{A}^*$. Because, $\mathbf{A}$ is real for a DCT, one has

$$\mathbf{X} = \mathbf{A}^T \mathbf{x} \mathbf{A}.$$

This is equivalent to first calculating $\mathbf{T} = \mathbf{A}^H \mathbf{x}$: the 1-D DCT transform of each column of $\mathbf{x}$ is placed in its corresponding column in a temporary matrix $\mathbf{T}$. Then, each row of $\mathbf{T}$ is transformed by another 1-D DCT, which could be accomplished by $\mathbf{A}^H \mathbf{T}^T$. This

[12] A good way to visualize DCT in image coding is by running the (deprecated) dctdemo.m Matlab demonstration.

result should be transposed to generate $\mathbf{X}$. These operations are equivalent to:

$$\mathbf{X} = (\mathbf{A}^H(\mathbf{A}^H\mathbf{x})^T)^T = \mathbf{A}^H\mathbf{x}\mathbf{A}^*.$$

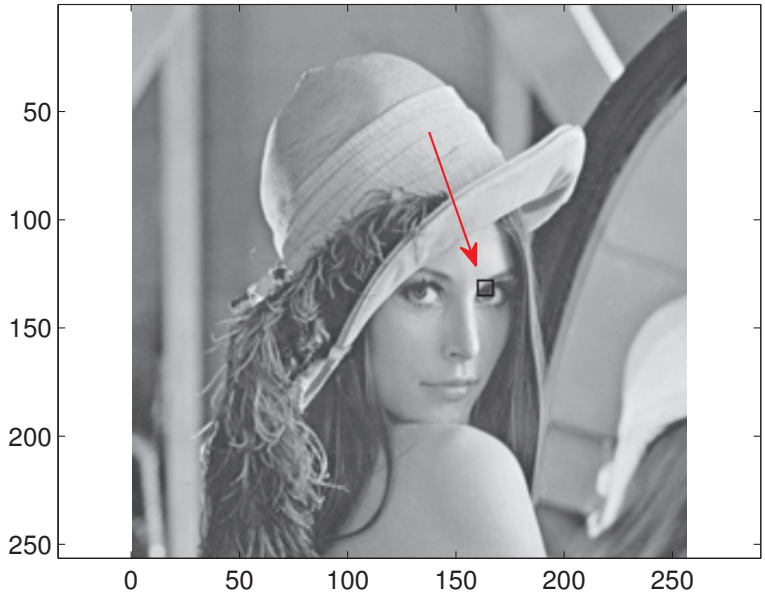


Figure 2.27: *A zoom of the eye region of the Lenna image.*

Listing 2.7 illustrates the adoption of a 2-D DCT for coding an image block (represented by a matrix).

Listing 2.7: *MatlabBookFigures/figs_transforms_dctimagecoding*

```
%load black & white 256 x 256 pixels of Lenna:
fullPath='../../BitmapOriginalFigures/lenna_bw.gif' %file location
  [lenaImage,map]=imread(fullPath); %read file
colormap(map); %use the proper color map
x1=128; x2=135; y1=160; y2=167; %to take a 8x8 block close to her eye
x=lenaImage(x1:x2,y1:y2); %get the region
x=double(x); %cast x to be double instead of integer
%draw a rectangle to indicate the block:
lenaImage(x1,y1:y2)=1; lenaImage(x2,y1:y2)=1;
lenaImage(x1:x2,y1)=1; lenaImage(x1:x2,y2)=1;
imagesc(lenaImage) %show the image
axis equal %correct the aspect ratio
%Some calculations to practice:
X=octave_dct2(x); %calculate forward DCT
x2=octave_idct2(X); %calculate inverse DCT to check
numericError = max(x(:)-x2(:)) %any numeric error?
Ah=octave_dctmtx(8); %now calculate in an alternative way
X2 = Ah*x*transpose(Ah);
%X2 = transpose(Ah)*x*Ah; %Note: this would be wrong!
numericError2 = max(X(:)-X2(:)) %any numeric error?
```

Similar to the 1-D ECG coding example in Application 2.4, few DCT coefficients can be enough to represent an image. Study the code MatlabOctaveFunctions/ak_dctdemo.m using DCT to represent different images,[13] including ones with text. Even better results can be obtained with the wavelet transform, which is used in the JPEG 2000 image coding standard. Write an encoder and decoder that use a transform to compress and decompress an image file. □

Application 2.6. Orthogonality of finite duration discrete-time sinusoids. It is important to evaluate in what conditions a pair of discrete-time sinusoids are orthogonal considering a finite duration interval. This issue is discussed in the sequel.

Two discrete-time exponentials $e^{j\Omega_0 n}$ and $e^{j\Omega_1 n}$, $\Omega_0 \neq \Omega_1$, are always orthogonal when summed over a range $-\infty$ to ∞ (see Example 2.7), but this is not the case when the summation interval is finite. One way of inspecting what happens when the duration is finite is to reason as following: 1) keep in mind that the sum of the samples of a single sinusoid over a multiple of its fundamental period is zero. 2) Interpreting Example 2.9 in discrete-time allows to decompose the inner product for testing orthogonality into a sum of two sinusoids. 3) the inner product is mapped to summations of sinusoids with frequencies equal to $\Omega_0 + \Omega_1$ and $\Omega_0 - \Omega_1$ and the goal is to find an integer N that is a multiple of both fundamental periods.

One application of this orthogonality principle is the design of a binary transmission, where the two signals are distinguished by their frequencies f_0 and f_1. This scheme is called frequency shift keying (FSK). We are interested on determining the minimum symbol period that guarantees orthogonality between the two sinusoids. In the context of telecommunications, this minimum period will correspond to the highest symbol rate. The script MatlabOctaveBookExamples/ex_transforms_check_orthogonality.m can be used for studying this question and is repeated below. The sampling frequency is F_s and one should check different pairs of f_0 and f_1.

Listing 2.8: *MatlabOctaveBookExamples/ex_transforms_check_orthogonality.m*

```
if 1 %choose between two options
    f0 = 980;   %frequency of first sinusoid (in telecom, of bit 0)
    f1 = 1180;   %frequency of 2nd sinusoid (in telecom, of bit 1)
else %another example
    f0 = 1650; %1st sinusoid
    f1 = 1850; %2nd sinusoid
end
Fs=9600; %assumed sampling frequency
sumSinusoidFrequency = f0+f1;
subractSinusoidFrequency = f1-f0;
%reduce numbers to rational forms
[m,n]=rat(sumSinusoidFrequency/Fs)
[p,q]=rat(subractSinusoidFrequency/Fs)
Nmin = lcm(n,q) %find the least commom multiple
disp(['Minimum common period: ' num2str(Nmin)])
T_duration = Nmin / Fs; %minimum number of samples in seconds
```

[13] A popular repository is [url2usc].

```
%disp(['Maximum symbol rate: ' num2str(1/T_symbolDuration) ' bauds'])
%% Check if the product of the two sinusoids is really orthogonal
Tsum = 1 / sumSinusoidFrequency;
Tsub = 1 / subractSinusoidFrequency;
disp('Numbers below should be integers:')
T_duration/Tsum
T_duration/Tsub
Ts=1/Fs; %sampling interval
t = 0:Ts:T_duration-Ts; %generate (discrete) time
%use sin or cos with arbitrary phase
b0 = sin(2*pi*f0*t+4)';  %column vector
b1 = sin(2*pi*f1*t)';  %column vector
normalizedInnerProduct = sum(b0.*b1) / length(b0) %inner product
if (normalizedInnerProduct > 1e-13)
    error('not orthogonal! should not happen!');
end
```

Modify Listing 2.8 to use discrete-time sinusoids in radians, not Hz. Assume $N = 8$ and find a set of M sinusoids that are mutually orthogonal over an interval of eight consecutive samples. What was the maximum value of M that you could find? Now, consider the set of basis functions for a N-point DFT and evaluate their orthogonality. Going back to the question about M, can it be larger than N? Why? □

Application 2.7. Fourier series coefficients by inspection.

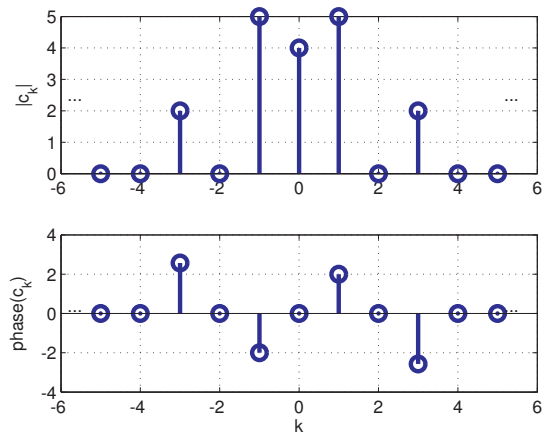


Figure 2.28: *Spectrum of* $x(t) = 4 + 10\cos(2\pi 50t + 2) + 4\sin(2\pi 150t - 1)$.

When the periodic signal $x(t)$ is composed by a sum of sines and cosines, it is convenient to obtain the Fourier series coefficients by inspection. For example, assume the signal $x(t) = 4 + 10\cos(2\pi 50t + 2) + 4\sin(2\pi 150t - 1)$. Using Euler's, one has

$$10\cos(2\pi 50t + 2) = 5e^{j2\pi 50t}e^{j2} + 5e^{-j2\pi 50t}e^{-j2}$$

and

$$\begin{aligned} 4\sin(2\pi 150t - 1) &= \frac{4}{2j}[e^{j2\pi 150t}e^{-j1} - e^{-j2\pi 50t}e^{j1}] \\ &= \frac{2}{e^{j\pi/2}}[e^{j2\pi 150t}e^{-j1} + e^{j\pi}e^{-j2\pi 50t}e^{j1}] \\ &= 2e^{j2\pi 150t}e^{-j(1+\pi/2)} + 2e^{-j2\pi 50t}e^{j(1+\pi/2)}. \end{aligned}$$

Hence, the only non-zero coefficients are $c_0 = 4$, $c_1 = 5e^{j2}$, $c_3 = 2e^{-j(1+\pi/2)}$, $c_{-1} = 5e^{-j2}$, $c_{-3} = 2e^{j(1+\pi/2)}$, with $1 + \pi/2 \approx 2.57$ as indicated in Figure 2.28, which shows the magnitude and phase graphs of the Fourier series coefficients for representing $x(t)$. □

Application 2.8. Bilateral and unilateral spectrum representations.

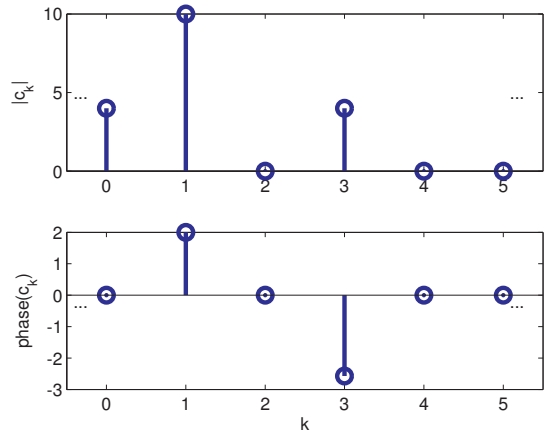


Figure 2.29: *Unilateral spectrum of (real) signal* $x(t) = 4 + 10\cos(2\pi 50t + 2) + 4\sin(2\pi 150t - 1)$.

The spectrum of Figure 2.28 is called *bilateral* because the negative frequencies are explicitly represented. In this case, a sinusoid of amplitude A is represented by a pair of coefficients with magnitudes $A/2$ each. An alternative representation, valid only for real signals, is the *unilateral* spectrum, where only $c'_k, k \geq 0$ are shown. In this case, $c'_0 = c_0$ and $c'_k = 2c_k, k > 0$ as illustrated in Figure 2.29. This text emphasizes the bilateral representation because it is more general and capable of representing the spectrum of a complex-valued signal $x(t)$. □

Application 2.9. DTFS coefficients by inspection. Assume $x[n] = A\cos(\frac{\pi}{6}n + \pi/3)$, which has a period of $N = 12$ samples. Instead of calculating via Eq. (2.25), one can rewrite the signal as a sum of complex exponentials:

$$x[n] = \frac{A}{2}\left[e^{j(\frac{\pi}{6}n+\pi/3)} + e^{-j(\frac{\pi}{6}n+\pi/3)}\right] = \frac{A}{2}e^{j\frac{\pi}{6}n}e^{j\pi/3} + \frac{A}{2}e^{-j\frac{\pi}{6}n}e^{-j\pi/3}.$$

With $N = 12$, the DTFS synthesis equation is:

$$x[n] = \sum_{k=\langle 12\rangle} X[k] e^{jk\frac{2\pi}{12}n},$$

where the range $k = \langle 12\rangle$ is chosen to be: $k = -6, -5, \ldots, 0, 1, \ldots, 5$, which leads to

$$x[n] = \sum_{k=-6}^{5} X[k] e^{jk\frac{\pi}{6}n}.$$

By inspection, $x[n]$ can be represented by two coefficients: $X[-1] = \frac{A}{2}e^{-j\pi/3}$ and $X[1] = \frac{A}{2}e^{j\pi/3}$, while all other coefficients in the range $k \in [-6, 5], k \neq \pm 1$ are zero.

If the range $k = \langle 12\rangle$ were $k = 0, 1, \ldots, 11$, it would be necessary to recall that $X[k] = X[k + N]$. Hence, $X[11] = X[1]$ and the spectrum of $x[n]$ could be represented by $X[1] = \frac{A}{2}e^{j\pi/3}$ and $X[11] = \frac{A}{2}e^{-j\pi/3}$ with $X[k] = 0$ for $k = 0$ and $1 < k < 11$. An alternative view of this periodicity is to sum $2\pi n$ to the angle $e^{-j\frac{\pi}{6}n}$, which leads to $e^{j2\pi n}e^{-j\frac{\pi}{6}n} = e^{j\frac{11\pi}{6}n}$ and allows to write

$$x[n] = \frac{A}{2}e^{j\frac{\pi}{6}n}e^{j\pi/3} + \frac{A}{2}e^{j\frac{11\pi}{6}n}e^{-j\pi/3}.$$

□

Application 2.10. Better visualizing the spectrum: Using fftshift to reorganize the spectrum and masking numerical errors. Two improvements of the procedure adopted to obtain Figure 2.8 are typically used. The first one is with respect to the range $k = \langle N\rangle$. Figure 2.8 reminds that the definition of an N-point DFT assumes $k = 0, \ldots, N - 1$. Sometimes it is more convenient to work with ranges that use "negative" frequencies. Typical ranges are $k = -N/2, -N/2 + 1, \ldots, 0, 1, N/2 - 1$ when N is even and $-(N - 1)/2, -(N - 1)/2 + 1, \ldots, 0, 1, (N - 1)/2$ when N is odd. In these cases, to represent negative frequencies (or angles in discrete-time processing) the command fftshift can be used. For example, fftshift([0 1 2 3]) returns [2 3 0 1], while fftshift([0 1 2 3 4]) returns [3 4 0 1 2].

The second one aims at eliminating spurious values. Note that, due to numerical errors, the elements of X that should be zero, have small values such as -1.4e-15 - j 3.3e-16, which leads to random angles for $k \neq 1, 11$ in Figure 2.8. It is a good practice to eliminate these random angles based on a threshold for the magnitude. This post-processing step can be done with a command such as X(abs(X) < 1e-12)=0, and is adopted in the Matlab/Octave fft routine.

Figure 2.30 benefits from these two improvements and was obtained using Listing 2.9.

Listing 2.9: *MatlabOctaveCodeSnippets/snip_transforms_fftshift.m*

```
k=-N/2:N/2-1; %range with negative k (assume N is even)
X(abs(X)<1e-12)=0;%discard small values (numerical errors)
X=fftshift(X); %rearrange to represent negative freqs.
```

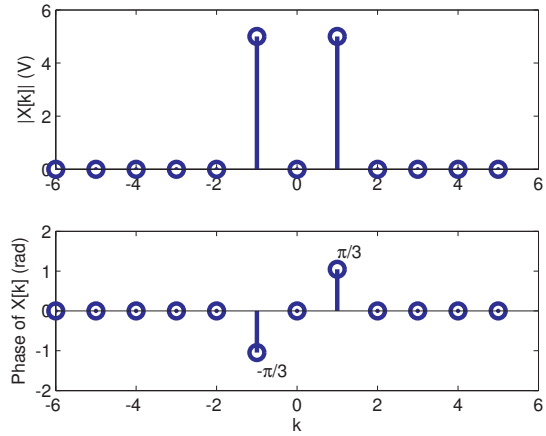


Figure 2.30: *Alternative representation of the DTFS / DFT of* $x[n] = 10\cos(\frac{\pi}{6}n + \pi/3)$ *using* fftshift. *Compare with Figure 2.8 and note that the current plots clearly indicates that when* $x[n]$ *is real, the magnitude and phase are even and odd functions, respectively.*

```
subplot(211);stem(k,abs(X));subplot(212);stem(k,angle(X));
```

Note that fftshift does not calculate the FFT but simply reorders the elements in a vector. □

Application 2.11. DTFS analysis of single cosines. Figure 2.31 depicts five cosines $x_i[n] = 10\cos(\Omega_i n)$ with angular frequencies $\Omega_i = i2\pi/32$ rad, where $i = 0, 1, 2, 16, 31$, and their respective DTFS spectrum calculated with a DFT of $N = 32$ points and normalized by N.

The first $x_0[n] = 10$ is a DC signal ($\Omega_0 = 0$), the second is $x_1[n] = 10\cos(2\pi/32n)$, where $\Omega_1 = \frac{2\pi}{32}$ is the fundamental frequency in this DTFS analysis (given that $N = 32$) and the other three signals are cosines with the following harmonic frequencies: $\Omega_2 = 2\Omega_1$, $\Omega_{16} = 16\Omega_1$ and $\Omega_{31} = 31\Omega_1$ rad. Because the cosines are even functions, the imaginary part of their DTFS is zero.

Note in Figure 2.31 that the frequency increases up to the $N/2$-th harmonic $\Omega_{16} = 16\Omega_0 = \pi$. When N is even, the coefficient of $k = N/2$ corresponds to $\Omega = \frac{2\pi}{N}k|_{k=N/2} = \pi$. It is important to note that the highest "frequency" of a discrete-time signal is always $\Omega = \pi$ rad.

Another aspect of Figure 2.31 is that for $x_0[n]$ and $x_{16}[n]$, corresponding to the angles $\Omega = 0$ and $\Omega = \pi$, the spectra have only one non-zero coefficient $X[k]$, while the other three signals are represented by two coefficients. Figure 2.3, for $N = 4$ and $N = 6$ can be invoked to help interpreting why the coefficients for angles different than $\Omega = 0$ and $\Omega = \pi$ occur in pairs with Hermitian symmetry (i. e., $X[k] = X^*[-k]$ when $x[n]$ is real). The basis functions for $\Omega = 0$ and $\Omega = \pi$ are real signals while the other angles lead to complex-valued signals. In the analysis of a real signal $x[n]$, using the k-th complex basis requires also using the $-k$-th basis, such that their imaginary parts can cancel in the synthesis of a real $x[n]$. □

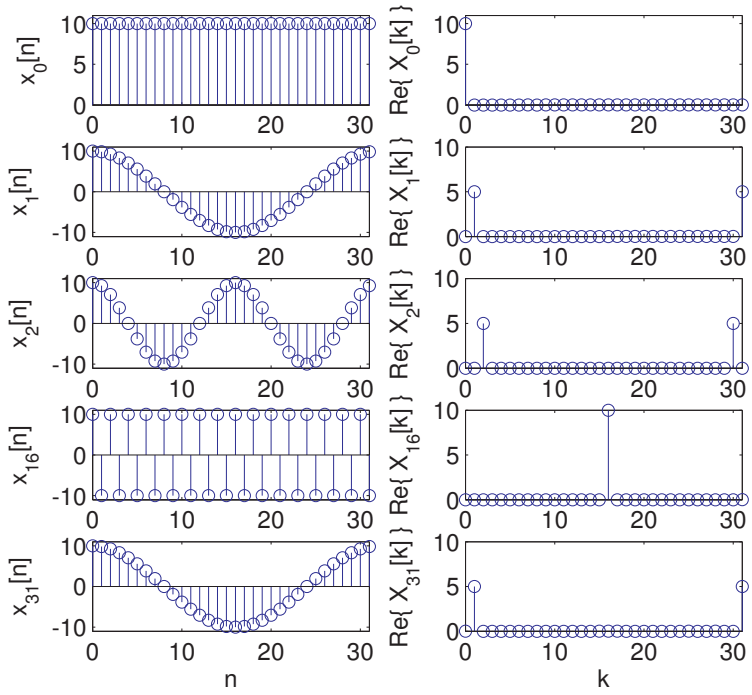


Figure 2.31: *Five cosine signals* $x_i[n] = 10\cos(\Omega_i n)$ *with frequencies* $\Omega_i = 0,\ 2\pi/32,\ 4\pi/32,\ \pi,\ 31\pi/16$ *for* $i = 0, 1, 2, 16, 32$*, and the real part of their DTFS using* $N = 32$ *points.*

Application 2.12. DTFS analysis of a sum of sines and cosines. This example aims at illustrating how the cosines and sines are mapped in the DTFS, especially when two of them have the same frequency. Figure 2.32 shows the spectrum of a more complex example obtained with Listing 2.10.

Listing 2.10: *MatlabOctaveCodeSnippets/snip_transforms_DTFS_sinusoid.m*

```
N=32; %number of DFT-points
n=0:N-1; %abscissa to generate signal below
x=2+3*cos(2*pi*6/32*n)+8*sin(2*pi*12/32*n)-...
    + 4*cos(2*pi*7/32*n)+ 6*sin(2*pi*7/32*n);
X=fft(x)/N; %calculate DTFS spectrum via DFT
X(abs(X)<1e-12)=0; %mask numerical errors
```

Note in Figure 2.32 that the coefficient $X[0] = 2$ is due to the DC level, $X[6] = X[-6] = 1.5$ are due to the cosine $3\cos(6(2\pi/32)n)$, $X[12] = -4j$ and $X[-12] = 4j$ are due to $8\sin(12(2\pi/32)n)$, $X[7] = -2 - 3j$ and $X[-7] = -2 + 3j$ are due to the two parcels of frequency $7(2\pi/32)$ with the $-4\cos(7(2\pi/32)n)$ being represented by the real part (-2) and $6\sin(7(2\pi/32)n)$ represented by the imaginary part $(3j)$. □

Application 2.13. Spurious frequency components: When the number N of DFT points is not a multiple of the signal period. Figure 2.33 illustrates the DTFS spectrum of a signal $x[n] = 4\cos((2\pi/6)n)$ with period of 6 samples obtained

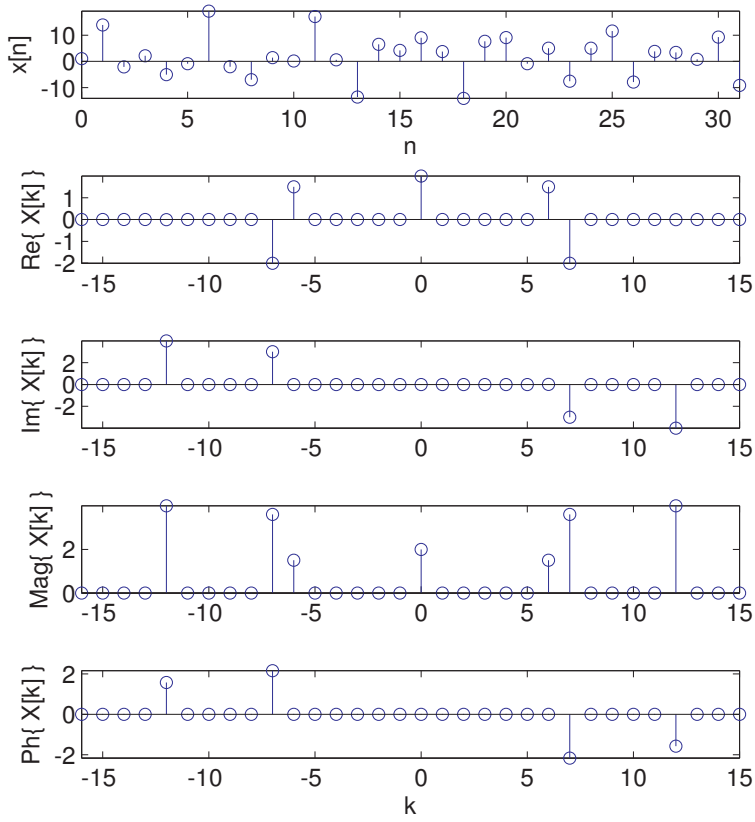


Figure 2.32: *Analysis with DFT of 32 points of $x[n]$ composed by three sinusoids and a DC level. From top to bottom: $x[n]$, real, imaginary, magnitude and phase of $X[k]$.*

via a 16-points DFT. Note that because 16 is not a multiple of 6, the spectrum has many non-zero spurious components. The coefficients $X[3] = 0.99 - 1.48j$ and $X[-3] = 0.99 + 1.48j$, both with magnitude 1.78, are the closest to representing the angle $\pi/3 \approx 1.05$ rad. In fact, when using $N = 16$ points to divide 2π (see Figure 2.3), the angle increment is

$$\Delta\Omega = \frac{2\pi}{N}, \tag{2.50}$$

which in this case is $\Delta\Omega = \pi/8 \approx 0.39$, and the DFT / DTFS can deal only with the angles $[0, 0.39, 0.78, 1.18, 1.57, \ldots, 5.89]$ rad. This explains why $X[3]$ and $X[-3]$ are concentrating most of the power $P = 4^2/2$. The other coefficients absorb the remaining power that "leaks" due to the imperfect match of the cosine frequency $\pi/3$ with the discrete grid of 16 points imposed by the DFT / DFTS.

The creation of spurious components is associated to *leakage* and the *picket-fence effect*, which are discussed in Section **??**. This phenomenon is very common when using the FFT to analyze signals that are non-periodic, have infinite duration, etc. In the current case, the spurious components could be avoided by choosing an appropriate number of points for the DFT / DTFS (a multiple of 6). In practice, spurious

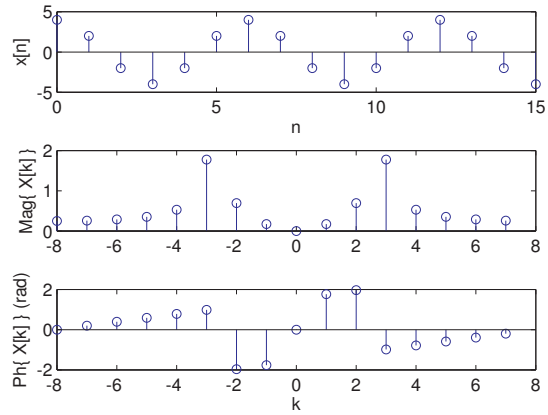


Figure 2.33: *Spectrum of a signal* $x[n] = 4\cos((2\pi/6)n)$ *with period of 6 samples obtained with a 16-points DFT, which created spurious components.*

components due to FFT usage occur most of the times. Even when the original signal is periodic, say $x(t)$ with period T, it is typically not guaranteed that NT_s is a multiple of T, where N is the number of DFT points and T_s the sampling period.

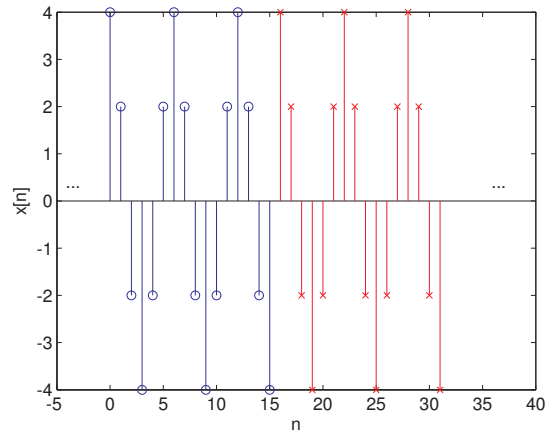


Figure 2.34: *Explicitly repeating the block of* N *cosine samples from Figure 2.33 to indicate that spurious components are a manifest of the lack of a perfect cosine in time-domain.*

Figure 2.34 shows part of the signal obtained by repeating the segment of 16 samples depicted in Figure 2.33. This represents the signal "assumed" by the DFT / DTFS. Spurious components appear because, in spite of $x[n]$ having a period of $N = 6$, an abrupt transition (from -4 to 4) occurs in the boundary of the segments of 16 samples. In general, the periodic extension (that is assumed by the FFT when approximating a DTFS) of a signal $x[n]$ not commensurate[14] with its period, leads to discontinuities at the boundaries of the replicas of $x[n]$. □

[14] See Example 1.11 for a discussion of commensurate frequencies.

Application 2.14. DTFS for periodic discrete-time pulses. Assume a periodic train of pulses $x[n]$ with period N and one period described as $x[n] = 1$ from $n = 0$ to $N_1 - 1$ and $x[n] = 0$ from N_1 to $N - 1$. Its DTFS coefficients are given by

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk} = \frac{1}{N}\sum_{n=0}^{N_1-1} e^{-j\frac{2\pi}{N}nk} = \frac{1}{N}\sum_{n=0}^{N_1-1}\left(e^{-j\frac{2\pi}{N}k}\right)^n,$$

which is the sum of N_1 terms of a geometric series with ratio $r = e^{-j\frac{2\pi}{N}k}$. Applying Eq. (A.17) leads to

$$X[k] = \frac{1}{N}\left(\frac{1 - e^{-j\frac{2\pi}{N}N_1 k}}{1 - e^{-j\frac{2\pi}{N}k}}\right).$$

This expression can be simplified by applying Eq. (A.12) to both numerator and denominator:

$$X[k] = \frac{1}{N}\frac{\sin(kN_1\pi/N)}{\sin(k\pi/N)}e^{-j\frac{k\pi}{N}(N_1-1)}, \tag{2.51}$$

for $k = 0, \ldots, N-1$ and then periodically repeated in k. For $k = 0$ the expression is undetermined (because of the 0/0 fraction) and L'Hospital's rule leads to $X[0] = N_1/N$.

Listing 2.11 obtains the DTFS of the periodic pulses in two distinct ways.

Listing 2.11: *MatlabOctaveCodeSnippets/snip_transforms_DTFS_pulses.m*

```
N=10; N1=5; k=0:N-1; %define durations and k
Xk=(1/N)*(sin(k*N1*pi/N)./sin(k*pi/N)) .* ...
    exp(-j*k*pi/N*(N1-1)); %obtain DTFS directly
Xk(1)=N1/N; %eliminate the NaN (not a number) in Xk(1)
%second alternative, via DFT. Generate x[n]:
xn=[ones(1,N1) zeros(1,N-N1)] %single period of x[n]
Xk2=fft(xn)/N %DTFS via DFT, Xk2 is equal to Xk
```

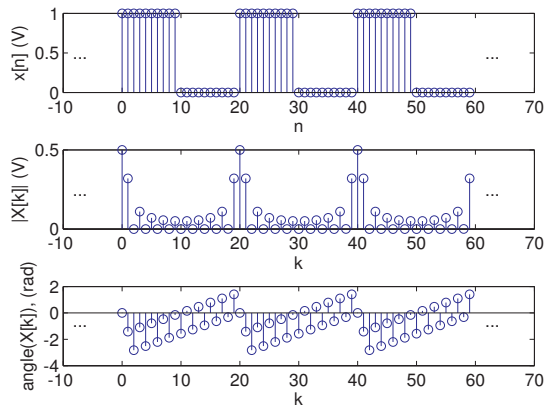
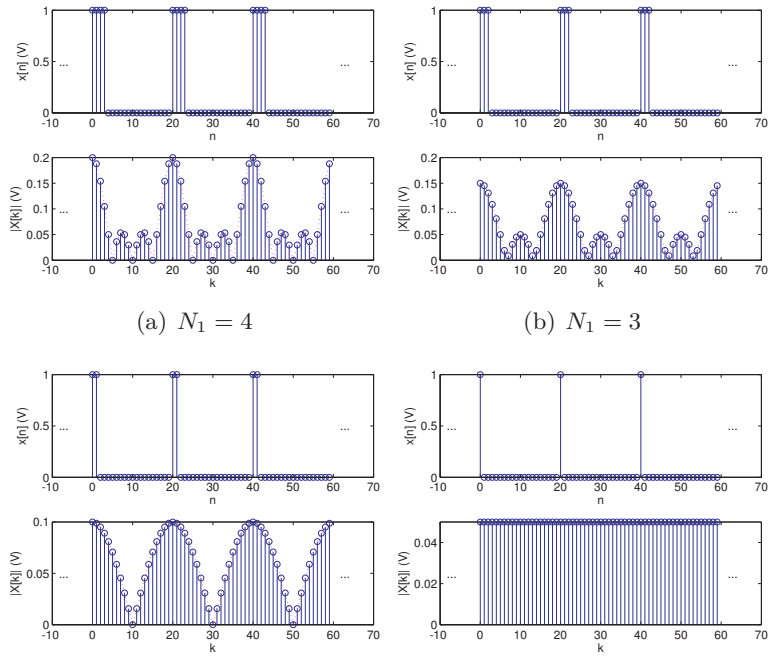


Figure 2.35: *Three periods of each signal: pulse train $x[n]$ with $N = 10$ and $N_1 = 5$ and amplitude assumed to be in Volts (a), the magnitude (b) and phase (c) of its DTFS.*

Figure 2.35 illustrates the result using $N = 20$ and $N_1 = 10$. Note that $|X[k]|$ has the behavior of a sinc function, as expected. The DTFS phase in Figure 2.35 c) is the combined effect of the linear phase $e^{-j\frac{k\pi}{N}(N_1-1)}$ and the sign of $\sin(kN_1\pi/N)/\sin(k\pi/N)$ in Eq. (2.51). This sign is responsible for Figure 2.35 c) not being a linear function of frequency.



(a) $N_1 = 4$ (b) $N_1 = 3$

(c) $N_1 = 2$ (d) $N_1 = 1$

Figure 2.36: *Behavior when N_1 of Figure 2.35 is decreased from $N_1 = 4$ to 1.*

Figure 2.36 shows the behavior of the DTFS of $x[n]$ when N_1 of Figure 2.35 is decreased from $N_1 = 4$ to 1 Note the duality between time and frequency: as the pulse duty cycle gets smaller (smaller N_1), the spectrum gets wider and with smaller maximum values. When $N_1 = 1$, then $|X[k]| = N_1/N = 0.05$. Also, in this case, N_1 dictates the number of nulls in the spectrum, while N controls how the unity circle $e^{j\Omega}$ is sampled by the DFT. □

Application 2.15. Calculating the DTFT of a pulse via the DFT. Following steps similar to the ones that led to Eq. (2.51), the DTFT pair of a pulse is given by

$$x[n] = \begin{cases} 1, & 0 \le n \le N_1 - 1 \\ 0, & \text{otherwise} \end{cases} \quad \Leftrightarrow \quad \frac{\sin(\Omega N_1/2)}{\sin(\Omega/2)} e^{-j\Omega(N_1-1)/2} \tag{2.52}$$

Listing 2.12 illustrates how to obtain samples of the DTFT for a pulse with $N_1 = 5$ non-zero samples using a DFT of $N = 20$ points.

Listing 2.12: *MatlabOctaveCodeSnippets/snip_transforms_DTFT_pulse.m*

```
N=20; %DFT size
N1=5; %num. non-zero samples in the (aperiodic) pulse
x=[ones(1,N1) zeros(1,N-N1)]; %signal x[n]
N2=512; %num. samples of the DTFT theoretical expression
k=0:N2-1; %indices of freq. components
wk = k*2*pi/N2; %angular frequencies
Xk_fft = fft(x); %use N-point DFT to sample DTFT
Xk_theory=(sin(wk*N1/2)./sin(wk/2)).*exp(-j*wk*(N1-1)/2);
Xk_theory(1) = N1; %take care of 0/0 (NaN) at k=0
subplot(211); stem(2*pi*(0:N-1)/N,abs(Xk_fft));
hold on, plot(wk,abs(Xk_theory),'r');
subplot(212); stem(2*pi*(0:N-1)/N,angle(Xk_fft));
hold on, plot(wk,angle(Xk_theory),'r');
```

The code plots the DTFT superimposed to the DFT result, as illustrated in Figure 2.37.

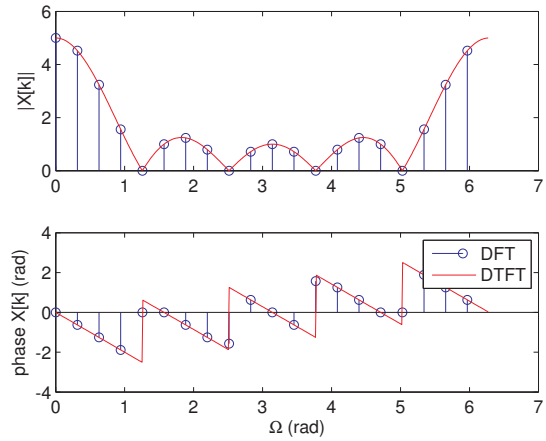


Figure 2.37: *DTFT of an aperiodic pulse with $N_1 = 5$ non-zero samples and DTFT estimates obtained via a DFT of $N = 20$ points.*

As stated by Eq. (2.30), Figure 2.37 pictorially illustrates that the DFT values at the discrete frequencies $\Omega_k = k(2\pi/N)$ coincide with the DTFT of the aperiodic pulse $x[n]$ calculated at these frequencies using Eq. (2.52).

The DTFT can be calculated in a grid of frequencies with arbitrary resolution by increasing the number N of DFT points. For example, using $N = 256$ leads to the spectrum in Figure 2.38. Note that if x has less than N samples, the command fft(x,N) conveniently extends its duration with zero-padding. □

Application 2.16. Calculating the DTFT using Matlab/Octave's freqz. When N is relatively large, it is more convenient to use plot instead of stem (as in Figure 1.16), in spite of $X[k]$ being discrete. This is also convenient when $X[k]$ is used to represent values of $X(e^{j\Omega})$ as in this example. Matlab/Octave has the command

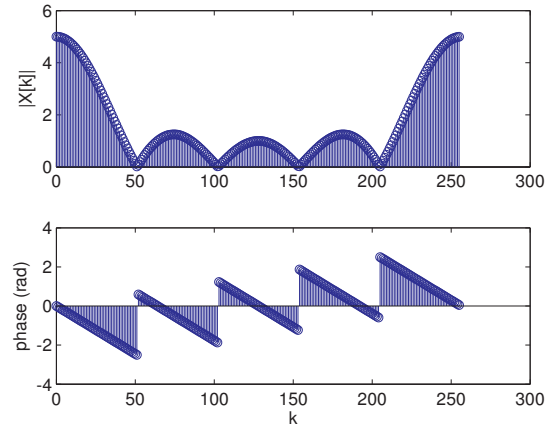


Figure 2.38: *A version of Figure 2.37 using a DFT of $N = 256$ points.*

freqz that uses the DFT to calculate the DTFT at a grid of discrete frequencies. Assuming the same vector x of the previous code snippet, the command freqz(x) generates Figure 2.39.

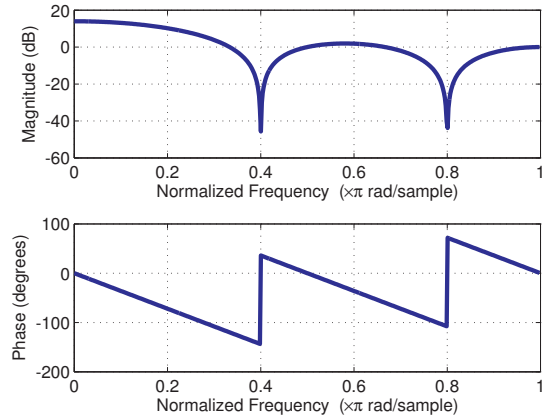


Figure 2.39: *A version of Figure 2.37 using freqz with 512 points representing only the positive part of the spectrum.*

To better understand how freqz works, Listing 2.13 mimics freqz and was used to obtain Figure 2.40.

Listing 2.13: *MatlabOctaveCodeSnippets/snip_transforms_freqz.m*

```
N1=5; %num. non-zero samples in the (aperiodic) pulse
x=[ones(1,N1) zeros(1,2)]; %signal x[n]
halfN=512; N=2*halfN; %half of the DFT dimension and N
k=0:N-1; %indices of freq. components
wk = k*2*pi/N; %angular frequencies
Xk_fft=fft(x,N);%use 2N-DFT to get N positive freqs.
Xk_fft=Xk_fft(1:halfN);%discard negative freqs.
```

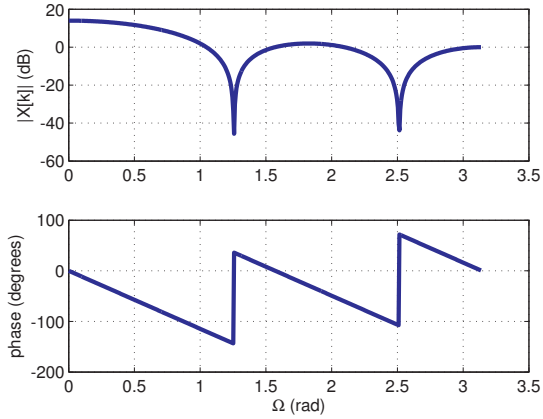


Figure 2.40: *Reproducing the graphs generated by* freqz *in Figure 2.39.*

```
wk=(0:halfN-1)*(2*pi/N); %positive freq. grid
subplot(211); plot(wk,20*log10(abs(Xk_fft))); %in dB
subplot(212); plot(wk,angle(Xk_fft)*180/pi); %in degrees
```

It can be seen that Figure 2.40 is very similar to Figure 2.39. The only distinction is that, by default, Matlab/Octave represents the abscissa normalizing Ω by π, such that the maximum frequency is not π but 1. It is interesting to note that Matlab/Octave and GNU Radio, for example, indicate Ω in rad/sample, while this and other publications assume Ω is given in radians (n is dimensionless). □

2.11 Comments and Further Reading

Some authors recognize the advantages of presenting concepts of digital signal processing before their analog counterpart. This chapter adopts this approach. For example, it presents block transforms before the continuous time Fourier transform.

A commonly adopted notation is to call *inverse* transformation the matrix multiplication $\mathbf{x} = \mathbf{AX}$, while the *forward* (or direct) transformation is denoted as $\mathbf{X} = \mathbf{A}^{-1}\mathbf{x}$., such that the basis are the columns of $\mathbf{A}$ [Mal92]. Experience proved that this notation confuses beginners and it was not adopted here.

Most textbooks introduce Z and Laplace transforms using the fact that their basis functions are eigenfunctions of linear and time-invariant systems. In this chapter, both transforms were presented as extensions of their Fourier counterparts.

All the four Fourier representations are interrelated. There are many interesting properties among the representations, which are explored in, e. g., [OS09].

With respect to block transforms, the attention is restricted to the ones represented by square matrices. Also, the emphasis is on their use and interpretation. Fast algorithms are out of the scope. A more advanced treatment of transforms, including lapped transforms (that correspond to non-square matrices) and fast algorithms can be found in [Mal92].

There are four DCT transforms [RY90]. It was assumed the DCT-II, the most popular for coding.

It is assumed by default the bilateral Laplace and Z transforms. The unilateral versions are useful for solving differential and difference equations, which is a task not emphasized in this text.

Unitary matrices are sometimes called orthogonal matrices (instead of orthonormal), which is confusing. For vectors, the jargon is more consistent.

A nice geometrical explanation about linearly independent, orthogonal, and uncorrelated variables is provided in [RNT84].

2.12 Review Exercises

2.1. Given two vectors $[4, 3]$ and $[-5, 2]$, find their norms, inner product and angle between them.

2.2. Given two vectors $\mathbf{x} = [4, 0]$ and $\mathbf{y} = [-5, 2]$, find the projections $\mathbf{p}_{yx}$ and $\mathbf{p}_{xy}$ and the respective error vectors $\mathbf{e}_{yx}$ and $\mathbf{e}_{xy}$.

2.3. Is the matrix $\mathbf{A} = \begin{bmatrix} 1 & -5 \\ 3 & 4 \end{bmatrix}$ unitary? Design a unitary matrix $\mathbf{B}$ that is similar to $\mathbf{A}$ in the sense that its first basis (column) is a vector in the same direction as $[1,\ 3]^T$.

2.4. Assume a 2-d vector space has non orthogonal basis given by $\bar{i} = [2, 1]$ and $\bar{j} = [0, 3]$. Find the coefficients α and β that allow to represent the vector $\mathbf{x} = [4, 1]$ as the linear combination $\mathbf{x} = \alpha\bar{i} + \beta\bar{j}$.

2.5. Assuming a 2-d vector space with orthonormal basis vectors $\bar{i}$ and $\bar{j}$, prove that inner products can be used to find the coefficients α and β that allow to represent a vector $\mathbf{x} = [x_1, x_2]$ as the linear combination $\mathbf{x} = \alpha\bar{i} + \beta\bar{j}$.

2.6. Find the inner product between the signals: a) $u(t)$ and $e^{-0.9t}$, b) $\cos(0.5\pi n)$ and $\delta[n]$, c) $u[-n-1]$ and $u[n]$.

2.13 Exercises

2.1. A segment of an ECG signal $x[n]$ with six samples $[-1, 0.6, 0.5, 1, 2.5, 2]$ must be encoded using a 3-points DCT (i. e., using two frames with three samples each). Find the DCT coefficients of these two vectors. Then, reconstruct an approximation of the two vectors assuming that only the first coefficient can be used while the other two are discarded (assumed to be zero). Confirm Eq. (2.19) calculating the norms of both error vectors $\mathbf{e}_t$ and $\mathbf{e}_f$.

2.2. A segment of an ECG signal with four samples $[0.5, 1, 2.5, 2]$ must be encoded using a 2-points DCT (i. e., using two frames with two samples each). The coefficients at the DCT domain must be quantized with 4 bits per frame, with the first coefficient (DC) being quantized with 3 bits and the other one with 1 bit. Both quantizers have a step of $\Delta = 0.5$ and their first output level is 0.25. a) What is the decoded signal

(after coding and decoding)? b) What is the mean squared-error? c) What is the SNR in dB? d) What is the bit rate in bps assuming $F_s = 150$ Hz? e) What is the bit rate and SNR if the second coefficient is discarded (no bits are allocated for it and its value is assumed to be zero during decoding)?

2.3. Consider that a block of pixels $\mathbf{C} = \begin{bmatrix} 120 & 140 & 139 \\ 124 & 220 & 120 \\ 114 & 130 & 122 \end{bmatrix}$ was extracted from an image. Calculate the bi-dimensional DCT and DFT first processing each row and organizing the result of this intermediate step as a matrix $\mathbf{T}$, then apply the transforms to each column of $\mathbf{T}$ to generate the final matrix. Follow this procedure to obtain a matrix with the DCT coefficients and then repeat it to obtain the DFT coefficients.

2.4. Design a DCT-based image coding system that is capable of handling 24 bits/pixel color images and 8 bits/pixel black and white (B&W, also called gray) images, all with a size of 512×512 pixels. The system should operate using: a) 3 bits/pixel, b) 1 bit/pixel or c) 0.25 bit/pixel. For the three cases, provide the average peak SNR (PSNR), i. e., the SNR when assuming the "signal" in the numerator is an image with all pixels at the peak value of 255 and the denominator is the mean squared-error. The test sequence should be the colorful and B&W versions of the images Lenna, Peppers, Baboon, Splash and Tiffany, available from [url2us2]. Do not download B&W versions of the five mentioned images, but make the conversion from color to B&W yourself and describe the conversion method you used. Hence, the test set will have 5 images. You can use as many images you want to compose your training set (distinct from the test set images). Provide the source code and a short report of your results that includes the average PSNR.

2.5. Use the Gram-Schmidt procedure to find an orthonormal basis to represent the vectors in the set $\mathcal{C} = \{[001], [010], [008], [050], [412], [003]\}$. Show also that any vector in $\mathcal{C}$ can be written as a linear combination of the obtained basis vectors.

2.6. PCA assumes the signal is zero mean. This exercise uses this fact to exemplify the importance of following an assumed model. KLT (or PCA) is the optimal linear transform for coding but it does assume a model. Design a coding system for the data `x` below.

Listing 2.14: *MatlabOctaveCodeSnippets/snip_transforms_KLT.m*

```
N=100; %number of vectors
K=4;   %vector dimension
line=1:N*K; %straight line
noisePower=2; %noise power in Watts
temp=transpose(reshape(line,K,N)); %block signal
x=temp + sqrt(noisePower) * randn(size(temp)); %add (AWGN) noise
z=transpose(x); plot(z(:)) %prepare for plot
```

Note that the KLT does not have info about the mean. Therefore, in practice, one needs to keep the mean and, eventually, normalize the variances to be unitary (this

normalization is called in statistics to "z-transform" the data, but this term should not be confused with the Z transform).

2.7. Generate $N = 10,000$ samples from a bidimensional Gaussian $f_{\mathbf{X}_1,\mathbf{X}_2}(x_1, x_2)$ with mean $\hat{\boldsymbol{\mu}} = (30,\ -20)^T$ and covariance matrix $\mathbf{C} = \begin{bmatrix} 10 & 0.99 \\ 0.99 & 40 \end{bmatrix}$. Obtain two transforms: $\mathbf{A}_g$ using Gram-Schmidt orthonormalization and $\mathbf{A}_p$ using PCA. You can remove means if useful. Using each transform, project the data into the new space and compute the variance of the coefficients in each dimension. Which transform you expect to lead to smaller variances of the transformed vectors?

2.8. Following a scheme similar to the one presented in Application 2.4 - ECG Coding, compare the performance of KLT and DCT using a test set of at least 50 ECT records from the MIT-BIH Arrhythmia Database. Note that for performing any preparation / training you should use a training set, disjoint of the test set. Otherwise your results can be biased by the fact that you already knew what should be coded.

2.9. Calculate "manually" the DFT of the sequence $x[n] = [2, 0, 2, 0]$. Then use the `fft` function in Matlab/Octave and compare the results.

2.10. What are the forward $\mathbf{A}$ and inverse $\mathbf{A}^H$ matrices for a 3-points orthonormal DFT?

2.11. Consider that $s[n]$ was obtained by sampling with $F_s = 100$ kHz an analog and periodic signal $s(t)$. The sampling theorem was obeyed. The 8-DFT of $s[n]$ was calculated in Matlab/Octave with `fft(s)/N` and is shown in Figure 2.41. The value N was chosen to be an exact multiple of the fundamental period of $s(t)$ such that the problem of spurious components illustrated in Figure 2.34 did not occur. a) What is the frequency of each component of $s(t)$? What is the resolution Δf in Hz of this DFT? c) Find a reasonable expression for $s(t)$.

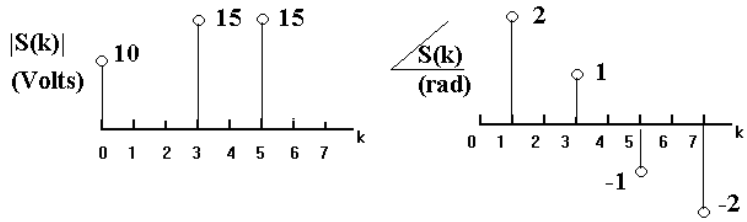


Figure 2.41: *DFT with $N = 8$ points of a signal sampled at $F_s = 100$ kHz.*

2.12. Given that $\mathbf{X} = [8, 0, 0, 0]$ was obtained with a 4-points orthonormal DFT, find its inverse $\mathbf{x}$. For the same spectrum, assuming that $F_s = 10$ Hz, to which frequencies correspond the elements of $\mathbf{X}$?

2.13. Consider that a periodic $x(t)$ was sampled with $F_s = 20$ Hz, obeying the sampling theorem. A 128-points FFT was calculated using Matlab/Octave and the output $\mathbf{X}$

was normalized by $N = 128$. The only non-zero coefficients were $k = 0, 3$ and 125, corresponding to the elements of $\mathbf{X}[1] = 12$, $\mathbf{X}[4] = 3 + j4$ and $\mathbf{X}[126] = 3 - j4$, respectively. a) What is the frequency in Hz of the component with largest power of $x(t)$? b) what is the average (DC) value of $x(t)$? c) what is the spacing in Hz between neighboring elements of $\mathbf{X}$? d) what is the maximum frequency found in $x(t)$?

2.14. Using the orthogonality condition of the basis functions derive the pair of Fourier series equations.

2.15. What are the Fourier series coefficients for the signal $x(t) = 2 + 3\cos(2\pi 100t) + 8\sin(2\pi 300t + \pi) + \cos(2\pi 50t - \pi)$?

2.16. The bilateral Fourier series of a signal $x(t)$ with period $T_0 = 0.01$ s has non-zero coefficients $c_0 = 3$, $c_1 = 2e^{-j4}$, $c_4 = 5$, $c_{-1} = 2e^{j4}$ and $c_{-4} = 5$. What is the expression for $x(t)$?

2.17. Prove the expressions: a) for the DTFT of $x[n] = 0.9^n u[n] + 0.8^n u[n]$ and b) for its Z transform, indicating the ROC.

2.18. Given that the DTFT $H(e^{j\Omega})$ of a signal in the range $]\pi, \pi]$ is 1 when $-\Omega_c \leq \Omega \leq \Omega_c$ and 0 otherwise (i. e., an ideal low-pass filter), find its inverse DTFT $h[n]$.

2.19. Prove that $X(e^{j\Omega}) = \sum_{k=-\infty}^{\infty} 2\pi\delta(\Omega - \Omega_0 + 2\pi k)$ is the DTFT of $x[n] = e^{j\Omega_0 n}$ and indicate when the impulse sifting property is used. Using the result, carefully draw the graph of the real part of the DTFT of $y[n] = 3 + 4e^{j\pi n}$ for the range $-3\pi \leq \Omega \leq 3\pi$.

2.20. Prove the cited properties of the Fourier tools for the case of the Fourier transform.

2.21. Prove the cited Fourier pairs for the case of the DTFT.

2.22. A right-sided signal has Laplace transform $X(s) = 5/(s^2 + 2s + 10)$, with ROC $\sigma > -1$. Calculate: a) $X(s)|_{s=2}$, b) the corresponding Fourier transform $X(\omega)$ if it exists and c) if $X(\omega)$ exists, $|X(\omega)|$ for $\omega = 3$ rad/s.

2.23. A signal $x(t)$ has Laplace transform $X(s) = (3s + 2)/((s - 1)(s - 3))$, with ROC $\sigma < 1$. Find $x(t)$.

2.24. A signal $x[n]$ has Z transform $X(z) = (5z^2 + 4z + 3)/((z - 0.7)(z - 0.8))$, with ROC $|z| > 0.8$. Find $x[n]$.

2.25. Find the Z transform and associated ROC of the signal $x[n] = -3^n u[-n - 1]$. Does this signal have a DTFT?

2.26. What is the Z transform $X(z)$ of the signal $x[n] = 4\delta[n] + 5\delta[n - 11]$? What are the values of $X(z)$ for $z = 0, -1$ and $z = 8e^{j\pi}$? If the DTFT of $x[n]$ exists, what are its values for $\Omega = 0, \pi$ and 2π rad?

3 Analog and Digital Systems

4 Spectral Estimation Techniques

5 Principles of Communications with Focus on AM

6 Baseband Digital Modulation with Focus on PAM

7 Passband Digital Modulation with Focus on QAM

8 Channels: Modeling, Simulating and Mitigating their Effects

9 Optimal Receivers and Probability of Error for AWGN

10 Multicarrier systems

Appendices

Appendix A Useful Mathematics

A.1 Even and odd functions

An even function $f(x)$ (also valid for a signal $x(t)$ or a sequence $x[n]$) has the property that $f(x) = f(-x)$, while an odd function has the property that $f(x) = -f(-x)$. Any function can be decomposed into an even $f_e(x)$ and odd $f_o(x)$ functions, such that $f(x) = f_e(x) + f_o(x)$. The two component functions can be obtained as

$$f_e(x) = \frac{f(x) + f(-x)}{2} \quad \text{and} \quad f_o(x) = \frac{f(x) - f(-x)}{2}.$$

For example, assume the unit step function $x(t) = u(t)$. Its even and odd parts are $x_e(t) = 0.5, \forall t$ and $x_o(t) = 0.5u(t) - 0.5u(-t)$, respectively.

A.2 Euler's formula

$$e^{jx} = \cos(x) + j\sin(x), \tag{A.1}$$

where x is given in radians. When $x = \pi$, it leads to the famous identity $e^{j\pi} = -1$. The value e^{jx} can be interpreted as a complex number with magnitude one and angle x rad. Hence, Eq. (A.1) represents the conversion of this complex number from the polar to the Cartesian form.

Using the fact that cosine and sine are even and odd functions, respectively, one can write $e^{-jx} = \cos(x) - j\sin(x)$ and using Eq. (A.1) obtain

$$\cos(x) = \frac{1}{2}(e^{jx} + e^{-jx}) \tag{A.2}$$

and

$$\sin(x) = \frac{1}{2j}(e^{jx} + e^{-jx}). \tag{A.3}$$

A.3 Trigonometry

$$\sin(a+b) = \sin a \cos b + \cos a \sin b. \tag{A.4}$$

$$\sin(2a) = 2\sin a \cos a. \tag{A.5}$$

$$\cos(a+b) = \cos a \cos b - \sin a \sin b. \tag{A.6}$$

$$\cos(2a) = \cos^2 a - \sin^2 a. \tag{A.7}$$

From A.7 and $\cos^2 a + \sin^2 a = 1$:

$$\cos^2 a = \frac{1}{2}(1 + \cos(2a)) \tag{A.8}$$

and

$$\sin^2 a = \frac{1}{2}(1 - \cos(2a)).$$

The following are some of the so-called *product to sum* identities:

$$\cos(a)\cos(b) = \frac{1}{2}[\cos(a-b) + \cos(a+b)]. \tag{A.9}$$

$$\sin(a)\sin(b) = \frac{1}{2}[\cos(a-b) - \cos(a+b)]. \tag{A.10}$$

$$\sin(a)\cos(b) = \frac{1}{2}[\sin(a+b) + \sin(a-b)], \tag{A.11}$$

where a is the argument of the sine in Eq. (A.11).

A.4 Manipulating complex exponentials

In Fourier and Z transforms it is common to encounter expressions such as $1 - e^{-j\theta}$. In some cases it is convenient to rewrite them as

$$1 - e^{-j\theta} = \frac{e^{j\theta/2}}{e^{j\theta/2}}(1 - e^{-j\theta}) = \frac{e^{j\theta/2} - e^{-j\theta/2}}{e^{j\theta/2}} = 2je^{-j\theta/2}\sin(\theta/2). \tag{A.12}$$

Similarly, one can write

$$1 + e^{-j\theta} = \frac{e^{j\theta/2}}{e^{j\theta/2}}(1 + e^{-j\theta}) = \frac{e^{j\theta/2} + e^{-j\theta/2}}{e^{j\theta/2}} = 2e^{-j\theta/2}\cos(\theta/2). \tag{A.13}$$

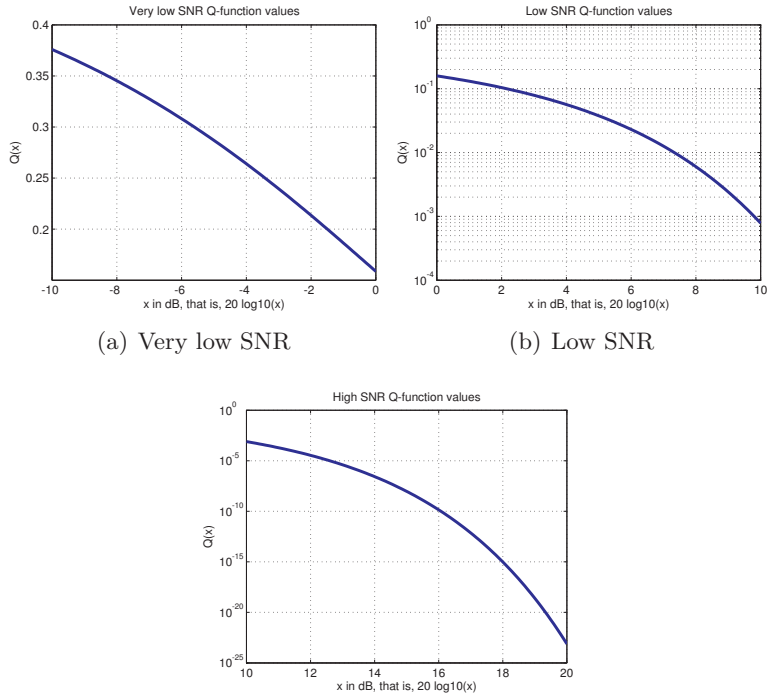
A.5 Q function

- One just needs to know $Q(x)$ for positive x because

$$Q(-x) = 1 - Q(x) \tag{A.14}$$

- When expressed in dB, it is used $20\log_{10}(x)$.
- $Q(x) = 0.5\,\mathrm{erfc}(x/\sqrt{2})$, where erfc is the complimentary error function.
- Matlab provides the qfunc in the comm toolbox. In case this toolbox is not available or using Octave, it is possible to use the erfc function as follows: `y = 0.5*erfc(x/sqrt(2))`. See `ak_qfunc.m` and `ak_qfuncinv.m`.

The Q values for three different ranges of its argument are shown in Figure A.1(a), Figure A.1(b) and Figure A.1(c).



(a) Very low SNR (b) Low SNR

(c) High SNR

Figure A.1: *Q function for three different SNR ranges.*

A Q function approximation that is good for $x > 3$:

$$Q(x) \approx \frac{1}{x\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \tag{A.15}$$

An accurate approximation (less than 1.2% of error) is:

$$Q(x) \approx \left[\frac{1}{\frac{\pi-1}{\pi}x + \frac{1}{\pi}\sqrt{x^2+2\pi}}\right] \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right). \tag{A.16}$$

The expression is valid for $x \geq 0$ and for $x < 0$ one should use $Q(-x) = 1 - Q(x)$. The function ak_qfuncApprox implements Eq. (A.16). In case you have Matlab, the code below compares it with Matlab's qfunc.

```
x=-3:0.01:3; %define a range for x
qx1=ak_qfuncApprox(x); %use the approximation
qx2=qfunc(x); %qfunc in Matlab's Communication Toolbox
plot(x,100*(qx1-qx2)./qx2); %get error in %
grid, xlabel('x'); ylabel('Error in Q approximation (%)')
```

A.6 Matched filter and Cauchy-Schwarz's inequality

Proving the matched filter is analogous to the following problem: given a vector $\mathbf{x}$, what is the vector $\mathbf{y}$ that maximizes the inner product $\langle \mathbf{x}, \mathbf{y} \rangle$? By Cauchy-Schwarz's inequality:

$$|\langle \mathbf{x}, \mathbf{y} \rangle||^2 \leq \langle \mathbf{x}, \mathbf{x} \rangle \langle \mathbf{y}, \mathbf{y} \rangle,$$

or taking the square root $|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \ \ \|\mathbf{y}\|$. The inner product is maximized when the vector $\mathbf{y}$ is colinear with $\mathbf{x}$.

A.7 Geometric series

A geometric series is the sum of numbers that form a geometric progression with common ration r and scale factor (or starting value) α:

$$\sum_{n=0}^{N-1} \alpha r^n = \alpha r^0 + \alpha r^1 + \alpha r^2 + \ldots + \alpha r^{N-1} = \frac{\alpha(1 - r^N)}{1 - r}.$$

When the number N of terms goes to infinity, the series converges if and only if $|r| < 1$. In this case

$$\sum_{n=0}^{\infty} \alpha r^n = \frac{\alpha}{1 - r}, |r| < 1. \tag{A.17}$$

A.8 Sum of squares

The sum of the squares of the first N integers is

$$\sum_{n=1}^{N} n^2 = \frac{N(N+1)(2N+1)}{6}, \tag{A.18}$$

which can be proved by induction [urlBMson].

A.9 Summations and integrals

Note that

$$\left(\sum_{n=1}^{N} x[n] \right) \left(\sum_{n=1}^{N} y[n] \right) = \sum_{n=1}^{N} \sum_{m=1}^{N} (x[n]y[m])$$

because, e.g., $(a + b + c)(d + e + f) = ad + ae + af + bd + be + bf + cd + ce + cf$. Similarly, in the continuous-case

$$\left(\int_{\langle T_0 \rangle} x(t)dt \right) \left(\int_{\langle T_0 \rangle} y(t)dt \right) = \int_{\langle T_0 \rangle} \int_{\langle T_0 \rangle} (x(t)y(s))\, dtds,$$

where one should note the adoption of distinct integration variables t and s. This result allows to express

$$\left(\int_{\langle T_0 \rangle} x(t)dt\right)^2 = \int_{\langle T_0 \rangle}\int_{\langle T_0 \rangle} (x(t)x(s))\, dtds, \tag{A.19}$$

which is an useful expression. Note that, in general,

$$\left(\int_{\langle T_0 \rangle} x(t)dt\right)^2 \neq \int_{\langle T_0 \rangle} x^2(t)dt.$$

A.10 Partial fraction decomposition

A *partial fraction decomposition* is used to convert a rational function $P(x)/Q(x)$ into a sum of simpler fractions, where $P(x)$ and $Q(x)$ are two polynomials with N and M being the degrees of $P(x)$ and $Q(x)$, respectively and $x \in \mathbb{C}$.

Two assumptions simplify the decomposition:

1. $M > N$, i. e., the denominator has larger degree than the numerator,
2. all M roots p_i of the denominator (called poles when dealing with transforms such as Laplace and Z) are distinct, which allows to write $Q(x) = (x - p_1)(x - p_2)\ldots(x - p_M)$.

In this special case, it is possible to write

$$\frac{P(x)}{Q(x)} = \frac{r_1}{x - p_1} + \frac{r_2}{x - p_2} + \ldots + \frac{r_1}{x - p_M}, \tag{A.20}$$

where

$$r_i = \left(\frac{P(x)}{Q(x)}(x - p_i)\right)\Big|_{x=p_i} \tag{A.21}$$

is called the *residue* of the (pole) p_i. To understand (and prove) Eq. (A.21), one can observe that multiplying both sides of Eq. (A.20) by $(x - p_1)$ leads to

$$\frac{P(x)}{Q(x)}(x - p_1) = r_1 + \frac{r_2(x - p_1)}{x - p_2} + \ldots + \frac{r_1(x - p_1)}{x - p_M}$$

and substituting x by p_1 makes all terms $r_i(x - p_1)/(x - p_i), i \neq 1$, equal to zero. The same can be done to the other poles and the general expression for this procedure is Eq. (A.21). For example, expanding $1/(x^2 - 5x + 6)$ leads to

$$\frac{1}{x^2 - 5x + 6} = \frac{r_1}{x - 2} + \frac{r_2}{x - 3},$$

where

$$r_1 = \left(\frac{1}{(x - 2)(x - 3)}(x - 2)\right)\Big|_{x=2} = \frac{1}{x - 3}\Big|_{x=2} = -1$$

and

$$r_2 = \left(\frac{1}{(x-2)(x-3)}(x-3)\right)\Big|_{x=3} = 1.$$

If the roots are complex (typically they occur as complex conjugate pairs), the procedure is similar, but the parcels can be rearranged.

When the first assumption is not valid, one needs to use polynomial division to first obtain

$$\frac{P(x)}{Q(x)} = R(x) + \frac{S(x)}{Q(x)},$$

where the degree L of $S(x)$ is $L < M$. This pre-processing stage is similar to writing an improper fraction as a mixed fraction, e. g., $\frac{7}{4} = 1\frac{3}{4} = 1 + \frac{3}{4}$. For example, when $P(x)/Q(x) = (3x^3 - 13x^2 + 8x + 13)/(x^2 - 5x + 6)$, some algebra shows that it is not possible to find two residues r_1 and r_2 such that

$$\frac{P(x)}{Q(x)} = \frac{r_1}{x-2} + \frac{r_2}{x-3}.$$

Hence, first one obtains

$$\frac{P(x)}{Q(x)} = \frac{3x^3 - 13x^2 + 8x + 13}{x^2 - 5x + 6} = 3x + 2 + \frac{1}{x^2 - 5x + 6},$$

with $R(x) = 3x + 2$ and $S(x) = 1$ having a degree $L = 0$ smaller than $M = 2$, and then uses the standard partial fraction expansion on $\frac{S(x)}{Q(x)}$ to obtain

$$\frac{3x^3 - 13x^2 + 8x + 13}{x^2 - 5x + 6} = 3x + 2 + \frac{1}{x-3} - \frac{1}{x-2}.$$

When one or more roots p_i of $Q(x)$ (poles) have multiplicity λ_i larger than one, the second simplifying assumption does not hold and the expansion is trickier as discussed in the sequel.

Note that, in general, $Q(x)$ can be written as $Q(x) = (x - p_1)^{\lambda_1}(x - p_2)^{\lambda_2} \ldots (x - p_M)^{\lambda_M}$, while the previous results were restricted to $\lambda_i = 1, \forall i$. A pole p_i with $\lambda_i > 1$ requires not only a parcel $r_i/(x - p_i)$ but λ_i parcels with residues $r_{ij}, j = 1, \ldots, \lambda_i$ for the following powers of $(x - p_i)$:

$$\frac{r_{i1}}{(x-p_i)} + \frac{r_{i2}}{(x-p_i)^2} + \frac{r_{i3}}{(x-p_i)^3} + \ldots + \frac{r_{i\lambda_i}}{(x-p_i)^{\lambda_i}}.$$

The residues can be obtained using factorial and derivatives via the *Theorem of residuals*:

$$r_{ij} = \frac{1}{(\lambda_i - j)!}\frac{d^{\lambda_i - j}}{dx^{\lambda_i - j}}\left(\frac{P(x)}{Q(x)}(x-p_i)^{\lambda_i}\right)\Big|_{x=p_i} \tag{A.22}$$

for $j = 1, \ldots, \lambda_i$. When $\lambda_i = 1$, this equation simplifies to Eq. (A.21). For example, the expansion of $(x^3 + 5)/(x^4 - 9x^3 + 30x^2 - 44x + 24)$ can use Eq. (A.22) because the

denominator can be written as $(x-2)^3(x-3)$, having a single pole $p_1 = 3$ and a pole $p_2 = 2$ with multiplicity 3. Hence, the following residues need to be found

$$\frac{x^3+5}{x^4-9x^3+30x^2-44x+24} = \frac{r_1}{(x-p_1)} + \frac{r_{21}}{(x-p_2)} + \frac{r_{22}}{(x-p_2)^2} + \frac{r_{23}}{(x-p_2)^3}.$$

The residue r_1 can be found with Eq. (A.21):

$$r_1 = \left(\frac{x^3+5}{(x-2)^3(x-3)}(x-3)\right)\Bigg|_{x=3} = \frac{3^3+5}{(3-2)^3} = 32,$$

while the other residues are given by Eq. (A.22) and require using Eq. (A.23) to obtain the following derivatives:

$$\frac{d}{dx}\left(\frac{x^3+5}{x-3}\right) = \frac{3x^2(x-3)-(x^3+5)}{(x-3)^2} = \frac{2x^3-9x^2-5}{(x-3)^2}$$

and

$$\frac{d}{dx}\left(\frac{2x^3-9x^2-5}{(x-3)^2}\right) = \frac{(6x^2-18x)(x-3)^2-(2x^3-9x^2-5)2(x-3))}{(x-3)^4},$$

which will be used for calculating r_{22} and r_{21}, respectively. Therefore,

$$r_{21} = \frac{1}{(3-1)!}\frac{d^2}{dx^2}\left(\frac{P(x)}{Q(x)}(x-2)^3\right)\Bigg|_{x=2} = \frac{1}{2}(-62) = -31,$$

$$r_{22} = \frac{1}{(3-2)!}\frac{d}{dx}\left(\frac{P(x)}{Q(x)}(x-2)^3\right)\Bigg|_{x=2} = -25$$

and

$$r_{23} = \frac{1}{(3-3)!}\left(\frac{P(x)}{Q(x)}(x-2)^3\right)\Bigg|_{x=2} = -13.$$

It is useful to use algebra and double check the obtained expansion:

$$\frac{x^3+5}{x^4-9x^3+30x^2-44x+24} = \frac{32}{(x-3)} + \frac{-31}{(x-2)} + \frac{-25}{(x-2)^2} + \frac{-13}{(x-2)^3}.$$

Alternatively, one can use Matlab/Octave to obtain the residues with the commands `b=[1 0 0 5],a=[1 -9 30 -44 24],[r,p,k]=residue(b,a)`. It should be noted that Octave has the option of a more complete output with `[r,p,k,e]=residue(b,a)`, where the vector `e` relates each residue to the corresponding parcel in the expansion. When using Matlab, one needs to know that the residues are given in the order $r_{i1}, r_{i2}, \ldots, r_{i\lambda_i}$.

A.11 Calculus

1. Derivative product rule: $(f(x)g(x))' = f(x)g'(x) + f'(x)g(x)$
2. Derivative of a rational function

$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)} \tag{A.23}$$

3. Integration by parts:

$$\int f(x)g'(x)dx = f(x)g(x) - \int f'(x)g(x)dx \tag{A.24}$$

4. Derivative of an exponential: $(e^{f(t)})' = e^{f(t)}f'(t)$
5. Integral of an exponential: $\int (e^{f(t)})dt = \frac{e^{f(t)}}{f'(t)}$

A.12 Sinc Function

Our definition of sinc is:

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}.$$

Some authors call it Sa (sample function) and others do not include π in the definition. Its first zero occurs when $x = 1$. Its value $\text{sinc}(0) = 1$ at origin can be determined using L'Hospital rule. The sinc is an energy signal with unitary energy $E = 1$, which can be determined by its Fourier transform and Parseval's relation. Its scaled version $\text{sinc}(t/T_s)$ is widely used in sampling theory and has energy $E = T_s$. The sincs $\text{sinc}(t)$ are orthogonal when shifted by integers $m, n \in \mathbb{Z}$ (e. g., $\text{sinc}(t-3)$ and sinc(t+1) are orthogonal) and, consequently, the scaled sincs $\text{sinc}(t/T_s)$ are orthogonal when shifted by multiples of T_s, i. e.

$$\int_{-\infty}^{\infty} \text{sinc}\left(\frac{t - mT_s}{T_s}\right) \text{sinc}\left(\frac{t - nT_s}{T_s}\right) dt = T_s\delta[m-n]. \tag{A.25}$$

A.13 Rectangular Integration to Define Normalization Factors for Functions

In several situations a computer is used to obtain points that should represent a continuous function $f(x)$, $x \in \mathbb{R}$. Two examples of this situation are the estimation of probability density functions (PDF) via histograms and power spectral density (PSD) estimation via an FFT routine.

Instead of aiming at an analytical expression $\hat{f}(x)$ to represent $f(x)$, the task consists in obtaining a set of points $\{\hat{f}(x[n])\}$ calculated at the values $x[n]$, $n = 0, \ldots, N-1$, which are a uniformly-sampled version of the abscissa x.

Often it is possible to first obtain a set of values $\{\hat{g}(x[n])\}$ in which the value $\hat{g}(x[n])$ is proportional to $f(x[n])$, i. e., $f(x[n]) \propto \hat{g}(x[n])$. In this case, it is required to later determine a scaling factor $\kappa \in \mathbb{R}$ such that the final set of values to represent $f(x)$ is obtained via

$$\hat{f}(x[n]) = \kappa \hat{g}(x[n]). \tag{A.26}$$

Note that the goal is not necessarily to have $\hat{f}(x) \approx f(x)$. There are situations in which the set of points $\{\hat{f}(x[n])\}$ must obey a property. For example, when histograms are used to estimate probability mass functions, one desired property is that $\sum_{x[n]} \hat{f}(x[n]) = 1$. Alternatively, the goal may be to scale the histogram such that the two resulting curves (normalized histogram and probability density function) coincide. The values of κ are different for these two possible cases of histogram normalization as discussed after recalling the rectangle method.

The *rectangle method* [urlBMrec] is used for approximating a definite integral:

$$\int_a^b f(x)dx \approx h \sum_{n=0}^{N-1} f(x[n]), \tag{A.27}$$

where $h = (b-a)/N$ is the rectangle width and $x[n] = a + nh$. Rectangle integration is useful to calculate κ in the two cases discussed in the sequel.

A.13.1 Two normalizations for the histogram

When the task is to estimate the PDF $f(x)$ of a continuous random variable, one can try using a discrete histogram $\hat{g}(x[n])$, which is obtained by drawing M values from $f(x)$ and counting the number of values occurring at each of B bins. Intuitively, for large M and B, the curve (or the "envelope") of the histogram resembles $f(x)$ but it is off by a normalization factor κ.

If $\kappa = 1/M$ is chosen, which is the most adopted option, one has $\hat{f}(x[n]) = (1/M)\hat{g}(x[n])$ and, consequently $\sum_n \hat{f}(x[n]) = 1$. However, in this case, $\hat{f}(x[n])$ may be far from $f(x)$ by a large scaling factor. This can be observed in the curves generated by the following code:

```
M=1000; x=3*rand(1,M); %M random numbers from 0 to 3
B=100; [hatgx,N_x]=hist(x,B); %histogram with B bins
hatfx = hatgx/M; %normalize the histogram to sum up to 1
plot(N_x,hatfx,[-1,0,0,3,3,4],[0,0,1/3,1/3,0,0],'o-')
xlabel('random variable x'), ylabel('PDF f(x)')
legend('estimated','theoretical'); sum(hatfx)
```

The result of sum(hatfx) is equal to one, as specified, but the PDF of the simulated distribution $\mathcal{U}(0,3)$ is $1/3$ over its support and the superimposed estimated and theoretical graphs do not match. This discrepancy between the curves should be expected given that the normalized histogram hatfx was in fact an estimate of a probability mass function (PMF) of a discrete random variable, obtained by quantizing the original x. Another normalization factor $\kappa \neq 1/M$ must be used if the goal is to have $f(x) \approx \hat{f}(x[n])$.

To obtain κ such that $f(x) \approx \hat{f}(x[n])$, one can use the property that the integral of a PDF is one. Based on the rectangle method one can write

$$\int_a^b f(x)dx \approx h \sum_{n=0}^{N-1} \hat{f}(x[n]) = h\kappa \sum_{n=0}^{N-1} \hat{g}(x[n]) = 1,$$

where $h = (b-a)/B$. Because $\sum_{n=0}^{N-1} \hat{g}(x[n]) = M$, one obtains $\kappa = 1/(hM)$, which is the original factor $1/M$ divided by the bin width h. The function ak_normalize_histogram.m uses this approach. Using the same example of the previous code, the following commands for obtaining hatfx would lead to consistent theoretical and estimated curves:

```
M=1000; x=3*rand(1,M); %M random numbers from 0 to 3
B=100; [hatgx,N_x]=hist(x,B); %histogram with B bins
h=3/B; %h is the bin width assuming the support is 3
hatfx = hatgx/(M*h); %PDF values via normalized histogram
plot(N_x,hatfx,[-1,0,0,3,3,4],[0,0,1/3,1/3,0,0],'o-')
xlabel('random variable x'), ylabel('PDF f(x)')
legend('estimated','theoretical'); sum(hatfx)
```

As expected, in contrast to the sum equal to one in the first code, in this case sum(hatfx)=1/h=33.3. Both histogram normalization factors, $\kappa = 1/M$ and $\kappa = 1/(hM)$, are useful and the choice depends whether the application requires values from a PMF or PDF, respectively.

A.13.2 Two normalizations for PSD estimation via FFT

Another application that can be related to Eq. (A.18) is using FFT for PSD estimation. In this case, a finite-duration discrete-time signal $x[n]$ with N non-zero samples is assumed.

The periodogram $\hat{S}[k] = \frac{|\text{FFT}\{x[n]\}|^2}{N}$ (Eq. (**??**)) plays the role of the function $\hat{g}$ in Eq. (A.26). The choice $\kappa = 1$ leads to an estimate $\hat{f}(\cdot)$ of the PSD $S(e^{j\Omega})$ in Eq. (**??**), while $\kappa = 1/N$ corresponds to estimating the mean-square spectrum (MSS) $S_{\text{ms}}[k]$ of Eq. (**??**). As indicated in Table A.1, the two options for κ have similarities with the ones for histogram normalization.

Table A.1: *Analogy between using the histogram and DFT for estimation, where $\hat{g}(x[n])$ is the estimated function and $\hat{f}(x[n]) = \kappa\hat{g}(x[n])$ its normalized version. The unit of $\hat{f}(x[n])$ is indicated within parentheses.*

	$\hat{g}(\cdot)$ is histogram	$\hat{g}(\cdot)$ is periodogram ($\hat{S}[k] = \|\text{FFT}\{x[n]\}\|^2/N$)
Estimate a discrete function	$\kappa = 1/M$	$\kappa = 1/N$
	$\hat{f}(\cdot)$ is PMF (probability)	$\hat{f}(\cdot)$ is MSS $\hat{S}_{\text{ms}}[k]$ (Watts)
Estimate a continuous function	$\kappa = \frac{1}{hM}$	$\kappa = 1$
	$\hat{f}(\cdot)$ is PDF (likelihood)	$\hat{f}(\cdot)$ is PSD $\hat{S}(e^{j\Omega})$ (Watts/rad)

To prove the values of κ in Table A.1, consider first that the goal is to multiply the

periodogram values $\hat{S}[k]$ by κ such that resulting values sum up to the average power $\mathcal{P}$, i. e.

$$\sum_{k=0}^{N-1} \kappa \hat{S}[k] = \mathcal{P}. \tag{A.28}$$

Recall from the definition of MS spectrum $\hat{S}_{\text{ms}}[k]$ in Eq. (**??**), that $\sum_{k=0}^{N-1} \hat{S}_{\text{ms}}[k] = \mathcal{P}$. And from Eq. (**??**), $\hat{S}_{\text{ms}}[k] = \frac{1}{N}\hat{S}[k]$, such that Eq. (A.28) can be written as $\sum_{k=0}^{N-1} \kappa \hat{S}[k] = N\kappa \sum_{k=0}^{N-1} \hat{S}_{\text{ms}}[k] = \mathcal{P}$ or $\kappa N = 1$. Therefore $\kappa = 1/N$ allows to normalize the periodogram to interpret it as a discrete function with the property indicated by Eq. (A.28).

Alternatively, if the goal is to interpret the periodogram as a continuous function with values corresponding to estimates of the PSD $S(e^{j\Omega})$, from Eq. (**??**) and the rectangle method of Eq. (A.27), a property that can be used to determine κ is

$$\frac{1}{2\pi}\int_{<2\pi>} S(e^{j\Omega})d\Omega \approx \frac{1}{2\pi} h \sum_{k=0}^{N-1} \kappa \hat{S}[k] = \mathcal{P}.$$

Substituting the bin width $h = 2\pi/N$ and normalizing the periodogram by N to convert it into the MS spectrum leads to

$$\frac{1}{2\pi} h \sum_{k=0}^{N-1} \kappa \hat{S}[k] = \frac{1}{2\pi}\frac{2\pi}{N}\kappa \sum_{k=0}^{N-1} N \hat{S}_{\text{ms}}[k] = \mathcal{P},$$

such that $\kappa = 1$ in this case. Example **??** illustrates the two options for κ discussed in Table A.1.

A.14 Linear Algebra

A.14.1 Inner products and norms

The inner product in a K-dimensional space with complex-valued vectors is:

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{K} \mathbf{a}_i \mathbf{b}_i^* = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos(\theta). \tag{A.29}$$

See Table 2.2 for alternative definitions of inner products.

When $\mathbf{a} = \mathbf{b}$, Eq. (A.29) can be written as

$$\|\mathbf{a}\| = \sqrt{\langle \mathbf{a}, \mathbf{a} \rangle}. \tag{A.30}$$

An inner product $\langle \mathbf{a}, \mathbf{b} \rangle$ can also be written as a multiplication of two vectors

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{b}^H \mathbf{a}, \tag{A.31}$$

where in this case both are assumed to be column vectors (row vectors would suggest $\mathbf{ab}^H$).

In case a vector $\mathbf{b} = \mathbf{Aa}$ is obtained via multiplication by a unitary matrix $\mathbf{A}$, Eq. (A.30) and Eq. (A.31) lead to

$$\|\mathbf{b}\| = \sqrt{\langle \mathbf{Aa}, \mathbf{Aa} \rangle} = \sqrt{(\mathbf{Aa})^H \mathbf{Aa}} = \sqrt{\mathbf{a}^H \mathbf{A}^H \mathbf{Aa}} = \sqrt{\mathbf{a}^H \mathbf{Ia}} = \|\mathbf{a}\| \tag{A.32}$$

because $\mathbf{A}^H \mathbf{A} = \mathbf{I}$, which indicates that the unitary $\mathbf{A}$ does not alter the norm of the input vector.

A.14.2 Projection of a vector using inner product

To explore the properties and advantages of linear transforms, it is useful to study the *vector projection* (or simply projection) of a vector onto another one.

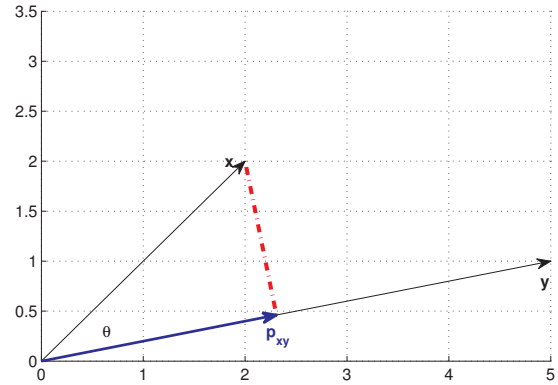


Figure A.2: *The perpendicular line for obtaining the projection $\boldsymbol{p}_{xy}$ of a vector $\boldsymbol{x}$ onto $\boldsymbol{y}$ in $\mathbb{R}^2$. Note that $\theta = \cos^{-1}(\langle \boldsymbol{x}, \boldsymbol{y} \rangle / (\|\boldsymbol{x}\| \, \|\boldsymbol{y}\|))$ is the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$ and the inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \|\boldsymbol{p}_{xy}\| \, \|\boldsymbol{y}\| = \|\boldsymbol{p}_{yx}\| \, \|\boldsymbol{x}\|$.*

Using $\mathbb{R}^2$ for simplicity, note that the projection $\mathbf{p}_{xy}$ of a vector $\mathbf{x}$ in another vector $\mathbf{y}$ is obtained by choosing the point along the direction of $\mathbf{y}$ that has the minimum distance to $\mathbf{x}$. This line is perpendicular to $\mathbf{y}$, as indicated in Figure A.2. Using the Pythagorean theorem and assuming that $0 \leq \theta \leq \pi/2$, the norm $\|\mathbf{p}_{xy}\|$ of the projection can be written as $\|\mathbf{p}_{xy}\| = \|\mathbf{x}\| \cos(\theta)$. If $\pi/2 < \theta \leq \pi$, $\|\mathbf{p}_{xy}\| = \|\mathbf{x}\| \cos(\pi - \theta) = -\|\mathbf{x}\| \cos(\theta)$. Hence, in general,

$$\|\mathbf{p}_{xy}\| = \|\mathbf{x}\| \, |\cos(\theta)| = \frac{|\langle \mathbf{x}, \mathbf{y} \rangle|}{\|\mathbf{y}\|}.$$

For a given norm $\|\mathbf{y}\|$, the larger the inner product, the larger the norm of the projection. The same is valid for $\mathbf{p}_{yx}$ as depicted in Figure A.3:

$$\|\mathbf{p}_{yx}\| = \|\mathbf{y}\| \, |\cos(\theta)| = \frac{|\langle \mathbf{x}, \mathbf{y} \rangle|}{\|\mathbf{x}\|}.$$

Any vector $\mathbf{z}$ can be written as its norm $\|\mathbf{z}\|$ multiplied by a unity norm vector $\frac{\mathbf{z}}{\|\mathbf{z}\|}$

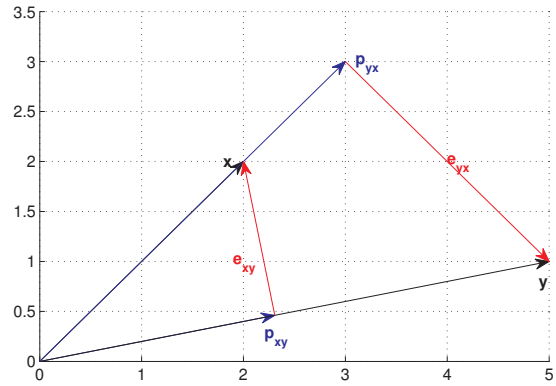


Figure A.3: *Projections of a vector* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *onto each other. Note the errors are orthogonal to the directions of the respective projections.*

that indicates its direction. Note that the vector $\mathbf{p}_{yx}$ is in the same or the opposite direction of $\mathbf{x}$, which can be specified by the unity norm vector $\text{sign}(\cos(\theta))\frac{\mathbf{x}}{\|\mathbf{x}\|}$. Hence, one has

$$\mathbf{p}_{yx} = \|\mathbf{p}_{yx}\|\text{sign}(\cos(\theta))\frac{\mathbf{x}}{\|\mathbf{x}\|} = \left(\frac{\langle \mathbf{x}, \mathbf{y}\rangle}{\|\mathbf{x}\|^2}\right)\mathbf{x},$$

where $\frac{\langle \mathbf{x},\mathbf{y}\rangle}{\|\mathbf{x}\|^2}$ is a scaling factor that can be negative but does not change the direction of $\mathbf{x}$. Similarly, the projection $\mathbf{p}_{xy}$ of $\mathbf{x}$ onto $\mathbf{y}$ is given by

$$\mathbf{p}_{xy} = \left(\frac{\langle \mathbf{x}, \mathbf{y}\rangle}{\|\mathbf{y}\|^2}\right)\mathbf{y}.$$

Note that if the vector $\mathbf{y}$ has unitary norm, the absolute value of the inner product $\langle \mathbf{x}, \mathbf{y}\rangle$ coincides with the norm $\|\mathbf{p}_{xy}\|$. These expressions are valid for other vector spaces, such as $\mathbb{R}^n$, $n > 2$.

Using geometry to interpret a projection vector is very useful. When one projects $\mathbf{x}$ in $\mathbf{y}$, the result $\mathbf{p}_{xy}$ is the "best" representation (in the minimum distance sense, or least-square sense) of $\mathbf{x}$ that $\mathbf{y}$ alone can provide. The error vector $\mathbf{e}_{xy} = \mathbf{x} - \mathbf{p}_{xy}$ is orthogonal to $\mathbf{y}$ (and, consequently, to $\mathbf{p}_{xy}$), i. e., $\langle \mathbf{e}_{xy}, \mathbf{y}\rangle = 0$. The vector $\mathbf{e}_{xy}$ represents what should be added to $\mathbf{p}_{xy}$ in order to completely represent $\mathbf{x}$, and the orthogonality $\langle \mathbf{e}_{xy}, \mathbf{y}\rangle = 0$ indicates that $\mathbf{y}$ cannot contribute any further.

Figure A.3 completes the example. It was obtained using the following Matlab/Octave script.[1] The example assumes $\mathbf{x} = [2, 2]$ and $\mathbf{y} = [5, 1]$, having an angle θ of approximately 33.7 degrees between them.

Listing A.1: *MatlabOctaveCodeSnippets/snip_transforms_projection.m*

```
x=[2,2]; %define a vector x
y=[5,1]; %define a vector y
magx=sqrt(sum(x.*x))%magnitude of x
```

[1] The function ak_drawvector.m was also used.

```
magy=sqrt(sum(y.*y))%magnitude of y
innerprod=sum(x.*y) %<x,y>=||x|| ||y|| cos(theta)
theta=acos(innerprod / (magx*magy)) %angle between x and y
%obs: could use acosd to have angle in degrees
disp(['Angle is ' num2str(180*theta/pi) ' degrees']);
%check if inverting direction is needed:
invertDirection=1; if theta>pi/2 invertDirection=-1; end
%find the projection of x over y (called p_xy) and p_yx
mag_p_xy = magx*abs(cos(theta)) %magnitude of p_xy
%directions: obtained as y normalized by magy, x by magx
y_unitary=y/magy; %normalize y by magy to get unitary vec.
p_xy = mag_p_xy * y_unitary* invertDirection %p_xy
mag_p_yx = magy*abs(cos(theta)); %magnitude of p_yx
x_unitary=x/magx; %normalize x by magx to get unitary vec.
p_yx = mag_p_yx * x_unitary* invertDirection %p_yx
%test orthogonality of error vectors:
error_xy = x - p_xy; %we know: p_xy + error_xy = x
sum(error_xy.*y) %this inner product should be zero
error_yx = y - p_yx; %we know: p_yx + error_yx = y
sum(error_yx.*x) %this inner product should be zero
```

The concept of projections via inner products will be extensively used in our discussion about transforms. For example, the coefficients of a Fourier series of a signal $x(t)$ correspond to the normalized projections of $x(t)$ on the corresponding basis functions. A large value for the norm of a projection indicates that the given basis function provides a good contribution in the task of representing $x(t)$.

Chapter 2 discusses block transforms and relies on orthogonal functions. Hence, it is useful to discuss why orthogonality is important.

A.14.3 Orthogonal basis allows inner products to transform signals

Assume the existence of a set of orthogonal vectors composing the basis of a vector space. For example, in $\mathbb{R}^2$, a convenient basis is the standard, composed by $\overline{i} = [1, 0]$ and $\overline{j} = [0, 1]$. The inner product $\langle \overline{i}, \overline{j} \rangle = 0$ indicates that these two vectors are orthogonal. The orthogonality property simplifies the following *analysis* task: given any vector $\mathbf{x}$, the coefficients α and β of the linear combination $\mathbf{x} = \alpha\overline{i} + \beta\overline{j}$, can be easily found by using the dot products $\alpha = \langle \mathbf{x}, \overline{i} \rangle$ and $\beta = \langle \mathbf{x}, \overline{j} \rangle$. The following theorem proves this important result.

Theorem 3. Analysis via inner products. If the basis set $\{\mathbf{b}_1, \ldots, \mathbf{b}_N\}$ of an inner product space (e. g., Euclidean) is orthonormal, the coefficients of a linear combination $\mathbf{x} = \sum_{i=1}^{N} \alpha_i \mathbf{b}_i$ that generates a vector $\mathbf{x}$ can be calculated by the inner product $\alpha_i = \langle \mathbf{x}, \mathbf{b}_i \rangle$ between $\mathbf{x}$ and the respective vector $\mathbf{b}_i$ in the basis set.

Proof: Recall the following properties of a dot product: $\langle \overline{a} + \overline{b}, \overline{c} \rangle = \langle \overline{a}, \overline{c} \rangle + \langle \overline{b}, \overline{c} \rangle$ and $\langle \alpha\overline{a}, \overline{b} \rangle = \alpha \langle \overline{a}, \overline{b} \rangle$ and write

$$\langle \mathbf{x}, \mathbf{b}_j \rangle = \langle \sum_{i=1}^{N} \alpha_i \mathbf{b}_i, \mathbf{b}_j \rangle = \sum_{i=1}^{N} \alpha_i \langle \mathbf{b}_i, \mathbf{b}_j \rangle$$

because the basis vectors are orthonormal, $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = 1$ if $i = j$ and $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = 0$ if $i \neq j$. Therefore,

$$\langle \mathbf{x}, \mathbf{b}_j \rangle = \alpha_j.$$

because all the terms in the above summation are zero but the one for $i = j$. □

Example A.1. **Obtaining the coefficients of a linear combination of basis functions**. A simple example can illustrate the analysis procedure: the coefficients of $\mathbf{x} = 4\bar{i} + 8\bar{j}$ are $\alpha = 4$ and $\beta = 8$ by inspection, but they could be calculated as $\alpha = \langle \mathbf{x}, \bar{i} \rangle = \langle [4, 8], [1, 0] \rangle = 4$ and $\beta = \langle \mathbf{x}, \bar{j} \rangle = \langle [4, 8], [0, 1] \rangle = 8$. Note that the zeros in these basis vectors make the calculation overly simple. Another example may be more useful to highlight orthogonality and the following alternative basis set is assumed: $\bar{i} = [0.5, 0.866]$ and $\bar{j} = [0.866, -0.5]$. Let $\mathbf{x} = 3\bar{i} + 2\bar{j} = [3.232, 1.5980]$. Given $\mathbf{x}$, the task is again to find the coefficients such that $\mathbf{x} = \alpha\bar{i} + \beta\bar{j}$. Due to the orthonormality of $\bar{i}$ and $\bar{j}$, one can for example obtain $\alpha = \langle \mathbf{x}, \bar{i} \rangle = \langle [3.232, 1.5980], [0.5, 0.866] \rangle = 3$. These computations can be done in Matlab/Octave as follows.

```
i=[0.5,0.866], j=[0.866 -0.5] %two orthonormal vectors
x=3*i+2*j %create an arbitrary vector x to be analyzed
alpha=sum(x.*i), beta=sum(x.*j) %find inner products
```

In contrast, let us modify the previous example, adopting a non orthogonal basis. Assume that $\bar{i} = [1, 1]$ and $\bar{j} = [0, 1]$ (note that $\langle \bar{i}, \bar{j} \rangle = 1$, hence the vectors are not orthogonal). Let $\mathbf{x} = 3\bar{i} + 2\bar{j} = [3, 5]$. In this case, $\langle \mathbf{x}, \bar{i} \rangle = 8$ and $\langle \mathbf{x}, \bar{j} \rangle = 5$, which do not coincide with the coefficients $\alpha = 3$ and $\beta = 2$. How could the coefficients be properly recovered in this case? An alternative is to write the problem as a set of linear equations, organize it in matrix notation and find the coefficients by inverting the matrix. In Matlab/Octave:

Listing A.2: *MatlabOctaveCodeSnippets/snip_transforms_non_orthogonal_basis.m*

```
i=transpose([1,1]), j=transpose([0,1]) %non-orthogonal
x=3*i+2*j %create an arbitrary vector x to be analyzed
A=[i j]; %organize basis vectors as a matrix
temp=inv(A)*x; alpha=temp(1), beta=temp(2) %coefficients
```

In summary, the analysis procedure for many linear transforms (such as Fourier, Z, etc.) obtain the coefficients via an inner product, and the procedure can be interpreted as calculating the projection of $\mathbf{x}$ onto basis $\bar{i}$ (eventually scaled by the norm of $\bar{i}$). □

This discussion leads to the conclusion that a basis with orthogonal vectors significantly simplifies the task: in this case, the analysis procedure can be done via inner products. This applies when the basis vectors do not have unitary-norm, but in this case a normalization by their norms is needed. Orthogonal basis vectors are a property of all block transforms discussed in this text.

A.14.4 Moore-Penrose pseudoinverse

Pseudoinverses[2] are generalizations of the inverse matrix and are useful when the given matrix does not have an inverse (for example, when the matrix is not square or full rank).

The Moore-Penrose pseudoinverse has several interesting properties and is adequate to least square problems. It provides the minimum-norm least squares solution $\mathbf{z} = \mathbf{X}\mathbf{b}$ to the problem of finding a vector $\mathbf{z}$ that minimizes the error vector norm $||\mathbf{X}\mathbf{z} - \mathbf{b}||$. Assuming $\mathbf{X}$ is an $m \times n$ matrix, the pseudoinverse provides the solution for a set of overdetermined or underdetermined equations if $m > n$ or $m < n$, respectively.

Two properties of a Moore-Penrose pseudoinverse $\mathbf{X}^+$ are

$$\mathbf{X}^H = \mathbf{X}^H \mathbf{X} \mathbf{X}^+, \tag{A.33}$$

and

$$\mathbf{X}^H = \mathbf{X}^+ \mathbf{X} \mathbf{X}^H. \tag{A.34}$$

With r being the rank of $\mathbf{X}$, then:

- if $m = n$ and $r = m = n$, the pseudoinverse $\mathbf{X}^+ = \mathbf{X}^{-1}$ is equivalent to the usual inverse;
- if $m > n$ (overdetermined) and, besides, $r = n$ (the columns of $\mathbf{X}$ are linearly independent), $\mathbf{X}^H \mathbf{X}$ is invertible and using Eq. (A.33) the pseudoinverse is given by
$$\mathbf{X}^+ = \left(\mathbf{X}^H \mathbf{X}\right)^{-1} \mathbf{X}^H; \tag{A.35}$$
- if $n > m$ (underdetermined) and $r = m$ (the rows of $\mathbf{X}$ are linearly independent), $\mathbf{X}\mathbf{X}^H$ is invertible and using Eq. (A.34) leads to
$$\mathbf{X}^+ = \mathbf{X}^H \left(\mathbf{X}\mathbf{X}^H\right)^{-1}. \tag{A.36}$$

Whenever available, instead of Eq. (A.35) or Eq. (A.36) that requires linear independence, one should use a robust method to obtain $\mathbf{X}^+$ such as the pinv function in Matlab/Octave, which adopts a SVD decomposition to calculate $\mathbf{X}^+$. Listing A.3 illustrates such calculations and the convenience of relying on pinv when the independence of rows or columns is not guaranteed.

Listing A.3: *MatlabOctaveCodeSnippets/snip_systems_pseudo_inverse.m*

```
test=3; %choose the case below
switch test
    case 1 %m>n (overdetermined) and linearly independent columns
        X=[1 2 3; -4+j -5+j -6+j;1 0 0;0 1 0];
    case 2 %n>m (underdetermined) and rows are linearly independent
        X=[1 2 3; -4+j -5+j -6+j];
    case 3 %neither rows nor columns of X are linearly independent
```

[2] See, e. g., [url3inv] and [url2psi].

```
        X=[1 2 3 3; 2 4 6 6; -4+j -5+j -6+j -12+3*j];
end
Xp_svd = pinv(X) %pseudoinverse via SVD decomposition
Xp_over = inv(X'*X)*X' %valid when columns are linearly independent
Xp_under = X'*inv(X*X') %valid when rows are linearly independent
rank(X'*X) %X'*X is square but may not be full rank
rank(X*X') %X*X' is square but may not be full rank
Xhermitian=X'*X*pinv(X) %equal to X' (this property is always valid)
Xhermitian2=pinv(X)*X*X' %equal to X' (the property itself is valid)
maxError_over=max(abs(Xp_svd(:)-Xp_over(:))) %error for overdetermin.
maxError_under=max(abs(Xp_svd(:)-Xp_under(:))) %for underdetermined
```

A.15 Gram-Schmidt orthonormalization procedure

The Gram-Schmidt is an automatic procedure to obtain a set of N orthonormal vectors $\mathbf{y}$ from an input set $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ composed by M vectors. If the M vectors are *linearly independent*, then $N = M$. If there is linear dependency among the vectors, then $N < M$. Listing A.4 illustrates the procedure.

In summary, the first basis function is $\mathbf{y}_1 = \mathbf{x}_1/\|\mathbf{x}_1\|$. Because $\mathbf{y}_1$ is the normalized first input vector, it can properly represent $\mathbf{x}_1$. The next step is to iteratively (in a loop) add new basis functions to represent the remaining vectors. For example, if $\mathbf{x}_2$ were colinear to $\mathbf{y}_1$, it would not be necessary to enlarge the basis set ($\mathbf{y}_1$ could represent both $\mathbf{x}_1$ and $\mathbf{x}_2$). If that is not the case, $\mathbf{y}_2$ cannot be simply $\mathbf{y}_2 = \mathbf{x}_2/\|\mathbf{x}_2\|$ because, eventually, $\mathbf{y}_1$ and $\mathbf{y}_2$ can be linearly dependent. Therefore, first $\mathbf{x}_2$ is projected in $\mathbf{y}_1$ and $\mathbf{y}_2$ is the (normalized) error vector of this projection. This process is repeated. For example, when obtaining $\mathbf{y}_i$ to represent a vector $\mathbf{x}_j$, one first takes in account the projection of $\mathbf{x}_j$ into all previously selected basis $1, 2, \ldots, i-1$. If the error is not numerically negligible (given a tolerance), this suggests that $\mathbf{x}_j$ does not reside in the space spanned by the current set of basis functions and the normalized projection error is added to this set.

Listing A.4: *MatlabOctaveFunctions/ak_gram_schmidt.m*

```
function [Ah,A]=ak_gram_schmidt(x,tol)
% function [Ah,A]=ak_gram_schmidt(x,tol)
%Gram-Schmidt orthonormalization procedure (x is real)
%Inputs:  x - real matrix: each row is an input vector
%         tol - tolerance for stopping procedure (and SVD)
%Outputs: Ah  - direct matrix, which is the Hermitian (transpose in
%               this case) of A
%         A   - inverse matrix with basis functions as columns
%Example of usage:
% x=[1 1; 0 2; 1 0; 1 4]
% [Ah,A]=ak_gram_schmidt(x,1e-12) %basis functions are columns of A
% testVector=[10; 10]; %test: vector in same direction as first basis
% testCoefficients=Ah*testVector %result is [14.1421; 0]

%% Notes about the code:
```

```
%1) The inner product sum(y(m,:).*x(k,:)) is alternatively
%   calculated as y(m,:)*x(k,:)' (row vectors)
%2) In general, a projection is projectionOverBasis=p_xy=
%              ( <y,x> / ||y||^2 ) * y =
%              ((y(m,:)*x(k,:)')/(y(m,:)*y(m,:)'))*y(m,:);
%but in the code y(m,:)*y(m,:)'=1 and the denominator is 1
if nargin<2
    tol=min(max(size(x)')*eps(max(x))); %find tolerance
end
[m,dimension]=size(x);
if dimension>m
    warning('Input dimension larger than number of vectors');
end
N=rank(x,tol); %note: rank is slow because it uses SVD
y=zeros(N,dimension); %pre-allocate space
%pick first vector and normalize it
y(1,:)=x(1,:)/sqrt(sum(x(1,:).^2));
numBasis = 1;
for k=2:m
    errorVector=x(k,:); %error (or target) vector
    for m=1:numBasis%go over all previously selected basis
        %p_xy = <x,y> * y; %recall, in this case: ||y||=1
        projectionOverBasis = ((y(m,:)*x(k,:)'))*y(m,:);
        %update target:
        errorVector = errorVector - projectionOverBasis;
    end
    magErrorVector = sqrt(sum(errorVector.^2));
    if ( magErrorVector > tol )
        %keep the new vector in basis set
        numBasis = numBasis + 1;
        y(numBasis,:)= errorVector / magErrorVector;
        if (numBasis >= N)
            break; %Abort. Reached final number of vectors
        end
    end
end
A=transpose(y);  %basis functions are the columns of inverse matrix A
Ah=A'; %the direct matrix transform is the Hermitian of A
```

The result of the Gram-Schmidt procedure depends on the order of the input vectors. Rearranging these vectors may produce a different (still valid) solution. An example of Gram-Schmidt execution can be found in Application 2.1.

The following section presents another procedure, which has similarities to the Gram-Schmidt and very interesting properties.

A.16 Principal component analysis (PCA)

Principal component analysis (PCA) or Karhunen-Loève transform (KLT) is an orthogonal linear transformation typically used to transform the input data to a new coordinate system such that the greatest variance by any projection of the data comes

to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

The following development does not try to be mathematically rigorous, but provide intuition. As in the Gram-Schmidt procedure, assume the goal is to obtain a set of N orthonormal vectors $\mathbf{y}$ from an input set $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ composed by M vectors of dimension K. An important point is that all elements of an input vector $\mathbf{x} = [\mathsf{X}_1, \ldots, \mathsf{X}_K]$ are assumed to have zero mean (or the mean is subtracted before PCA), i. e., $\mathbb{E}[\mathsf{X}_i] = 0, \forall i$. Instead of arbitrarily picking the first vector $\mathbf{x}_1$ as in Gram-Schmidt, PCA seeks the vector (first *principal component*) that maximizes the variance of the projected data. Restricting the basis function to have unity norm $\|\mathbf{y}\| = 1$, the absolute value of the inner product $\langle \mathbf{y}, \mathbf{x} \rangle$ corresponds to the norm of the projection of $\mathbf{x}$ over $\mathbf{y}$. PCA aims at minimizing the variance of this norm or, equivalently, the variance of $\langle \mathbf{y}, \mathbf{x} \rangle$. Because $\mathbf{y}$ is a fixed vector and $\mathbb{E}[\mathbf{x}] = 0$, it can be deduced that and $\mathbb{E}[\langle \mathbf{y}, \mathbf{x} \rangle] = 0$ and, consequently, the variance coincides with $\mathbb{E}\{\langle \mathbf{y}, \mathbf{x} \rangle^2\}$. Hence, the first PCA basis function is given by

$$\mathbf{y}_1 = \arg \max_{\|\mathbf{y}\|=1} \mathbb{E}\{\langle \mathbf{y}, \mathbf{x}_i \rangle^2\}.$$

After obtaining $\mathbf{y}_1$, similarly to Gram-Schmidt, one finds new targets by projecting all M input vectors in $\mathbf{y}_1$ and keeping the errors as targets $\hat{\mathbf{x}}$. For example, the targets for finding $\mathbf{y}_2$ are

$$\hat{\mathbf{x}}_i = \mathbf{x}_i - \langle \mathbf{x}_i, \mathbf{y}_1 \rangle \mathbf{y}_1, \quad i = 1, \ldots, M,$$

and the new basis function is obtained by

$$\mathbf{y}_2 = \arg \max_{\|\mathbf{y}\|=1} \mathbb{E}\{\langle \mathbf{y}, \hat{\mathbf{x}}_i \rangle^2\}.$$

Similarly,

$$\mathbf{y}_3 = \arg \max_{\|\mathbf{y}\|=1} \mathbb{E}\{\langle \mathbf{y}, \mathbf{x}_i - (\langle \mathbf{x}_i, \mathbf{y}_1 \rangle \mathbf{y}_1 + \langle \mathbf{x}_i, \mathbf{y}_2 \rangle \mathbf{y}_2) \rangle\}$$

and so on.

It is out of the scope of this text[3] to examine how to solve the maximization problems and find the optimal $\mathbf{y}$ vectors, but it is noted that the solution can be obtained via eigenvectors of the covariance matrix or singular value decomposition (SVD). Listing A.5 illustrates the former procedure, which uses eigen analysis. Note that the order of importance of the principal components $\mathbf{y}$ is given by the magnitude of the respective eigenvalues.

Listing A.5: *MatlabOctaveFunctions/ak_pcamtx.m*

```
function [Ah,A,lambdas] = ak_pcamtx(data)
% function [Ah,A,lambdas] = ak_pcamtx(data)
%Principal component analysis (PCA).
%Input:
```

[3] See, e. g., the Web for more details: [url2pca], [url2svd] and [url2fou].

```
%  data - M x D input matrix (M vectors of dimension D each)
%Outputs:
%  Ah - direct matrix, Hermitian (transpose in this case) of A
%  A  - inverse matrix with principal components (or eigenvectors) in
%       each column
%  lambdas - Mx1 matrix of variances (eigenvalues)
%Example of usage:
%x=[1 1; 0 2; 1 0; 1 4]
%[Ah,A]=ak_pcamtx(x) %basis functions are columns of A
%testVector=[1; 1]; %test: vector in same direction as first basis
%testCoefficients=Ah*testVector %result is [0.9683; -1.0307]
covariance = cov(data); % calculate the covariance matrix
[pc,lambdas]=eig(covariance);%get eigenvectors/eigenvalues
lambdas = diag(lambdas); % extract the matrix diagonal
[temp,indices]= sort(-1*lambdas);%sort in decreasing order
lambdas = lambdas(indices); %rearrange
A = pc(:,indices); %obtain sorted principal components: columns of A
Ah=A'; %provide also the direct transform matrix (Hermitian of A)
```

PCA is typically used for dimensionality reduction in a data set by retaining those characteristics of the data that contribute most to its variance, by keeping lower-order principal components and ignoring higher-order ones. Therefore, it is also useful for coding. One important question is why maximizing the variance is a good idea. In fact, this is not always the case and depends on the application.

The utility of PCA for representing signals will be illustrated by an example with data draw from a bidimensional Gaussian $f_{\mathbf{X}_1,\mathbf{X}_2}(x_1, x_2)$ with mean $\hat{\boldsymbol{\mu}} = (1,\ 3)^T$ and covariance matrix $\mathbf{C} = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 4 \end{bmatrix}$. Figure A.4 provides a scatter plot of the data and the basis functions obtained via PCA and, for comparison purposes, Gram-Schmidt orthonormalization. It can be seen that PCA aligns the basis with the directions where the variance is larger. The solution obtained with Gram-Schmidt depends on the first vector and the only guarantee is that the basis functions are orthonormal.

As already indicated, the basis functions of a orthonormal basis can be organized as the columns of a matrix $\mathbf{A}$. Examples will be provided in the sequel where $\mathbf{A}$ stores the basis obtained with PCA or the Gram-Schmidt procedure. At this point it is interesting to adopt the same convention used in most textbooks: instead of Eq. (2.1), the input and output vectors are assumed to be related by $\mathbf{x} = \mathbf{A}\mathbf{y}$. In other words, it is the inverse of $\mathbf{A}$ that transforms $\mathbf{x}$ into $\mathbf{y}$: $\mathbf{y} = \mathbf{A}^{-1}\mathbf{x}$. This is convenient because $\mathbf{y}$ will be interpreted as storing the *transform coefficients* of the linear combination of basis functions (columns of $\mathbf{A}$) that leads to $\mathbf{x}$.

It is possible now to observe the interesting effect of transforming input vectors using PCA and Gram-Schmidt. Using the same input data that generated Figure A.4, Figure A.5 and Figure A.6 show the scatter plots of $\mathbf{y}$ obtained by $\mathbf{y} = \mathbf{A}^{-1}\mathbf{x}$, where $\mathbf{A}$ represents the basis from PCA and Gram-Schmidt, respectively.

Comparing Figure A.5 and Figure A.6 clearly shows that PCA does a better job in extracting the correlation between the two dimensions of the input data. In the Gram-Schmidt case, the figure shows a negative correlation between the coefficients y_1

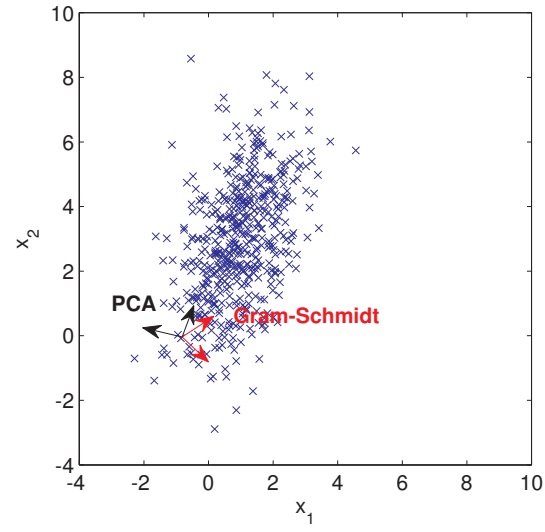


Figure A.4: *Scatter plot of the input data and the basis functions obtained via PCA and Gram-Schmidt orthonormalization.*

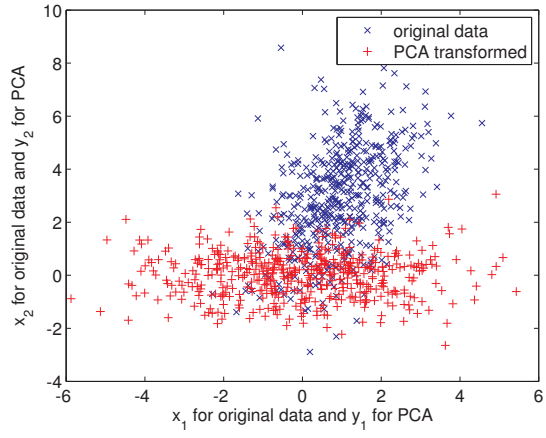


Figure A.5: *Scatter plots of two-dimensional Gaussian vector* $\boldsymbol{x}$ *(represented by x) and PCA transformed vectors* $\boldsymbol{y}$ *(represented by +). Note that the first dimension* y_1 *contains most of the variance in the data.*

and y_2. This is an example of a feature that may be useful. In other applications, the orthonormal basis provided by Gram-Schmidt may be enough as, for example, to obtain the basis functions in Eq. (**??**) for digital communications. In summary, some linear transforms will be designed such that the basis functions have specified properties while others will focus on properties of the coefficients.

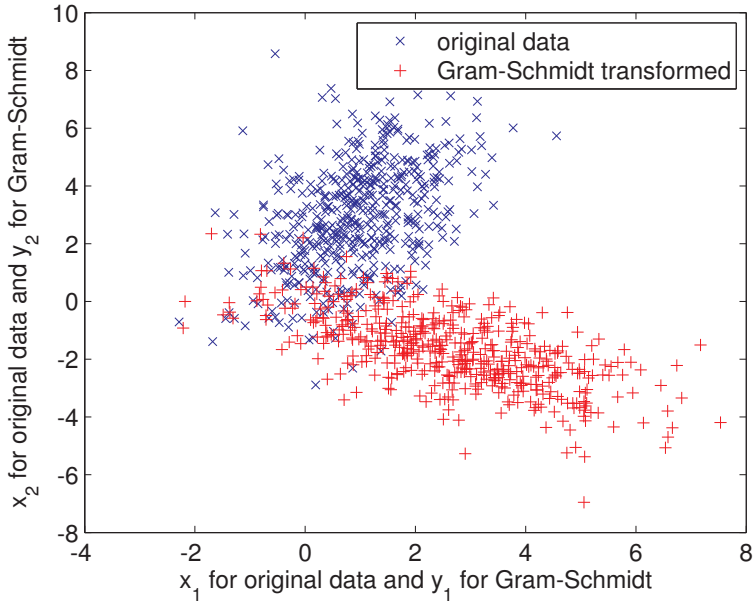


Figure A.6: *Scatter plots of two-dimensional Gaussian vector* $\boldsymbol{x}$ *(x) and Gram-Schmidt transformed vectors* $\boldsymbol{y}$ *(+).*

A.17 Fourier Analysis: Properties

In the sequel, it is assumed that $X(f)$, $Y(f)$ and $Z(f)$ are the Fourier transforms of $x(t)$, $y(t)$ and $z(t)$, respectively. A pair (time / frequency) is denoted by $\Leftrightarrow$. The following discussion assumes the Fourier transform, but the properties are valid for all four Fourier tools with subtle distinctions.

Linearity: if a signal $x(t) = \alpha y(t) + z(t)$ is obtained by multiplying $y(t)$ by a constant α and summing the result to $z(t)$, its transform is $X(f) = \alpha Y(f) + Z(f)$. Linearity can be stated as:

$$\alpha x(t) + \beta y(t) \Leftrightarrow \alpha X(f) + \beta Y(f). \tag{A.37}$$

Linearity can be decomposed into two properties: a) *homogeneity* and b) *additivity*, which correspond to the properties $\alpha x(t) \Leftrightarrow \alpha X(f)$ and $x(t) + y(t) \Leftrightarrow X(f) + Y(f)$, respectively.

Time-shift:

$$x(t - t_0) \Leftrightarrow X(f)e^{-j2\pi f t_0}. \tag{A.38}$$

Scaling:

$$x(at) \Leftrightarrow \frac{1}{|a|}X(f/a). \tag{A.39}$$

Time-reversal (scaling with $a = -1$):

$$x(-t) \Leftrightarrow X(-f). \tag{A.40}$$

Complex-conjugate:

$$x^*(t) \Leftrightarrow X^*(-f). \tag{A.41}$$

Combined time-reversal and complex-conjugate:

$$x^*(-t) \Leftrightarrow X^*(f). \tag{A.42}$$

Multiplication:

$$x(t)y(t) \Leftrightarrow X(f)*Y(f). \tag{A.43}$$

Frequency-shift:

$$x(t)e^{j2\pi f_0 t} \Leftrightarrow X(f - f_0). \tag{A.44}$$

Convolution:

$$x(t)*y(t) \Leftrightarrow X(f)Y(f). \tag{A.45}$$

Duality:

$$X(t) \Leftrightarrow x(-f). \tag{A.46}$$

Example: $\text{rect}(t) \Leftrightarrow \text{sinc}(f)$, then by duality $\text{sinc}(t) \Leftrightarrow \text{rect}(-f) = \text{rect}(f)$ (because $\text{rect}(\cdot)$ is an even function).

Energy and power conservation (Plancherel / Parseval theorem). For energy signals:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |X(f)|^2 df. \tag{A.47}$$

For periodic (power) signals with fundamental period T_0:

$$\mathcal{P} = \frac{1}{T_0} \int_{<T_0>} |x(t)|^2 dt = \sum_{k=-\infty}^{\infty} |c_k|^2, \tag{A.48}$$

where c_k are the coefficients of the Fourier series of $x(t)$.

Autocorrelation (Wiener-Khinchin theorem):

$$R_x(\tau) = \int_{-\infty}^{\infty} x(t+\tau)x^*(t)dt \Leftrightarrow X^*(f)X(f) = |X(f)|^2. \tag{A.49}$$

A.18 Fourier Analysis: Pairs

This section lists few pairs, which are among the most important ones. Both continuous and discrete-time signals are exemplified.

1. impulse $\Leftrightarrow$ DC level

$$x[n] = \delta[n] \Leftrightarrow X(e^{j\Omega}) = 1$$

$$x[n] = \frac{1}{2\pi} \Leftrightarrow X(e^{j\Omega}) = \sum_{k=-\infty}^{\infty} \delta(\Omega - k2\pi)$$

2. train of impulses $\Leftrightarrow$ train of impulses

   For the continuous-time, a train of unitary impulses spaced by T_s leads to a train of impulses spaced by $1/T_s$ with areas $1/T_s$:

$$\sum_{k=-\infty}^{\infty} \delta(t - kT_s) \Leftrightarrow \frac{1}{T_s} \sum_{k=-\infty}^{\infty} \delta\left(f - \frac{k}{T_s}\right) \tag{A.50}$$

   For the discrete-time, a train of (discrete-time) impulses with amplitude one and spaced by N_0 leads to a train of impulses spaced by $2\pi/N_0$ with areas $2\pi/N_0$:

$$\sum_{k=-\infty}^{\infty} \delta[n - kN_0] \Leftrightarrow \frac{2\pi}{N_0} \sum_{k=-\infty}^{\infty} \delta(\Omega - k\frac{2\pi}{N_0})$$

3. pulse $\Leftrightarrow$ sinc

   Several pairs of pulses and sincs are described below.

$$x(t) = \begin{cases} A, & -T/2 \leq t \leq T/2 \\ 0, & \text{otherwise} \end{cases} \Leftrightarrow X(f) = AT\text{sinc}(fT) \tag{A.51}$$

$$x[n] = \begin{cases} 1, & 0 \leq n \leq M-1 \\ 0, & \text{otherwise} \end{cases} \Leftrightarrow X(e^{j\Omega}) = \frac{\sin(\Omega M/2)}{\sin(\Omega/2)} e^{-j\Omega(M-1)/2}$$

   Due to the duality property, sincs in time-domain lead to pulses in frequency-domain. Considering discrete-time signals and assuming $\alpha \geq 1$ determines the spectrum bandwidth (recall that it suffices to specify $X(e^{j\Omega})$ for $-\pi \leq \Omega \leq \pi$) one has:

$$x[n] = \frac{2\pi}{\alpha}\text{sinc}\left(\frac{n}{\alpha}\right) \Leftrightarrow X(e^{j\Omega}) = \begin{cases} 1, & -\pi/\alpha \leq \Omega \leq \pi/\alpha \\ 0, & \text{otherwise} \end{cases} \tag{A.52}$$

   Now it is assumed a discrete-time pulse train $x[n]$ with period N and, for the pulse centered in 0, $x[n] = 1$ from $n = -M$ to $N = M$ and 0 otherwise. This pulse is replicated: the next is centered in $n = N$ and has non-zero values in the range $[N - M, N + M]$ and so on. The spectrum is

$$X[k] = \frac{1}{N} \frac{\sin\left(k(2M+1)\frac{\pi}{N}\right)}{\sin\left(k\frac{\pi}{N}\right)},$$

   where $X[k] = \frac{2M+1}{N}, k = 0, \pm N, \pm 2N, \ldots$, via L'Hopital's rule.

   For a continuous-time pulse train with each pulse of duration $2T_p$ and period T_0

(one pulse is centered at $t = 0$, with duration from $-T_p$ to T_p) one has:

$$c_k = \frac{2\sin(2\pi k f_0 T_p)}{T_0 2\pi k f_0} = \frac{2\sin(k\omega_0 T_p)}{T_0 k\omega_0},$$

where $f_0 = 1/T_0$ and $\omega_0 = 2\pi f_0$.

4. exponential $\Leftrightarrow$ rational function

$$e^{-at}u(t) \Leftrightarrow \frac{1}{(j\omega + a)}, a > 0$$

$$a^n u[n] \Leftrightarrow \frac{1}{(1 - ae^{-j\Omega})}, |a| < 1$$

5. complex exponential $\Leftrightarrow$ impulse

$$e^{j2\pi f_0 t} \Leftrightarrow \delta(f - f_0) \tag{A.53}$$

6. impulse train $\Leftrightarrow$ impulse train (or sum of complex exponentials)

   The transform of an impulse train of unitary area and period T_{sym} can be written as another impulse train or as a sum of complex exponentials:

$$\sum_{l=-\infty}^{\infty} \delta(t - lT) \Leftrightarrow \frac{1}{T} \sum_{k=-\infty}^{\infty} \delta(f - \frac{k}{T}) = \sum_{l=-\infty}^{\infty} e^{-j2\pi flT}. \tag{A.54}$$

   The equality in Eq. (A.54) is not trivial. The interested reader can use the code MatlabOctaveCodeSnippets/snip_appfourier_impulse_train.m to observe it via plots.

A.19 Probability and Stochastic Processes

A.19.1 Random variables

Understanding random variables require some knowledge of probability theory. There are many good textbooks about probability. The Web is also very useful [urlBM-pro]. The reader should study the concepts of conditional probability, Bayes rule and statistical independence.

A source of confusion is that a random variable (e.g., X or Y) is in fact a function. First note that the outcome of a probabilistic experiment need not be a number but any element of a set Ω of possible outcomes. For example, the outcome when a coin is tossed can be $\omega =$"heads" or $\omega =$"tails". A random variable (r.v.) is a function $\mathsf{X} : \Omega \to \mathbb{R}$ that associates a unique numerical value with every outcome of an experiment (r.v. can be complex numbers, vectors, etc., but here it will be assumed as a real number). In math, a function output is often represented as $y = f(x)$. Here, instead of adopting

something like $\mathsf{X}(\omega)$, both the random variable (equivalent to the function f) and its output value (equivalent to y) is represented by a single letter (e.g., X or Y).

There are two types of r.v.: discrete and continuous. Hence, a r.v. has either an associated probability distribution (discrete r.v.) or probability density function (continuous r.v.).

Assume a discrete r.v. X and a continuous r.v. Y. While the former is typically described by a probability mass function (pmf), the latter can be described by a probability density function (pdf).

Say that X represents the outcome of rolling a dice. Its PMF is shown in Figure A.7 and indicates that each face of a fair dice has a probability of 1/6.

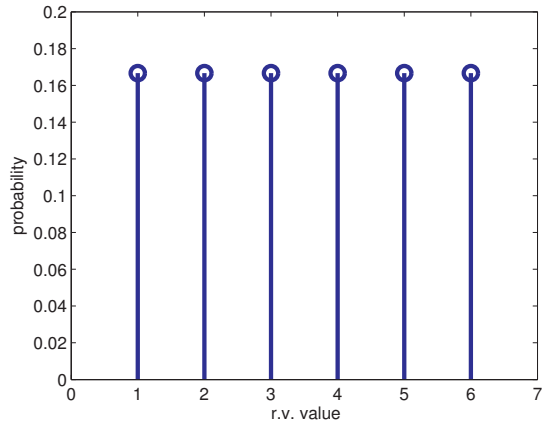


Figure A.7: *PMF for a dice result.*

Now consider that Y represents the amplitude of a Gaussian noise source with mean 2 and variance equal to 1. Its pdf is shown in Figure A.8. A common mistake is to assign a non-zero value to a specific value of a density function. For example, it is wrong to say that the probability of $\mathsf{Y} = 2$ is 0.4, in spite of this being the value of the function. The function represents a *density*, and the correct answer is that the probability of $\mathsf{Y} = 2$, or any other point, is 0. One can extract probability from a pdf only integrating it over a non-zero range of its abscissa. For example, over the range [2, 3] the probability is approximately 0.34 as indicated by the shaded area in Figure A.8.

When dealing with ratios of pdfs it is possible to have the abscissa range Δ_x canceling out. For example, if a discrete binary r.v. is used to represent two classes (A and B, for example), and each class has a pdf associated to it ($f(\mathbf{x}|A)$ and $f(\mathbf{x}|B)$, respectively), the Bayes' rule states

$$P(A|\mathbf{x}) = \frac{f(\mathbf{x}|B)}{f(\mathbf{x}|A)} P(B|\mathbf{x}). \tag{A.55}$$

In this case, Δ_x cancels because it appears on both numerator and denominator.

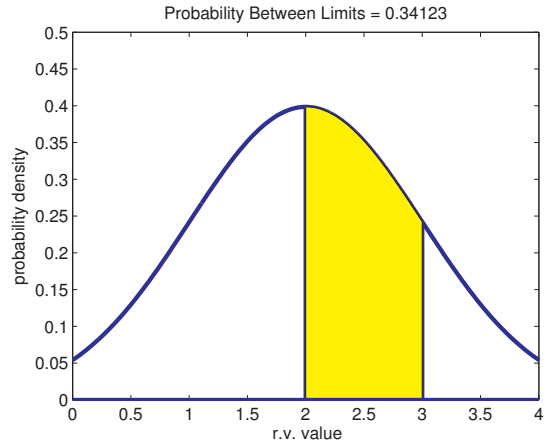


Figure A.8: *Obtaining probability from a pdf (density function) requires integrating over a range. Example shows that the probability of this Gaussian variable be within [2, 3] is approximately 0.34.*

A.19.2 Expected Values

The expected value $\mathbb{E}[\cdot]$ operator is the most common mathematical formalism for calculating an average (or mean). For example, if $\mathbf{x} = [3, 5, 4, 4, 5, 3]$ is a vector with random samples, as one could guess even without knowing the definition of $\mathbb{E}[\cdot]$, its expected value is $\mathbb{E}[\mathbf{x}] = (3 + 5 + 4 + 4 + 5 + 3)/6 = 4$, as can be calculated as a typical average.

A.19.3 Orthogonal versus uncorrelated

Two random variables X and Y are said to be *orthogonal* to each other if

$$\mathbb{E}[\mathsf{X}\mathsf{Y}^*] = 0.$$

They are said to be uncorrelated with each other if

$$\mathbb{E}[(\mathsf{X} - \mathbb{E}[\mathsf{X}])(\mathsf{Y} - \mathbb{E}[\mathsf{Y}])^*] = 0.$$

The above condition is equivalent to

$$\mathbb{E}[\mathsf{X}\mathsf{Y}^*] = \mathbb{E}[\mathsf{X}]\mathbb{E}[\mathsf{Y}]^*.$$

Note that if one or both of X and Y have zero mean, then the orthogonal and uncorrelated conditions are equivalent.

A.19.4 PDF of a sum of two independent random variables

If X and Y are independent, $\mathsf{Z} = \mathsf{X} + \mathsf{Y}$ implies $f(z) = f(x) * f(y)$. For example, the sum of a bipolar signal (-5 and +5V with 0.5 of probability each) with pdf $f_x(x) =$

$0.5[\delta(x-5)+\delta(x+5)]$ and additive white Gausssian noise (AWGN) with pdf $f_y(y)$ is a good example because the result is the Gaussian scaled by 0.5 and shifted to the position of each of the original impulses, at -5 and 5, i. e., $f_z(z) = 0.5[f_y(z-5) + f_y(z+5)]$.

A.19.5 Stochastic (random) processes

Stochastic or *random* processes are a powerful formalism for studying signals that incorporate some randomness. While the outcome of a random variable is a number, the outcome of a stochastic process is a time-series, each one called a *realization*. Hence, when using stochastic processes, it is possible to calculate "time" statistics of a single realization. Besides, one can choose a given time instant and calculate "ensemble" statistics over all realizations at that specific time. These two options may be confusing at first, but are essential. To get intuition on them, the reader is invited to observe the following simple example.

Example A.2. **Simple example to explain time and ensemble averages in a stochastic process**. Assume a vector [2, 0, 2, 3, 3, 2, 3] where each element represents the outcome of a random variable X indicating the number of victories of a football team in four matches (one per week), along a seven-months season (for example, the team won 2 games in the first month and none in the second). One can calculate the average number of victories as $\mathbb{E}[\mathsf{X}] \approx 2.14$, the standard deviation $\sigma \approx 1.07$ and other moments of a random variable. This vector could be modeled as a finite-duration random signal $\mathsf{x}[n]$, with Figure A.9 depicting its graph.

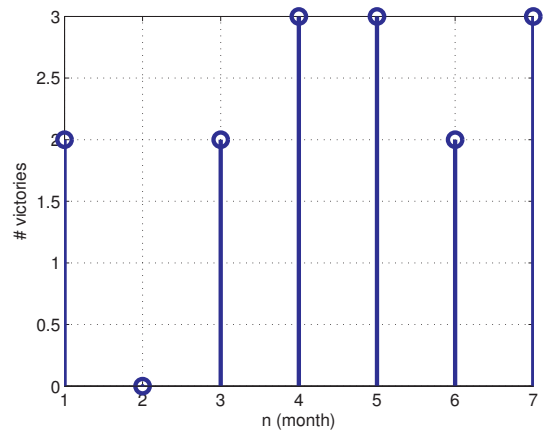


Figure A.9: *Example of a finite-duration random signal.*

Assume now there are 5 vectors, one for each year. A discrete-time random process can be used to model the data, with each vector (a time series) corresponding to a *realization* of the random process. For example, following the Matlab convention of organizing different time series in columns (instead of rows), the tabular data below represents the five random signals:

```
victories = [
```

```
2     3     0     0     1
0     1     3     2     1
2     1     4     0     2
3     4     3     1     2
3     3     4     3     4
2     1     1     2     3
3     4     4     2     4]
```

Each random signal (column) can be modeled as a realization of a random process. Figure A.10 shows a graphical representation of the data with the five realizations of the example. One should keep in mind that a random process is a model for generating an infinite number of realizations but only 5 are used here for simplicity.

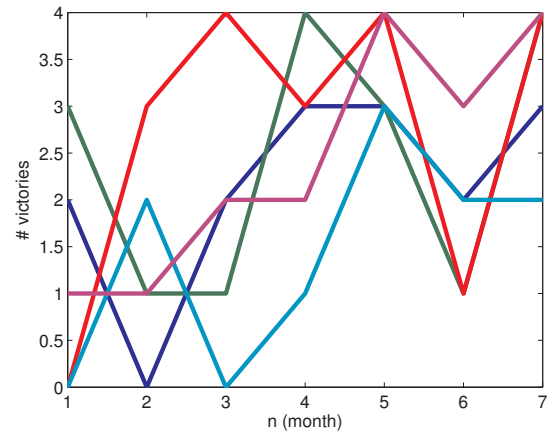


Figure A.10: *Example of five realizations of a discrete-time random process.*

A discrete-time random process will be denoted by $\mathcal{X}[n]$ and one of its realization as $\mathsf{x}[n]$. For the sake of explaining this example, let us define the notation $\mathsf{x}_i[n]$ to indicate the i-th realization of $\mathcal{X}[n]$. Because $\mathsf{x}_i[n]$ is a random signal, one can calculate for the given example, $\mathbb{E}[\mathsf{x}_1[n]] \approx 2.14$ and other statistics taken over time. But a random process is a richer model than a random signal and other statistics can be calculated or, equivalently, more questions can be asked.

Besides asking for statistics of one realization taken over "time", it is important to understand that when fixing a given time instant n_0, the values that the realizations assume are also outcomes of a random variable that can be denoted by $\mathcal{X}[n_0]$. In the example, assuming $n_0 = 4$ (i.e., assume the fourth month), one can estimate the average $\mathbb{E}[\mathcal{X}[4]] = 2.6$. Similarly, $\mathbb{E}[\mathcal{X}[6]] = 1.8$. Figure A.11 graphically illustrates the operation of evaluating a random process at these two time instants. Differently than the previous time averages, these "ensemble" statistics are obtained across realizations with fixed time instants.

The continuous-time random process $\mathcal{X}(t)$ is similar to its discrete-time counterpart. Taking two different time values t and s leads to a pair of random variables, which can then be fully (statistically) characterized by a joint pdf function $f(\mathcal{X}(t), \mathcal{X}(s))$.

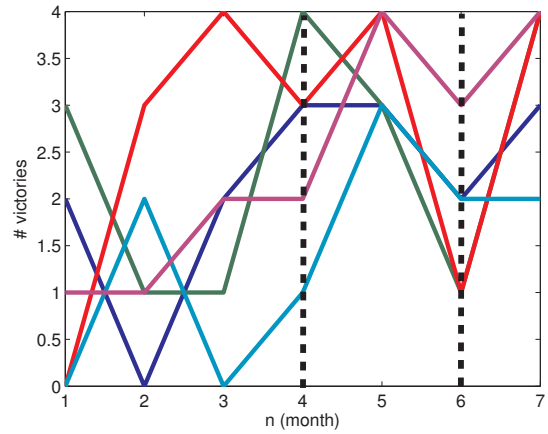


Figure A.11: *Example of evaluating a random process at time instants $n = 4$ and $n = 6$, which correspond to the values of two random variables $\mathcal{X}[4]$ and $\mathcal{X}[6]$. There are only five pairs and each occurs once, hence a 0.2 estimated probability for all points.*

This can be extended to three or more time instants (random variables). Back to the discrete-time example, one can use a normalized two-dimensional histogram to estimate the joint PMF of $\mathcal{X}[4]$ and $\mathcal{X}[6]$ as illustrated in Figure A.12.

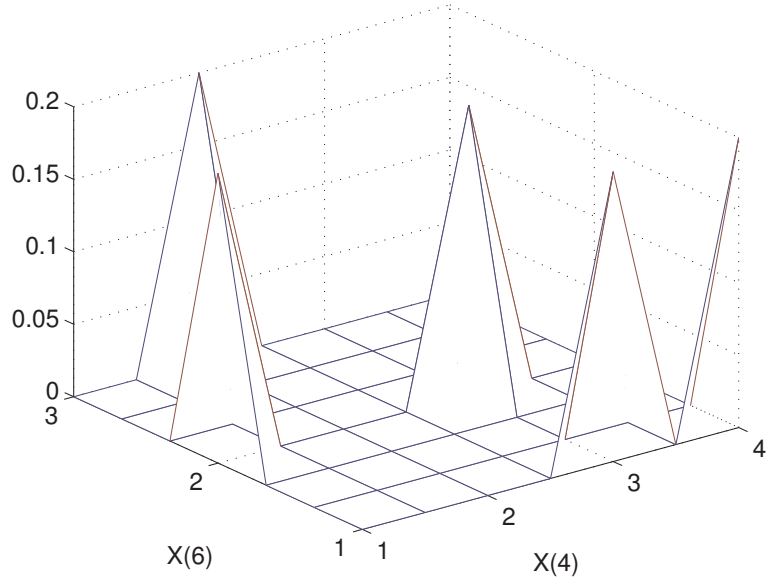


Figure A.12: *Example of a joint pdf of the continuous random variables $\mathcal{X}[4]$ and $\mathcal{X}[6]$.*

This example aimed at distinguishing time and ensemble averages, which is fundamental to understand stochastic processes. □

Correlation Function and Matrix

Second-order moments of a random process involve statistics taken at two distinct time instants. Two of them are the correlation and covariance functions, which are very useful in practice.

As discussed in Section 1.13, the correlation function is an extension of the correlation concept to random signals generated by stochastic processes. The *autocorrelation function* (ACF)

$$R_X(s,t) = \mathbb{E}[\mathcal{X}(s)\mathcal{X}(t)] \tag{A.56}$$

is the correlation between random variables $\mathcal{X}(s)$ and $\mathcal{X}(t)$ at two different points s and t in time of the same random process. Its purpose is to determine the strength of relationship between the values of the signal occurring at two different time instants. For simplicity, it is called $R_X(s,t)$ instead of $R_{XX}(s,t)$.

A discrete-time version of Eq. (A.56) is

$$R_X[n_1,n_2] = \mathbb{E}\left[\mathcal{X}[n_1]\mathcal{X}[n_2]\right], \tag{A.57}$$

where n_1 and n_2 are two "time" instants.

Specially for discrete-time processes, it is useful to organize autocorrelation values in the form of an autocorrelation matrix.

Example A.3. **Calculating and visualizing the autocorrelation matrix (not assuming stationarity)**. Stationarity is a concept that will be discussed in the sequel (Appendix A.19.5). Calculating and visualizing autocorrelation is simpler when the process is stationary. But in this example, this property is not assumed and the autocorrelation is obtained for a general process.

Listing A.6 illustrates the estimation of $R_X[n_1,n_2]$ for a general discrete-time random process.[4]

Listing A.6: *MatlabOctaveFunctions/ak_correlationEnsemble.m*

```
function [Rxx,n1,n2]=ak_correlationEnsemble(X, maxTime)
% function [Rxx,n1,n2]=ak_correlationEnsemble(X, maxTime)
%Estimate correlation for non-stationary random processes using
%ensemble averages (does not assume ergodicity).
%Inputs:  - X, matrix representing discrete-time process realizations
%           Each column is a realization of X.
%         - maxTime, maximum time instant starting from n=1, which
%           corresponds to first row X(1,:). Default: number of rows.
%Outputs: - Rxx is maxTime x maxtime and Rxx[n1,n2]=E[X[n1] X*[n2]]
%         - n1,n2, lags to properly plot the axes. The values of
%           n1,n2 are varied from 1 to maxTime.
[numSamples,numRealizations] = size(X); %get number of rows and col.
if nargin==1
    maxTime = numSamples; %assume maximum possible value
```

[4] Note that xcorr.m in Matlab/Octave assumes the process is wide sense stationary, while ak_correlationEnsemble.m does not.

```
end
Rxx = zeros(maxTime,maxTime); %pre-allocate space
for n1=1:maxTime
    for n2=1:maxTime %calculate ensemble averages
        Rxx(n1,n2)=mean(X(n1,:).*conj(X(n2,:))); %ensemble statistics
        %that depends on numRealizations stored in X
    end
```

Figure A.13 was created with the command ak_correlationEnsemble(victories) and shows the estimated correlation for the data in Figure A.10 of Example A.2. The data cursor indicates that $R_X[1,1] = 2.8$ because for $n = 1$ the random variables of the five realizations are $[2, 3, 0, 0, 1]$. In this case, $R_X[1,1] = \mathbb{E}[\mathcal{X}[1]^2] \approx (4 + 9 + 1)/5 = 2.8$. Similar calculation leads to $R_X[1,7] = \mathbb{E}[\mathcal{X}[1]\mathcal{X}[7]] \approx 4.4$ because in this case the products victories(1,:).*victories(7,:) are [6, 12, 0, 0, 4], which have an average of 4.4.

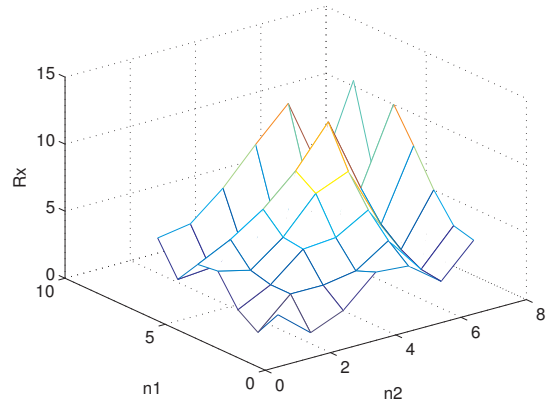


Figure A.13: *Correlation for data in matrix victories.*

Figure A.14 presents an alternative view of Figure A.13, where the values of the z-axis are indicated in color. The two datatips indicate that $R_x[1,2] = 0.8$ and $R_x[3,5] = 6.6$ (these are the values identified by Index for the adopted colormap, and the associated color is also shown via its RGB triple).

For real-valued processes, the autocorrelation is symmetric, as indicated in Figure A.13 and Figure A.14, while it would be Hermitian for complex-valued processes. □

Two time instants: both absolute or one absolute and a relative (lag)

When characterizing second-order moments of a random process, instead of using two absolute time instants, it is sometimes useful to make one instant as a relative "lag" with respect to the other. For example, take two absolute instants $t = 4$ and $s = 7$ in Eq. (A.56). Alternatively, the same pair of instants can be denoted by considering $t = 4$ as the absolute time that provides the reference instant and a lag $\tau = s - t = 3$ indicating that the second instant is separated by 3 from the reference.

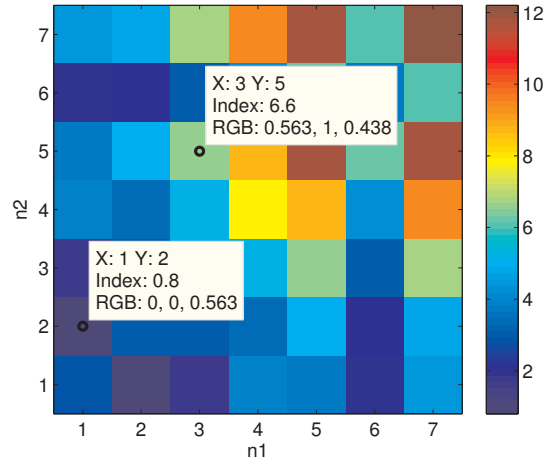


Figure A.14: *Version of Figure A.13 using an image.*

Using this scheme, Eq. (A.56) can be written as

$$R_X(t,\tau) = \mathbb{E}[\mathcal{X}(t+\tau)\mathcal{X}(t)]. \tag{A.58}$$

Similarly, Eq. (A.57) can be denoted as

$$R_X[n,l] = \mathbb{E}\left[\mathcal{X}[n]\mathcal{X}[n+l]\right], \tag{A.59}$$

with $n_1 = n$ and $n_2 = n + l$, where l is the lag in discrete-time.

Example A.4. **Contrasting two different representations of autocorrelations**. Figure A.13 is a graph that corresponds to Eq. (A.57), with two absolute time instants. When using the alternative representation of Eq. (A.59), with a lag, the resulting matrix is obtained by rearranging the elements of the autocorrelation matrix. Figure A.15 compares the two cases. Note that the name autocorrelation matrix is reserved for the one derived from Eq. (A.57) (and Eq. (A.56)) that is, for example, symmetric.

Figure A.16 is an alternative view of Figure A.15, which makes visualization easier. It better shows that Eq. (A.59) leads to shifting the column of the autocorrelation matrix (Figure A.13) corresponding to each value $n = n_1$ of interest.

Properties such as the symmetry of the autocorrelation matrix may not be so evident from the right-side representation of Figure A.16 as in the left-side representation, but both are useful in distinct situations. □

Covariance matrix

Similar to autocorrelation functions, it is possible to define covariance functions that expand the definition for random variables such as Eq. (1.33) and take as arguments two time instants of a random process. For example, for a complex-valued discrete-time

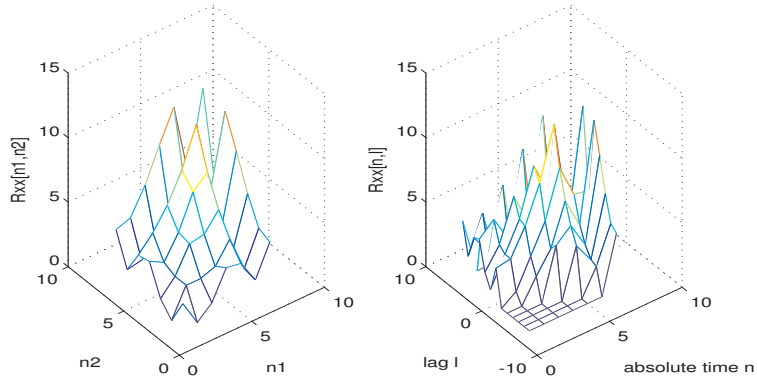


Figure A.15: *Comparison between the 3-d representation of the autocorrelation matrix in Figure A.13 and the one using lags as in Eq. (A.59).*

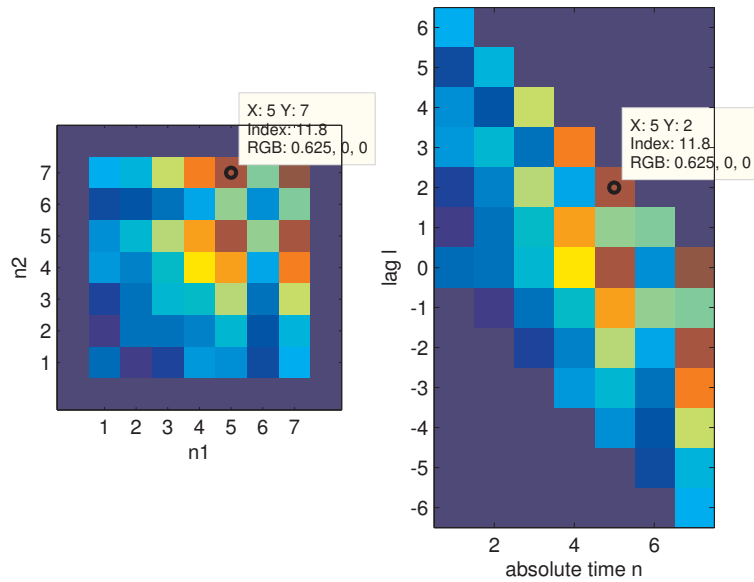


Figure A.16: *Comparison between autocorrelation representations using images instead of 3-d graphs as in Figure A.15.*

process the *covariance* is

$$c_X[n,l] = \mathbb{E}\left[(\mathcal{X}[n] - \mu[n])(\mathcal{X}[n+l] - \mu[n+l])^*\right]. \tag{A.60}$$

Stationarity, cyclostationarity and ergodicity

A random process is a powerful model. Hence, an important question is what information is necessary to fully characterize it. Take the 7×5 matrix in Example A.2: this data does not fully characterize the random process because it consists of only five realizations (tossing a coin five times does not allow to estimate its probability of heads robustly). We would need an infinite number of realizations if we insist in characterizing it in a

tabular form. The most convenient alternative is to describe it through pdfs. We should be able to indicate, for all time t, the pdf of $\mathcal{X}(t)$. Besides, for each time pair (t, s), the joint pdf $f(\mathcal{X}(t), \mathcal{X}(s))$ should be known. The same for the pdf $f(\mathcal{X}(t), \mathcal{X}(s), \mathcal{X}(r))$ of each triple (t, s, r) and so on. It is easy to see that a general random process requires a considerable amount of statistics to be fully described. So, it is natural to define particular and simpler cases of random processes as discussed in the sequel.

An important property of a random process is the stationarity of a statistics, which means this statistics is time-invariant even though the process is random. More specifically, a process is called *n-th order stationary* if the joint distribution of any set of n of its random variables is independent of absolute time values, depending only on relative time.

A first-order stationary process has the same pdf $f(\mathcal{X}(t)), \forall t$ (similar is valid for discrete-time). Consequently, it has a constant mean

$$\mathbb{E}[\mathcal{X}(t)] = \mu, \forall t, \tag{A.61}$$

constant variance and other moments such as $\mathbb{E}[\mathcal{X}^4(t)]$.

A second-order stationary process has the same joint pdf $f(\mathcal{X}(t), \mathcal{X}(s)) = f(\mathcal{X}(t + \tau), \mathcal{X}(s + \tau)), \forall t, s$. Consequently, its autocorrelation is

$$R_X(\tau) = \mathbb{E}[\mathcal{X}(t + \tau)\mathcal{X}(t)], \tag{A.62}$$

where the time difference $\tau = s - t$ is called the *lag* and has the same unit as t in $\mathcal{X}(t)$. Note that, in general, $R_X(t, \tau)$ of Eq. (A.58) depends on two parameters: t and τ. However, for a stationary process, Eq. (A.62) depends only on the lag τ because, given τ, $R_X(t, \tau) = R_X(\tau)$ is the same for all values of t.

Similarly, if a discrete-time process is second-order stationary, $R_X[n, l]$ Eq. (A.59) can be simplified to

$$R_X[l] = \mathbb{E}[\mathcal{X}[n]\mathcal{X}[n + l]], \tag{A.63}$$

where $l = n_2 - n_1$ is the lag.

A third-order process has joint pdfs $f(\mathcal{X}(t_1), \mathcal{X}(t_2), \mathcal{X}(t_3))$ that do not depend on absolute time values (t_1, t_2, t_3) and so on. A random process $\mathcal{X}(t)$ is said to be *strict-sense stationary* (SSS) or simply *stationary* if any joint pdf $f(\mathcal{X}(t_1), \ldots, \mathcal{X}(t_n))$ is invariant to a translation by a delay τ. A random process with realizations that consist of i. i. d. random variables (called an i. i. d. process) is SSS. A random process that is not SSS is referred to as a *non-stationary* process even if some of its statistics have the stationarity property.

The SSS process has very stringent requirements, which are hard to verify. Hence, many real-world signals are modeled as having a weaker form of stationarity called *wide-sense stationarity*.

Basically, a wide-sense stationary (WSS) process has a mean that does not vary over time and an autocorrelation that does not depend on absolute time, as indicated in Eq. (A.61) and Eq. (A.62), respectively. Broadly speaking, wide-sense theory deals with moments (mean, variance, etc.) while strict-sense deals with probability distributions.

Note that SSS implies WSS but the converse is not true in general, with Gaussian processes being a famous exception.

For example, the process corresponding to Figure A.13 is not WSS: its autocorrelation does not depend only on the time difference τ.

Example A.5. **The autocorrelation matrix of a WSS process is Toeplitz**. For a WSS process, elements $R_X(i,j)$ of its autocorrelation matrix depend only on the absolute difference $|i-j|$ and, therefore, the matrix is Toeplitz. In this example, the function toeplitz.m is used to visualize the corresponding autocorrelation matrix as an image. For example, the command Rx=toeplitz(1:4) generates the following real-valued matrix:

```
Rx=[1     2     3     4
    2     1     2     3
    3     2     1     2
    4     3     2     1].
```

Usually, the elements of the main diagonal correspond to $R_X(0) = \mathbb{E}[\mathcal{X}^2[n]]$.

To visualize a larger matrix, Figure A.17 was generated with Rxx=toeplitz(14:-1:4).

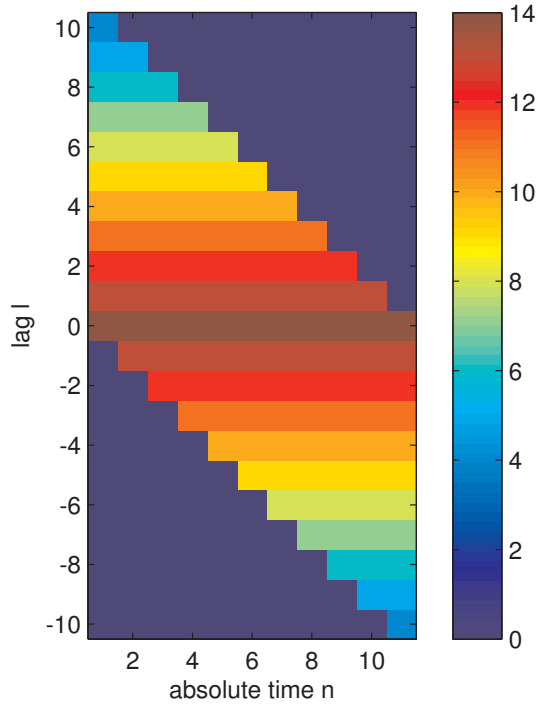


Figure A.17: *Representation of a WSS autocorrelation matrix that depends only on the lag l.*

Given the redundancy in Figure A.17, one can observe that for a WSS, it suffices to describe $R_X(l)$ for each value of the lag l. □

A *cyclostationary* process is a non-stationary process having statistical properties that vary periodically (cyclically) with time. In some sense it is a weaker manifestation of a stationarity property. Cyclostationary processes are further discussed in Appendix A.19.8.

A given statistics (variance, autocorrelation, etc.) of a random process is *ergodic* if its time average is equal to the ensemble average. For example, an i. i. d. random process is *ergodic* in the mean.[5] And, loosely speaking, a WSS is ergodic in both mean and autocorrelation if its autocovariance decays to zero.[6]

A process could be called "ergodic" if all ensemble and time averages are interchangeable. However, it is more pedagogical to consider ergodicity as a property of specific statistics, and always inform them.[7] Even non-stationary or cyclostationary processes can be ergodic in specific statistics such as the mean. Figure A.18 depicts the suggested taxonomy of random processes using a Venn diagram.

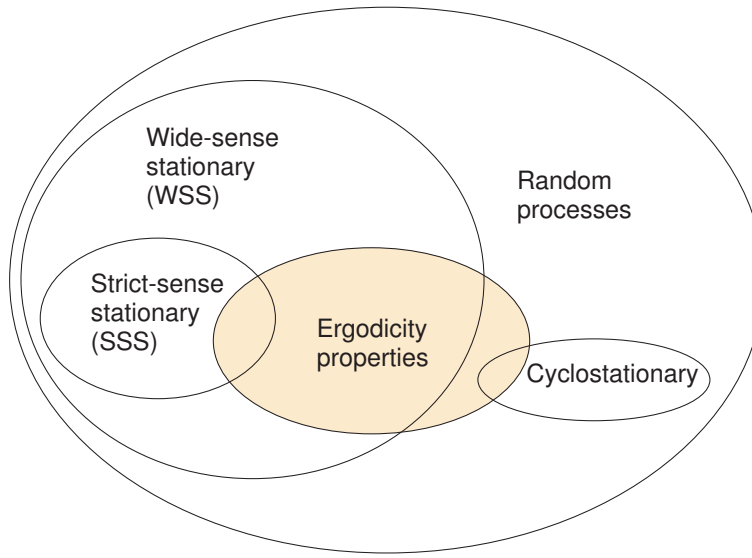


Figure A.18: *Suggested taxonomy of random processes. SSS are sometimes called stationary and all processes that are not SSS are called non-stationary.*

The following example discusses WSS processes using bit streams from Section **??**.

Example A.6. **Polar and unipolar bitstreams modeled as WSS processes**. This example is based on processes used in digital communications to represent sequences of bits. Note that, after these bit streams are upsampled in order to create line codes, they become cyclostationary processes. This issue is discussed in Appendix A.19.8, while here the sequences are not upsampled.

[5] See, e. g., [Kay06], pg. 564.
[6] See [urlBMerm] for the requirement of a WSS to be ergodic in the mean and/or [Kay06], pg. 577.
[7] See [urlBMerg] for an example of a SSS process that is not ergodic. The discussion also illustrates how it is confusing when the statistics that is ergodic is not specified and the process is called "ergodic".

Figure A.19 illustrates two processes for which the main diagonal $R_X[1,1] = R_X[2,2] = R_X[3,3]\ldots$ has the same values. The process (a) (left-most) is called a *unipolar* line code and consists of sequences with equiprobable values 0 and 1, while for the second process (called *polar*) the values are -1 and 1.

For the unipolar and polar cases, the theoretical correlations $R_X[n_1, n_2]$ for $n_1 = n_2$ are 0.5 and 1, respectively. The result is obtained by observing that $R_X[n_1, n_2] = R_X[n_1, n_1] = \mathbb{E}[\mathcal{X}[n_1]^2]$ and for the unipolar case, $\mathcal{X}[n_1]^2$ is 0 or 1. Because the two values are equiprobable, $\mathbb{E}[\mathcal{X}[n_1]^2] = 0.5$. For the polar case, $\mathcal{X}[n_1]^2 = 1$ always and $\mathbb{E}[\mathcal{X}[n_1]^2] = 1$. The values of $R_X[n_1, n_2]$ for $n_1 \neq n_2$ are obtained by observing the values of all possible products $\mathcal{X}[n_1]\mathcal{X}[n_2]$. For the unipolar case, the possibilities are $0\times0 = 0$, $0\times1 = 0$, $1\times0 = 0$ and $1\times1 = 1$, all with probability 1/4 each. Hence, for this unipolar example, $\mathbb{E}[\mathcal{X}[n_1]\mathcal{X}[n_2]] = (3\times0 + 1\times1)/4 = 0.25, n1 \neq n2$. For the polar case, the possibilities are $-1\times-1 = 1$, $-1\times1 = 1$, $1\times-1 = -1$ and $1\times1 = 1$, all with probability 1/4 each. Hence, for this polar example, $\mathbb{E}[\mathcal{X}[n_1]\mathcal{X}[n_2]] = (2\times-1+2\times1)/4 = 0, n1 \neq n2$.

In summary, for the polar code $R_X[n_1, n_2] = 1$ for $n_1 = n_2$ and 0 otherwise. For the unipolar code, $R_X[n_1, n_2] = 0.5$ for $n_1 = n_2$ and 0.25 otherwise.

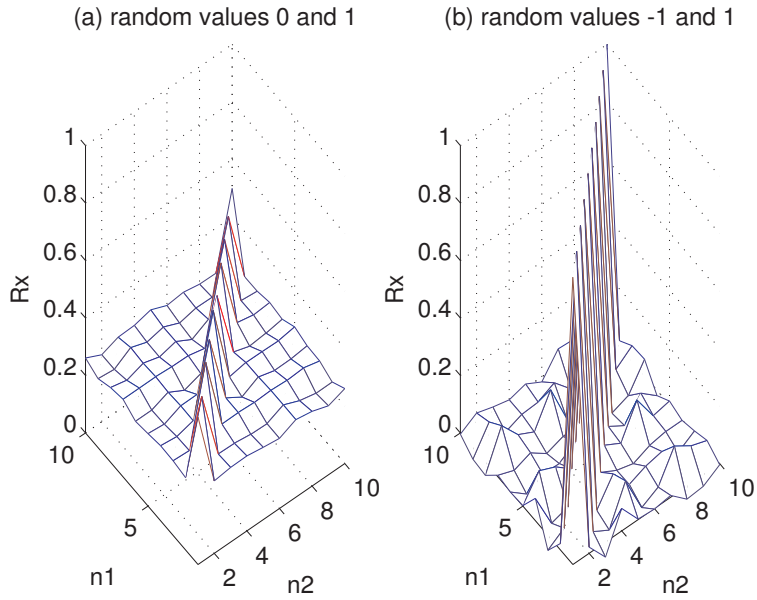


Figure A.19: *Correlation for random sequences with two equiprobable values: a) values are 0 and 1 (unipolar), b) values are* -1 *and 1 (polar).*

A careful observation of Figure A.19 indicates that two dimensions are not necessary and it suffices to describe $R_X[l]$ as in Eq. (A.59). As discussed previously, in the case of WSS processes, the correlation depends only on the difference between the "time" instants n_1 and n_2. Figure A.20 emphasizes this aspect using another representation for the polar case in Figure A.19.

Because the polar and unipolar processes that were used to derive Figure A.19 are

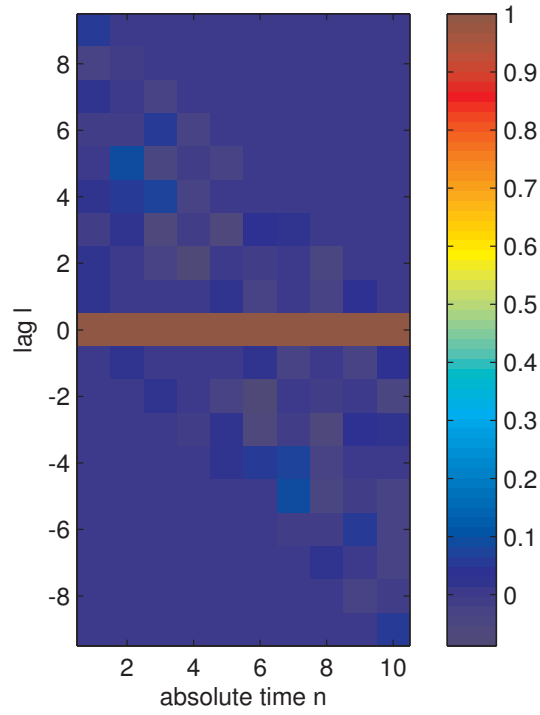


Figure A.20: *Alternative representation of the correlation values for the polar case of Figure A.19.*

WSS, Listing A.7 can be used to take advantage of having $R_X[n_1, n_2]$ depending only on the lag and convert this matrix to an array $R_X[l]$.

Listing A.7: *MatlabOctaveFunctions/ak_convertToWSSCorrelation.m*

```
function [Rxx_tau, lags] = ak_convertToWSSCorrelation(Rxx)
% function [Rxx_tau,lags]=ak_convertToWSSCorrelation(Rxx)
%Convert Rxx[n1,n2] into Rxx[k], assuming process is
%wide-sense stationary (WSS).
%Input: Rxx -> matrix with Rxx[n1,n2]
%Outputs: Rxx_tau[lag] and lag=n2-n1.
[M,N]=size(Rxx); %M is assumed to be equal to N
lags=-N:N; %lags
numLags = length(lags); %number of lags
Rxx_tau = zeros(1,numLags); %pre-allocate space
for k=1:numLags;
    lag = lags(k); counter = 0; %initialize
    for n1=1:N
        n2=n1+lag; %from: lag = n2 - n1
        if n2<1 || n2>N %check
            continue; %skip if out of range
        end
        Rxx_tau(k) = Rxx_tau(k) + Rxx(n1,n2); %update
        counter = counter + 1; %update
```

```
    end
    if counter ~= 0
        Rxx_tau(k) = Rxx_tau(k)/counter; %normalize
    end
end
```

Figure A.21 was generated using Listing A.7 for the data in Figure A.19.

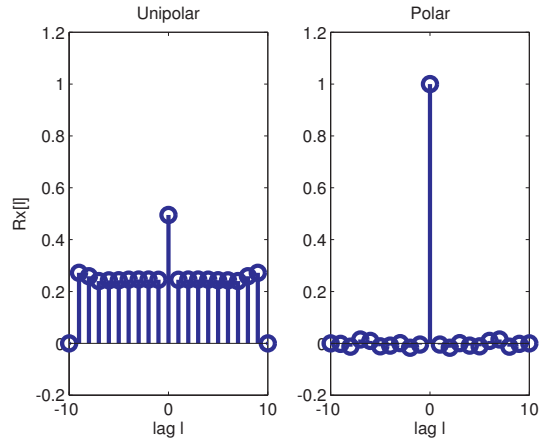


Figure A.21: *One-dimensional ACF $R_X[l]$ for the data corresponding to Figure A.19 (unipolar and polar codes).*

If ergodicity can be assumed, a single realization of the process can be used to estimate the ACF. Figure A.22 was generated using the code below, which generates a single realization of each process:

```
numSamples=1000;
Xunipolar=floor(2*rand(1,numSamples)); %r.v. 0 or 1
[Rxx,lags]=xcorr(Xunipolar,9,'unbiased');%Rxx for unipolar
subplot(121); stem(lags,Rxx) title('Unipolar');
Xpolar=2*[floor(2*rand(1,numSamples))-0.5]; %r.v. -1 or 1
[Rxx,lags]=xcorr(Xpolar,9,'unbiased'); %Rxx for polar code
subplot(122); stem(lags,Rxx) title('Polar');
```

Increasing the number of samples to numSamples=1e6 achieves estimation errors smaller than in Figure A.21.

The polar and unipolar processes discussed here are used to model line codes in Section **??**. □

A.19.6 Proper and circular complex-valued random variables

In many applications, complex random signals are assumed to be proper or circular. These assumptions are convenient and widely adopted because they simplify computations. These two properties are briefly discussed here. More information can be found e. g. in [NM93, SS10, ASS11, PS07].

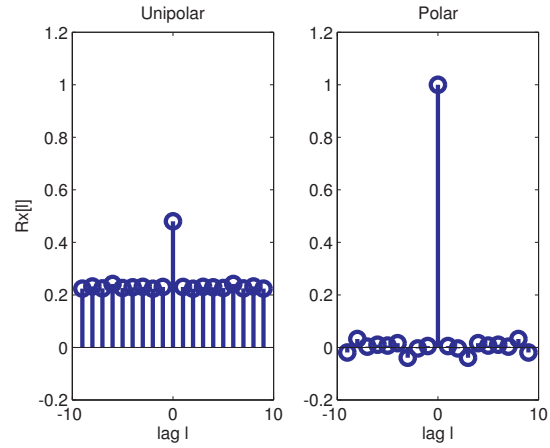


Figure A.22: *ACF estimated using ergodicity and waveforms with 1,000 samples for each process. The estimation error in this case is larger than in Figure A.21.*

A complex random variable is *proper* if its real and imaginary parts have equal variances and are uncorrelated. For example, QAM symbols such as the ones from constellations depicted in Figure **??** have uncorrelated real and imaginary components but their variances may differ as for $b = 3$. In the other cases of Figure **??** ($b = 5, 7$ and 9), the symbols are proper random variables. As another example, the code:

```
b=3; M=2^b; x=ak_qamSquareConstellation(M); cov(imag(x),real(x))
b=4; M=2^b; x=ak_qamSquareConstellation(M); cov(imag(x),real(x))
```

shows examples of improper ($b = 3$ bits) and proper ($b = 4$) constellations. In fact, all square QAM constellations with an even number of bits have the same variance per dimension and, consequently, are proper.

If X is proper and zero-mean, then $\mathbb{E}[\mathsf{X}^2] = 0$ because, denoting $\mathsf{X} = \mathsf{X}_r + j\mathsf{X}_i$:

$$\mathbb{E}[\mathsf{X}^2] = \mathbb{E}[(\mathsf{X}_r + j\mathsf{X}_i)^2] = \mathbb{E}[\mathsf{X}_r^2] - \mathbb{E}[\mathsf{X}_i^2] + 2j\mathbb{E}[\mathsf{X}_r\mathsf{X}_i] = 0. \tag{A.64}$$

The concept of a proper random variable can be extended to random vectors and random processes as discussed in [NM93]. In fact, when fully characterizing statistical moments of two complex random variables X and Y, one has to combine their real and imaginary parts. For example, it is possible to define four covariances. Hence, when dealing with complex-valued X and Y, besides the conventional covariance defined in Eq. (1.33), the pseudo-covariance

$$\text{pseudo-cov}(\mathsf{X}, \mathsf{Y}) = \mathbb{E}[(\mathsf{X} - \mu_x)(\mathsf{Y} - \mu_y)],$$

which does not use the complex conjugate of the second parcel as in Eq. (1.33), brings complementary information. For example (see [NM93] for details):

- when $\text{pseudo-cov}(\mathsf{X}, \mathsf{Y}) = 0$, the complex random variable or process is proper;
- in order to have X and Y uncorrelated (all four combinations of real and imaginary

parts), all non-diagonal elements of both conventional and pseudo covariances need to be zero.

Another interesting property of some complex random variables is circularity. A complex random variable X is *circular* (also called *circularly symmetric*) if its probability distribution is invariant under rotation in the complex plane, i. e., if X and $\mathsf{X}e^{j\theta}$ have the same pdf for any angle θ. For example, a complex-valued AWGN as the one generated in Listing **??** is circular. Figure **??** motivates the term circularly-symmetric, given the shape of its two-dimensional pdf estimation (visualized via the estimated histogram).

If any random variable is circular, then it is both proper and zero-mean. In the specific case of a Gaussian random variable or processes, being proper and zero-mean is equivalent to being circular. Also, when a circular Gaussian process is filtered by a linear (not necessarily time-invariant) system, the output is also a circular Gaussian process. The properties of being circular and proper are widely used to model complex-valued AWGN in telecommunications [PS07] and when they are not observed the algorithms that aim at optimality become more sophisticated [NM93, ASS11].

Digital modulation schemes produce noncircular complex baseband signals, since the system constellation are not rotationally invariant to an arbitrary angle θ. But whether or not the constellation is rotationally invariant to a discrete set of rotation angles leads to interesting properties, as discussed in the sequel.

A.19.7 Rotationally symmetric signal constellations

Constellations such as M-PSK, square QAM and cross QAM (see Figure **??**) are rotationally symmetric to specific angles [MdJ94]. Assuming equiprobable symbols, their pdfs do not change for these angles. For example, if X is a symbol from a M-PSK constellation, $\mathsf{X}e^{j\frac{2\pi}{M}}$ has the same pdf as X. If X is a (square or cross) QAM symbol, it has the same pdf as $\mathsf{X}e^{j\frac{\pi}{2}}$. Without a pilot tone or known training sequence, an arbitrary phase rotation cannot be identified from such rotationally symmetric constellations.[8] Hence, using nonequiproblabe systems or not zero-mean has been proposed to enable sensible blind statistics of the channel outputs [TMK00].

Related to this issue is the distinction between "circular" and "noncircular" constellations, as they are called in [CLSG02, CV03]. The "noncircular" (or *unbalanced*) constellations are characterized by $\mathbb{E}[\mathsf{X}^2] \neq 0$ where X is the random constellation symbol. One example is a PAM constellation. This nomenclature is not adopted here to avoid confusion with the concept of circularity just discussed in Appendix A.19.6. Instead, as in [Fra80], the "circular" is called here *balanced* constellation.

Note that the average constellation energy $\mathbb{E}[|\mathsf{X}|^2] \neq 0$ is non-zero, but many constellations are proper and zero-mean, such that, from Eq. (A.64), they are balanced. An application that exemplifies the importance of $\mathbb{E}[\mathsf{X}^2]$ is blind estimation of frequency offset. If the constellation is unbalanced, a cyclic frequency occurs when the signal is squared and can be used to estimate the frequency offset. This technique is discussed in [CLSG02, CV03] and exemplified in Listing **??**.

[8] In the case, of PAM, a rotation by $e^{j\pi}$ does not change the pdf but knowing the original symbols are real can be used in blind processing algorithms.

A.19.8 Cyclostationary random processes

This section aims at providing some intuition about cyclostationarity. More information can be found e. g. in [Gia99, Ant07, Gar91].

A discrete-time process $\mathcal{X}[n]$ is wide-sense cyclostationary (WSC) if, and only if, exists an integer period $P > 0$ such that the mean and correlation are periodic (consequently, the covariance is periodic too), as summarized by:

$$\begin{aligned} \mu_X[n] &= \mu_X[n+P] \quad \text{and} \\ R_X[n,l] &= R_X[n+P,l]. \end{aligned} \tag{A.65}$$

Note that the correlation is periodic but a cyclostationary process generates realizations that are random and not periodic. To avoid confusion, the period P in Eq. (A.65) of a WSC process is referred to as its *cycle*.

Because WSC are more common than strict-sense cyclostationary processes, the term *cyclostationary* is assumed here to correspond to a WSC (wide, not strict sense), as adopted in [Gia99], which is the main reference for the discussion presented in the sequel. In order to better understand WSC processes, it is useful to review the concepts of non-stationary processes in Appendix A.19.5 and get familiar with representations such as the ones in Figure A.15 and Figure A.16. While WSS processes have autocorrelations depending only on one parameter, the lag l, WSC correlations depend on two parameters ($R_X[n,l]$) and this is a key factor for the following discussion.

Note that Eq. (A.65) implies the correlation $R_X[n,l]$ in Eq. (A.59) is periodic in n for each "lag" l. In other words, given a lag value l_0, the time-series $R_X[n,l_0]$ over the "absolute time instant" n is periodic. Hence, it can be represented using a DTFS in which the Fourier coefficients $C_X[k,l]$ are called *cyclic* correlations. Therefore, when $R_X[n,l]$ has a period of P samples over n, the following Fourier pair can be used:

$$R_X[n,l] = \sum_{k=0}^{P-1} C_X[k,l] e^{j\frac{2\pi}{P}nk} \Leftrightarrow C_X[k,l] = \frac{1}{P}\sum_{n=0}^{P-1} R_X[k,l] e^{-j\frac{2\pi}{P}nk}. \tag{A.66}$$

$R_X[n,l]$ is also called *time-varying* correlation and *instantaneous* correlation.

As explained in [Gia99], sampling a continuous-time periodic signal may lead to an *almost periodic* signal for which Eq. (1.20) does not hold. This occurs in the so-called *wide-sense almost cyclostationary* processes, in which $R_X[n,l]$ has the following *generalized* Fourier series:

$$R_X[n,l] = \sum_{\alpha_k \in \mathcal{C}} C_X[\alpha_k,l] e^{j\alpha_k n} \Leftrightarrow C_X[\alpha_k,l] = \lim_{N\to\infty} \frac{1}{N}\sum_{n=0}^{P-1} R_X(k,l) e^{-j\alpha_k n}, \tag{A.67}$$

where $\mathcal{C}$ is the set $\mathcal{C} = \{\alpha_k : C_X[\alpha_k,l] \neq 0, -\pi \leq \alpha_k < \pi\}$ of cycles.

The estimation of both $C_X[k,l]$ and $C_X[\alpha_k,l]$ is often done with FFTs and, consequently, both periodic and almost periodic realizations are treated similarly. Hereafter, the notation $C_X[\alpha_k,l]$ is also adopted for $C_X[k,l]$.

For a given cycle α_k, the evolution of $C_X[\alpha_k, l]$ over the lags l can be interpreted over frequency using its DTFT to define the *cyclic spectrum*:

$$S_X(\alpha_k, \Omega) = \sum_{l=-\infty}^{\infty} C_X[\alpha_k, l] e^{-j\Omega l}. \tag{A.68}$$

In summary, while stationary discrete-time signals are analyzed using PSDs in frequency-domain (Ω) and time-invariant correlations in lag-domain (l), non-stationary signals include also the time (n) and cycle (α) domains. Figure A.23 illustrates the relations.

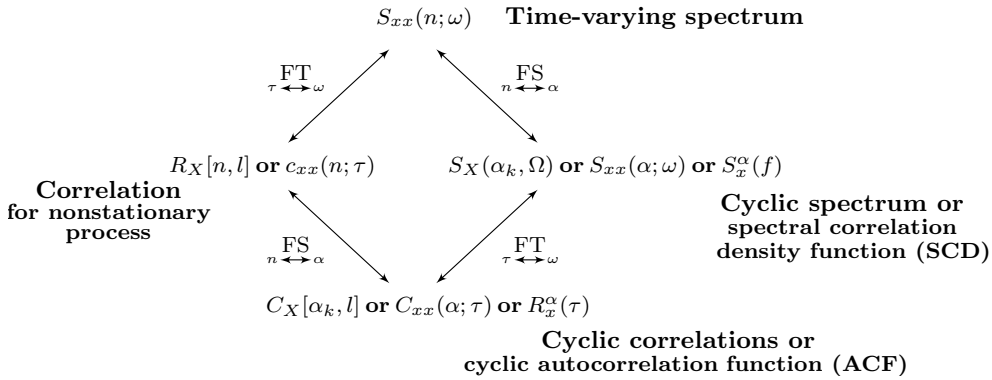


Figure A.23: *Functions used to characterize cyclostationary processes. FS stands for "generalized Fourier series", useful for representing "almost periodic" discrete-time signals (Adapted from Fig. 17.1 of [Gia99] to include the notation adopted in this text and the one widely used by [Gar91]).*

As further discussed in Section A.19.10, a cyclostationary process can be converted into a WSS through the "uniform shift" or "uniformly randomizing the phase" [Gia99].

The following example aims at making these concepts more concrete.

Example A.7. **Example: WGN modulated by sinusoid**. Assume $\nu[n]$ represents a realization of an i. i. d. white Gaussian noise process with variance σ^2 . Each realization is modulated by a carrier of frequency Ω_c and phase ϕ (both in radians) leading to[9]

$$x[n] = \nu[n] \cos(\Omega_c n + \phi). \tag{A.69}$$

The top plot in Figure A.24 illustrates a single realization of $x[n]$, as generated by Listing A.8. It can be seen that the sinusoid is "buried" in noise. All M=5000 realizations are stored in matrix X. The ensemble variance for each "time" instant n is shown in the bottom plot of Figure A.24, indicating that the variance has a period $P/2$ (in this case, 15 samples, given that P=30) because it depends on the absolute value of the sinusoid amplitude at instant n. This can be also seen from the correlation of $x[n]$ over time, as discussed in the sequel.

[9] See, e. g., Example 17.1 of [Gia99] and Example 1 of [Ant07] (Eq. (1)).

Listing A.8: *MatlabOctaveCodeSnippets/snip_appprobability_modulatednoise.m*

```
M=5000; %number of realizations for ensemble statistics
N=1000; %number of samples of each realization
P=30; %sinusoid period (in samples)
X=zeros(N,M); %pre-allocate space
Wc=2*pi/P; Phc=pi/5; %carrier frequency and phase, both in radians
n=0:N-1; carrier=transpose(cos(Wc*n+Phc)); %same for all realizations
%carrier(carrier>0)=1; carrier(carrier<0)=-1; %if wants square wave
for m=1:M %for each realization
    X(:,m)=randn(N,1).*carrier; %WGN modulated by sinusoidal carrier
end
```

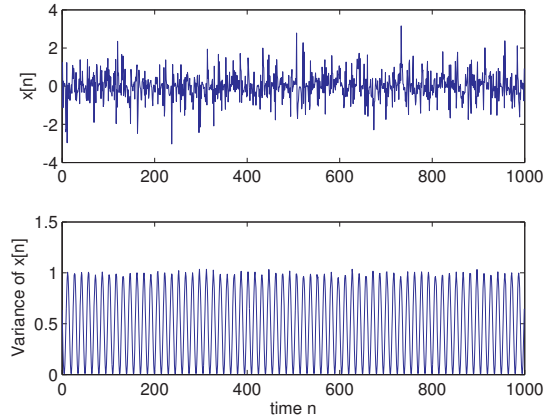


Figure A.24: *Single realization $x[n]$ of Eq. (A.69) (top) and the ensemble variance over time (bottom plot), which has a period of $P/2 = 15$ samples.*

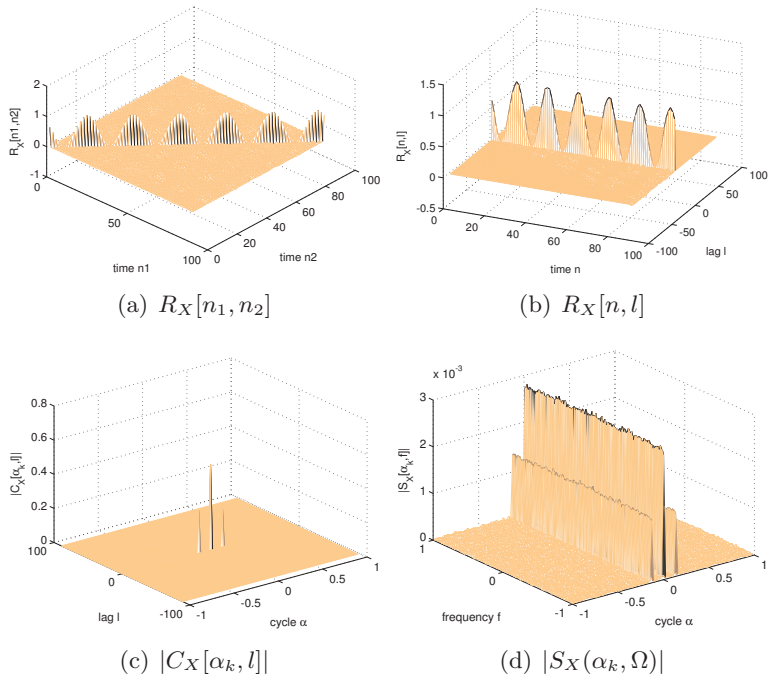
From Eq. (A.59), the correlation of $x[n]$ is

$$\begin{aligned} R_X[n,l] &= \mathbb{E}\left[\nu[n]\cos(\Omega_c n+\phi)\nu[n+l]\cos(\Omega_c(n+l)+\phi)\right] \\ &= \cos(\Omega_c n+\phi)\cos(\Omega_c(n+l)+\phi)\mathbb{E}\left[\nu[n]\nu[n+l]\right] = \cos^2(\Omega_c n+\phi)\sigma^2\delta[l] \\ &= \frac{\sigma^2}{2}\left(1+\cos(2\Omega_c n+2\phi)\right)\delta[l], \end{aligned} \tag{A.70}$$

which is depicted in Figure A.25(b). The value of $\delta[l]$ is 1 for all n when $l = 0$ and zero otherwise. Note that Figure A.25(a) is the alternative representation with a pair of absolute time instants $[n_1, n_2]$.

Eq. (A.70) is non-zero only at $l = 0$, where it is a squared cosine $\cos^2(\Omega_c n)\sigma^2$. Hence, from Eq. (A.8), its cyclic correlation is $C_X[\alpha_k, l] = \frac{\sigma^2}{2}\delta[\alpha] + \frac{\sigma^2}{4}(\delta[\alpha+2\Omega_c] + \delta[\alpha-2\Omega_c])$. And the cyclic spectrum is obtained by a DTFT for each cycle, which leads to the DC levels in Figure A.25(d).

Eq. (A.69) can be modified to $x[n] = \nu[n]p[n]$, where $p[n]$ is a generic periodic signal. In this case, $C_X[\alpha_k, l] = c_k\sigma^2\delta[l]$, where c_k is the Fourier coefficient of $|p[n]|^2$, as discussed in [Ant07].



(a) $R_X[n_1, n_2]$ (b) $R_X[n, l]$

(c) $|C_X[\alpha_k, l]|$ (d) $|S_X(\alpha_k, \Omega)|$

Figure A.25: *Cyclostationary analysis of the modulated white Gaussian noise..*

Listing A.9 indicates how the sinusoid frequency and phase can be obtained using the realizations in matrix X created by Listing A.8.

Listing A.9: *MatlabOctaveCodeSnippets/snip_appprobability_cyclo_analysis_ensemble.m*

```
snip_appprobability_modulatednoise %generate modulated WGN
alphaFFTLength=3*P; %P is the period, a multiple of P avoids leakage
[Cxx,alphas,lags] = ak_cyclicCorrelationEnsemble(X, alphaFFTLength);
onlyPositiveCycles=1; exclude0Alpha=1; %take only alpha>0 in account
[peakValue,peakAlpha,peakLag] = ak_peakOfCyclicCorrelation(Cxx, ...
    alphas,lags,exclude0Alpha,onlyPositiveCycles); %get the peak
WEstimated=peakAlpha/2; %estimated frequency in rad
phaseEstimated = angle(peakValue)/2; %phase in rad
phaseOffset=wrapToPi(Phc) - wrapToPi(phaseEstimated); %force [-pi,pi]
periodEstimated=round(2*pi/WEstimated); %estimate period (samples)
```

It should be noted that the phase estimation is highly sensitive to the value of alphaFFTLength (relatively, the frequency estimation is more robust). The reader is invited to modify this value and observe the estimation errors. As an example, the following output was obtained with Listing A.9:

```
Peak value=0.25187*exp(1.2545j)
Period:  Correct=30, Estimated=30 (samples)
Cycle:  Correct=0.20944, Estimated=0.20944(rad). Error=2.7756e-17
```

```
Phase:  Correct=0.62832, Estimated=0.62724(rad). Error=0.0010821
```

In this example the cyclic correlation was obtained with ensemble statistics. The following discussion concerns estimation using a single realization. □

When it is feasible to assume ergodicity in the autocorrelation of a WSC, the estimation can be done with a single realization of N samples. In this case and assuming a complex-valued $x[n]$ signal, an estimator for the cyclic correlation is:[10]

$$\hat{C}_X[\alpha_k, l] = \frac{1}{N} \sum_{n=0}^{N-l-1} x[n]x^*[n+l]e^{-j\alpha_k n}, \quad l \geq 0. \tag{A.71}$$

For negative l, one can use $\hat{C}_X[\alpha_k, -l] = \hat{C}_X^*[\alpha_k, l]e^{j\alpha_k l}$.

Example A.8. **Example: WGN modulated by sinusoid using a single realization**. Listing A.10 estimates the cyclic spectrum using a single realization of Example A.7.

Listing A.10: *MatlabOctaveCodeSnippets/snip_appprobability_cyclo_analysis.m*

```
snip_appprobability_modulatednoise %generate modulated WGN
x=X(:,1); %take one realization
maxTau=3*P; %maximum lag value (P is the sinusoid period)
numCycles=4*P; %the cycle resolution is then 2*pi/numCycles
[Cxx,alphas,lags]=ak_cyclicCorrelation(x,maxTau,numCycles);
```

The output of Listing A.10 in this case is

```
Peak value=0.24393*exp(1.3427j)
Period:  Correct=30, Estimated=30 (samples)
Cycle:  Correct=0.20944, Estimated=0.20944(rad). Error=0
Phase:  Correct=0.62832, Estimated=0.67133(rad). Error=-0.043013
```

Listing A.10 is similar to Listing A.9, but estimates the cyclic correlation $C_X[\alpha_k, l]$ from a single realization instead of ensemble statistics. The main functions for these codes are ak_cyclicCorrelation.m and ak_cyclicCorrelationEnsemble.m, respectively. □

In both Listing A.10 and Listing A.9, the duration of the signals was carefully controlled to improve accuracy. More sophisticated algorithms use windowing or estimate the cyclic spectrum directly, via periodograms.

Example A.9. **Example: Cyclic spectrum via periodograms**. The cyclic spectrum $S_X(\alpha_k, \Omega)$ of Eq. (A.68) can be obtained by first estimating the cyclic correlation $C_X[\alpha_k, l]$ as in ak_cyclicSpectrum.m, which was used to obtain Figure A.25(d). Alternatively, $S_X(\alpha_k, \Omega)$ can be directly estimated via periodograms [Ant07, GG98].

Listing A.11 estimates the cyclic spectrum using a single realization of the modulated WGN in Example A.7.

[10] See, e. g., Eq. (2) in [GLAK94] and/or Eq. (15) in [GG98].

Listing A.11: *MatlabOctaveCodeSnippets/snip_appprobability_cyclo_spectrum_periodograms*

```
P=30; %sinusoid period (in samples)
N=1e3*P; %number of samples of this realization (force multiple of P)
Wc=2*pi/P; Phc=pi/5; %carrier frequency and phase, both in radians
n=0:N-1; carrier=transpose(cos(Wc*n+Phc)); %same for all realizations
x=randn(N,1).*carrier; %WGN modulated by sinusoidal carrier
L = length(x);    % signal length
Nw = 128;     % window length
Nv = fix(2/3*Nw); % block overlap
nfft = 2*Nw;    % FFT length
da = 1/L;           % cyclic frequency resolution (normalized freq)
cycleIndex = round(2*Wc/(2*pi*da)); %find the correct cycle
a1 = cycleIndex-20; % first cyclic freq. bin to scan (cycle a1*da)
a2 = cycleIndex+20; % last cyclic freq. bin to scan (cycle a2*da)
S = zeros(nfft,a2-a1+1); %pre-allocate space
for k = a1:a2; % Loop over all chose cyclic frequencies
    spectralObject = cps_w(x,x,k/L,nfft,Nv,Nw,'sym'); %cycle k/L
    S(:,k-a1+1) = spectralObject.S; %extract S(alpha,f) from object
end
alphas = 2*pi*da*(a1:a2); %get all cycles in rad
sumMagOverFreq=sum(abs(S)); %for each cycle, sum all magnitudes
indMax=find(sumMagOverFreq == max(sumMagOverFreq),1);
WEstimated=alphas(indMax)/2; %estimated frequency from cycle of peak
periodEstimated=round(2*pi/WEstimated); %period in samples
phaseEstimated=mean(angle(S(:,indMax)))/2;%half averaged peak's phase
```

Listing A.11 is based on the third-party function cps_w.m,[11] which allows calculating $S_X(\alpha_k, \Omega)$ for specific values of α_k. This is handy when there is previous knowledge about the cycles of interest.

The output of Listing A.11 is:

```
Period:  Correct=30, Estimated=30 (samples)
Cycle:  Correct=0.20944, Estimated=0.20944(rad). Error=-2.7756e-17
Phase:  Correct=0.62832, Estimated=0.64059(rad). Error=-0.012276
```

Note that this relatively good result was obtained with N chosen as a multiple of P and, consequently, the sinusoid frequency Wc is a multiple of the cycle resolution da. Changing N from 30 to 40 thousand samples led to:

```
Period:  Correct=30, Estimated=30 (samples)
Cycle:  Correct=0.20944, Estimated=0.20947(rad). Error=-2.618e-05
Phase:  Correct=0.62832, Estimated=0.10469(rad). Error=0.52363
```

which has an significantly increased phase estimation error. Practical issues when estimating cyclostationary statistics are discussed, for example, in [Ant07]. □

[11] This code is available on the Web [Ant07].

A.19.9 Two cyclostationary signals: sampled and discrete-time upsampled

Cyclostationarity is discussed here in the context of digital communications. More specifically, two signals of interest with periodic statistics are:

$$m_u[n] \quad \text{and} \quad x_s(t), \tag{A.72}$$

which are generated, respectively, by the following processes:

- the output $m_u[n]$ of an upsampler when its input $m[n']$ is a discrete-time WSS process;
- the sampled signal $x_s(t)$ obtained by periodic sampling of a continuous-time WSS process.

The upsampler of the former case is discussed in Section **??**, and its block diagram is repeated here for convenience:

$$m[n'] \rightarrow \boxed{\uparrow L} \rightarrow m_u[n].$$

The first goal of this discussion is to indicate that $m_u[n]$ is WSC (wide-sense cyclostationary) while $m[n']$ is WSS.

Example A.6 discusses that the input $m[n']$ is WSS and presents the corresponding autocorrelations $R_m[l]$ for a polar and unipolar binary signal. But, as informed in Section **??**, the upsampling by L is not a time-invariant operation and the output process cannot be assumed WSS in spite of the input being WSS. This fact can be confirmed by observing Figure A.26, which corresponds to the autocorrelation of an upsampled signal $m_u[n]$ obtained from a polar signal $m[n']$ with $L = 4$.

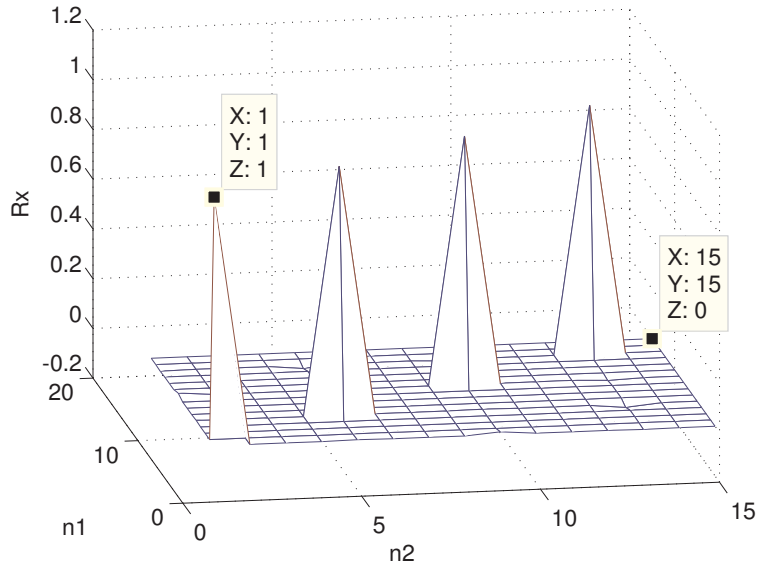


Figure A.26: *Autocorrelation of the cyclostationary $m_u[n]$ polar signal* ***upsampled*** *by $L = 4$.*

Figure A.26 was generated by Listing A.12. It can be seen that, different from Figure A.17, Figure A.26 cannot be represented by a one-dimensional function of the lag because the corresponding process is WSC, not WSS.

Listing A.12: *MatlabOctaveCodeSnippets/snip_digi_comm_upsampled_autocorr.m*

```
numRealizations = 500; numSamples = 4000; %500 waveforms
X=2*[floor(2*rand(numSamples,numRealizations))-0.5];
L=4; %oversampling (upsampling in this context) factor
Xu=zeros(L*numSamples,numRealizations);%pre-allocate space
Xu(1:L:end,:)=X; %generate upsampled realizations
[Rx,n1,n2] = ak_correlationEnsemble(Xu,15); %estimate correlation
mesh(n1,n2,Rx); %3-d plot
```

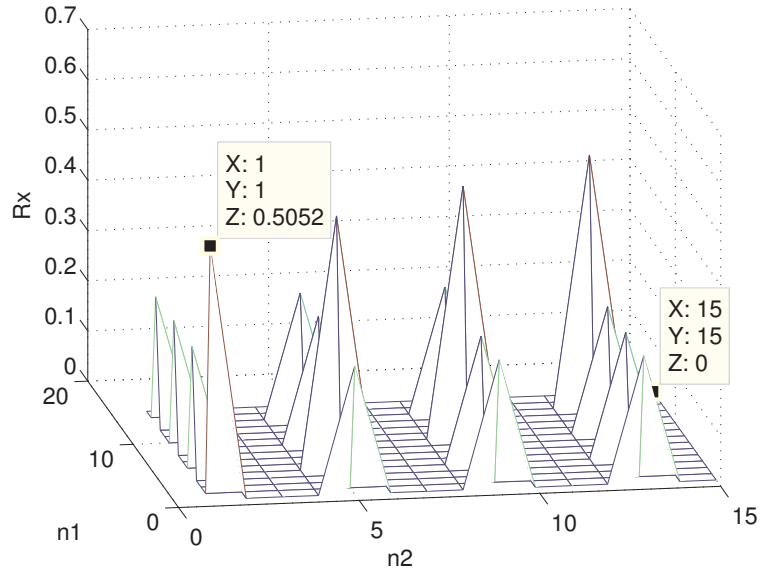


Figure A.27: *Autocorrelation of the cyclostationary $m_u[n]$ unipolar signal obtained with* ***upsampling*** *by $L = 4$.*

Similar to Figure A.26, Figure A.27 corresponds to the autocorrelation of an upsampled signal $m_u[n]$ obtained from a unipolar signal $m[n']$ with $L = 4$. In both cases the output processes are not WSS. For example, $R_X[1, 1] = 1$ and $R_X[15, 15] = 0$ in spite of both pairs (n_1, n_2) corresponding to a lag $l = n_2 - n_1 = 0$.

The reason for $m_u[n]$ not being WSS is that all realizations have well-defined time instants in which they have value zero and time instants in which there is randomness (the value may not be zero). In other words, the randomness is mixed along time with a deterministic behavior of zero values. For example, at $n = 0$ there is randomness, but all realizations have zero amplitude at $n = 1$. At $n = L$ the randomness appears again and so on. Note that the statistics are periodic and the random process is cyclostationary.

A.19.10 Converting a WSC into WSS by randomizing the phase

A WSC can be converted into a WSS by using the trick of "uniformly randomizing the phase". The following discussion, assumes the two signals $m_u[n]$ and $x_s(t)$ from Eq. (A.72), both representing realizations of WSC processes. The upsampled discrete-time signal $m_u[n]$ is discussed first, and then the results are adapted to the WSC process associated to $x_s(t)$.

The procedure of "uniformly randomizing the phase" means that $m_u[n]$ is randomly shifted to create a new signal $q[n] = m_u[n - \mathsf{K}]$, where K is a uniformly-distributed discrete random variable assuming integer values from 0 to $L - 1$ (periodic signals require another choice of values for K and are discussed later). The process is illustrated as

$$m[k] \rightarrow \boxed{\uparrow L} \rightarrow m_u[n] \rightarrow \boxed{\text{random shift by } \mathsf{K}} \rightarrow q[n]. \tag{A.73}$$

Example A.10. **Conversion of polar WSC process into WSS**. Based on Block (A.73), Figure A.28 shows three realizations of $q[n]$ for a polar[12] signal $m[k]$ with $L = 4$. The new process with realizations represented by $q[n]$ is WSS because the randomness is now spread over time.

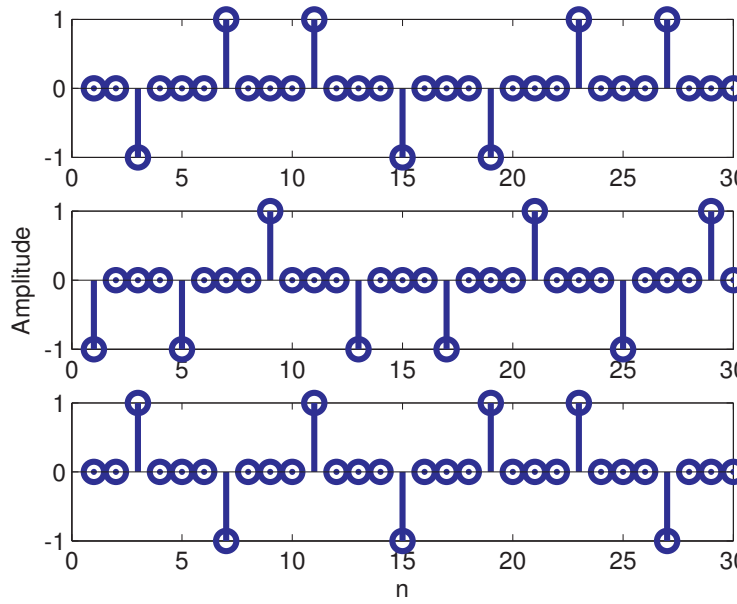


Figure A.28: *Realizations of an upsampled polar signal ($L = 4$) with random initial sample.*

Figure A.29 shows the ACF for the same process $q[n]$ that generated the realizations in Figure A.28. The random shift turned $q[n]$ into a WSS process. Because the product $\mathsf{X}_q[n_1]\mathsf{X}_q[n_2]$ is zero for several pairs of realizations, the original value $R_m[0] = 1$ (obtained for the signal $m[n]$) decreases to $R_q[0] = 1/L = 1/4$ when $q[n]$ is considered. □

[12] The polar signal is discussed in Example A.6 and Section **??**.

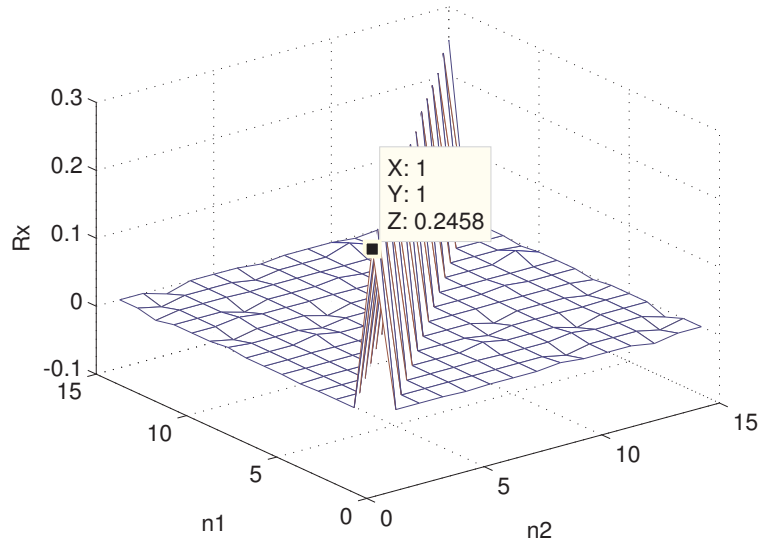


Figure A.29: *ACF for the same polar process that generated the realizations in Figure A.28. It can be seen that it has the characteristics of a WSS process ACF.*

Example A.11. **Conversion of upsampled sinusoids into WSS**. Another example is useful to illustrate the concept of randomizing the phase. This time the signal is a deterministic sinusoid of period equal to $N = 4$ samples. This signal is upsampled by $L = 4$ and randomly shifted. In this case, the uniformly distributed r.v. K assumes integer values in the range $[0, NL - 1]$ (see ak_upsampleRandomShift.m for an explanation). Some realizations of the resulting random process are depicted in Figure A.30.

Figure A.31 illustrates the correlation of the WSS process corresponding to Figure A.30. Note that the autocorrelation values (recall Eq. (1.44)) of the original sinusoid were reduced by a factor of 2 in Figure A.31. For example, $R_x[0] = A^2/2 = 4^2/2 = 8$ of the original cosine was reduced to $8/2 = 4$. □

In fact, from Eq. (**??**) and taking in account that the random shift will add $L - 1$ parcels equal to zero (decreasing the autocorrelation values by L), the autocorrelations of $m[k]$ and $q[n]$ in Block (A.73) are related by

$$R_m[l] \rightarrow \boxed{\uparrow L} \rightarrow \boxed{\div L} \rightarrow R_q[l],$$

that can be written mathematically as:

$$R_q[l] = \begin{cases} \frac{1}{L} R_m[l/L], & l = 0, \pm L, \pm 2L, \ldots \\ 0, & \text{otherwise} \end{cases} \tag{A.74}$$

The following example illustrates the use of Eq. (A.74).

Example A.12. **Conversion of an upsampled MA(1) process into WSS**. Assume

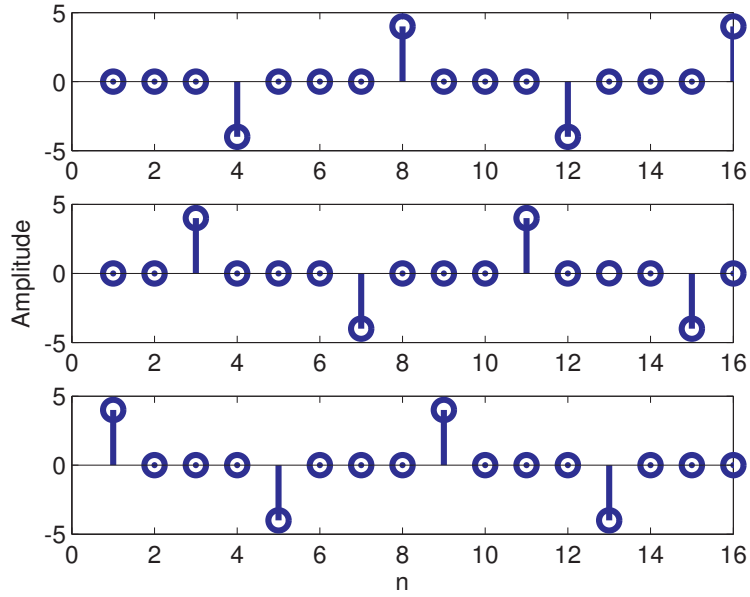


Figure A.30: *Three realizations of the random process corresponding to upsampling by 2 and randomly shifting a sinusoid of period $N = 4$ and amplitude $A = 4$. Note the original sinusoid already had zero values and one could think the upsampling factor was 4. The random shift* K *is such that any sample of the original sinusoid can occur at $n = 1$, for example.*

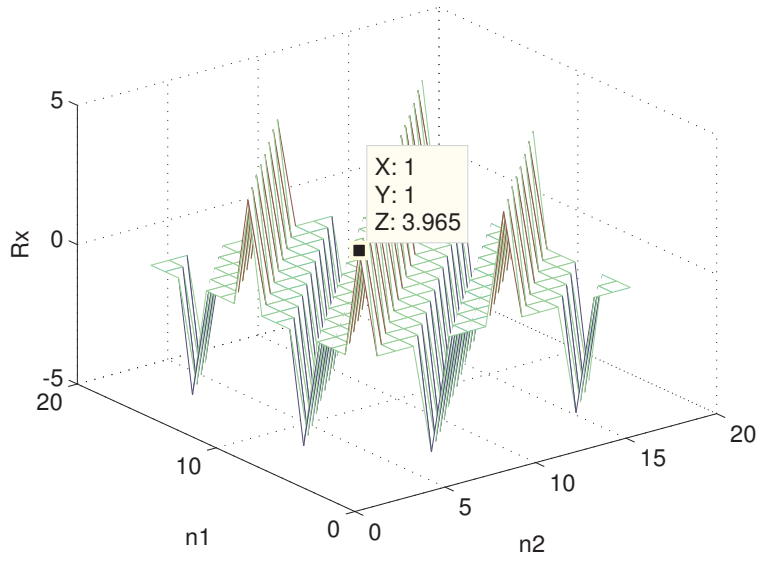


Figure A.31: *Correlation of the WSS process corresponding to Figure A.30: an upsampled sinusoid with random phase. It can be seen the characteristics of a WSS process.*

the MA(1) process of Example **??** is driven by WGN with power $\sigma^2 = 1$ and uses the FIR filter $B(z) = 1 - 0.8z^{-1}$. The MA(1) is then upsampled by a factor of $L = 3$ and becomes a WSC process. The processing of Block (A.73) is then used to convert it to a WSS. The goal here is to confirm Eq. (A.74).

In this case, from Eq. (**??**), the input to the upsampler has autocorrelation $R_m[l] = -0.8\delta[l+1] + 1.64\delta[l] - 0.8\delta[l-1]$. From Eq. (A.74), the autocorrelation of the phase randomizer block $q[n]$ has autocorrelation $R_q[l] = \frac{1}{3}(-0.8\delta[l+3] + 1.64\delta[l] - 0.8\delta[l-3]) \approx -0.267\delta[l+3] + 0.546\delta[l] - 0.267\delta[l-3]$. This can be confirmed with Listing A.13 that generates Figure A.32 (which can then be analyzed using a data cursor).

Listing A.13: *MatlabOctaveCodeSnippets/snip_app_correlationEnsemble.m*

```
B=[1 -0.8]; %Generate MA(1) process with these FIR coefficients
L=3; %upsampling factor (to upsample the MA(1) process)
numRealizations = 20000; %number of sequences (realizations)
numSamples = 300; %number of samples
%matrix X represents the input random process, add L extra samples
Z=randn(numSamples+L,numRealizations); %white Gaussian noise
X=filter(B,1,Z); %implement filtering by LTI system
X(1:2*length(B),:)=[]; %take out filter transient
Xup_shifted=ak_upsampleRandomShift(X, L); %upsample & random shift
[Rxx,n1,n2] = ak_correlationEnsemble(Xup_shifted,10); %get Rxx
subplot(121), ak_correlationMatrixAsLags(Rxx); %plot as n times lag
ylabel('lag'); xlabel('n'); colorbar; title('a) WSS')
Xup = zeros(L*size(X,1),numRealizations); %initialize with zeros
Xup(1:L:end,:)=X; %effectively upsample
subplot(122) %now the autocorrelation of the cyclostationary process
ak_correlationMatrixAsLags(ak_correlationEnsemble(Xup,10)); %plot Rxx
ylabel('lag'); xlabel('n'); title('b) WSC'); colorbar
```

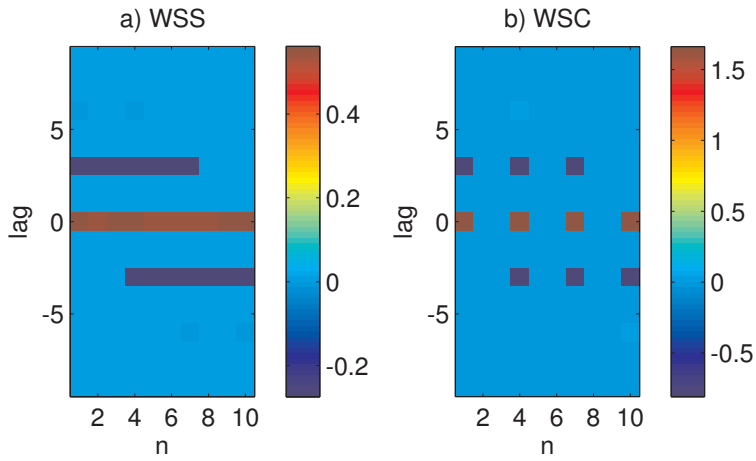


Figure A.32: *Autocorrelation matrices for the a) WSS process obtained by phase randomization of the b) WSC process.*

Figure A.32 illustrates that the upsampling creates zeros in the autocorrelation

over the lag l dimension and, that the phase randomization made the value of the autocorrelation independent of the absolute time n, as required by a WSS. □

Via the phase randomization, dealing with a WSC process has been circumvented, and one can obtain the PSD $S_q(e^{j\Omega})$ of $q[n]$ in Block (A.73) by calculating the DTFT of both sides of Eq. (A.74):

$$S_q(e^{j\Omega}) = \frac{1}{L} S_m(e^{jL\Omega}). \tag{A.75}$$

Observe that the PSD $S_m(e^{j\Omega})$ of $m[n]$ is simply obtained via the DTFT of $R_m[l]$, i. e.,

$$S_m(e^{j\Omega}) = \mathcal{F}\{R_m[l]\} = \sum_{l=-\infty}^{\infty} R_m[l] e^{-j\Omega l}, \tag{A.76}$$

such that Eq. (A.75) can be written as

$$S_q(e^{j\Omega}) = \frac{1}{L} \sum_{l=-\infty}^{\infty} R_m[l] e^{-jL\Omega l}. \tag{A.77}$$

The phase randomization of Block (A.73) and, consequently, Eq. (A.77), has several applications. For example, it can be used to obtain the PSD of a discrete-time PAM signal, generated as in Block (**??**).

The second case of a WSC listed in the beginning of this section occurs when a sampled signal $x_s(t)$ is obtained by periodic sampling of a continuous-time WSS process. Similar to Block (A.73), a continuous-time randomization with a uniformly distributed phase can be represented by

$$m[n] \rightarrow \boxed{\text{D/C}} \rightarrow m_s(t) \rightarrow \boxed{\text{random phase}} \rightarrow q_s(t), \tag{A.78}$$

and it turns the sampled signal $q_s(t)$ into a realization of a WSS process. This kind of strategy is useful, for example, to obtain the PSD of continuous-time PAM signals, as the ones generated by Block (**??**).

The PSD of $q_s(t)$ is given by

$$S_q(\omega) = \frac{1}{T_{\text{sym}}} S_m(e^{j\Omega})|_{\Omega=\omega T_{\text{sym}}}, \tag{A.79}$$

which is similar to Eq. (A.75) and also has the PSD $S_m(e^{j\Omega})$ of $m[n]$ (not $m_s(t)$) provided by Eq. (A.76).

Note that due to the D/C conversion in Block (A.78), $S_m(e^{j\Omega})$ is mapped to continuous-time ω in rad/s using the relation $\omega = \Omega F_s$. In this specific case, $F_s = R_{\text{sym}} = 1/T_{\text{sym}}$, which leads to:

$$S_m(e^{j\Omega})|_{\Omega=\omega T_{\text{sym}}} = \sum_{l=-\infty}^{\infty} R_m[l] e^{-j\omega T_{\text{sym}} l}.$$

Hence, Eq. (A.79) can be written as

$$S_q(\omega) = \frac{1}{T_{\text{sym}}} \sum_{l=-\infty}^{\infty} R_m[l] e^{-j\omega T_{\text{sym}} l}. \tag{A.80}$$

or, alternatively, with f in Hz:

$$S_q(f) = \frac{1}{T_{\text{sym}}} \sum_{l=-\infty}^{\infty} R_m[l] e^{-j2\pi f T_{\text{sym}} l}. \tag{A.81}$$

From Eq. (A.76), note that $S_m(\omega)$ is periodic, as it is a version of $S_m(e^{j\Omega})$ with a scaled abscissa. Consequently, $S_q(\omega)$ are $S_q(f)$ also periodic, in spite of the notation using the independent variables ω and f.

A.20 Estimation Theory

A.20.1 Probabilistic estimation theory

Estimation theory concerns estimating the values of parameters θ based on measured data. An *estimator* attempts to approximate the unknown parameters using measurements that have a random component (e. g., due to noise). There are other approaches, but the one of interest to this text is the probabilistic estimation theory, which aims at finding a set $\Theta = \{\theta_1, \ldots, \theta_M\}$ of M parameters. In this approach, the measured data set Ξ with N elements is random and with a probability distribution $p(\Xi|\Theta)$ that depends on Θ. The parameters θ themselves may have associated probability distributions. An estimator provides $\hat{\Theta} = \{\hat{\theta}_i\}$, where the "hat" suggests the element $\hat{\theta}_i$ is an estimate of the true parameter θ_i. The value $\hat{\theta}_i$ is a statistics with distribution, variance, mean $\mathbb{E}[\hat{\theta}_i]$, etc.

The *bias* of an estimator is $\mathbb{E}[\hat{\theta}_i] - \theta_i$ and an estimator is called *unbiased* if the bias is zero for all M elements of $\hat{\Theta}$. The estimator of the sample variance that uses N as normalization factor instead of $N-1$ is a good example of a biased estimator [urlBMbia]. There is a well-known tradeoff between bias and variance. In many applications it is possible to decrease the variance of an estimator at the expenses of allowing its bias to increase.

Some specific topics of probabilistic estimation theory are discussed in the sequel. But this theory is broad and some of the (many) concepts out of the scope of this text are:

- Cramér-Rao bound (CRB): lower bound of the variance of an estimator;
- Several estimators: Maximum likelihood (MLE), maximum a posteriori (MAP), generalized Bayes, etc.;
- Non-linear estimators such as artificial neural networks;
- Dynamic systems: Kalman filter, recursive Bayesian estimation, etc.

A.20.2 Minimum mean square error (MMSE) estimators

A popular estimator is the minimum mean square error (MMSE) estimator, which utilizes the error $e_i = \theta_i - \hat{\theta}_i, i = 1, \ldots, M$, between the estimated parameters and their actual values.

The measurements Ξ may be organized in several different forms. For example, in multivariate statistics, each element of Ξ can be an array. For simplicity, it is assumed here that Ξ and Θ are column vectors $\mathbf{y}$ and $\mathbf{x}$ with N and M complex-valued elements, respectively. The estimator $\hat{\mathbf{x}}(\mathbf{y})$ is a function of the measurements $\mathbf{y}$ and the estimation error is $\mathbf{e} = \hat{\mathbf{x}} - \mathbf{x}$ with elements $e_i, i = 1, \ldots, M$. Because the vectors are random, the MSE is given by

$$\text{MSE} \stackrel{\text{def}}{=} \sum_{i=1}^{M} \mathbb{E}[|e_i|^2], \tag{A.82}$$

which can be written in matrix notation as the expected value of the trace

$$\text{MSE} = \text{tr}\left\{\mathbb{E}\left[\mathbf{e}\mathbf{e}^H\right]\right\}, \tag{A.83}$$

where $R_{ee} = \mathbb{E}[\mathbf{e}\mathbf{e}^H]$ is the error autocorrelation matrix.

A.20.3 Orthogonality principle

AK-TODO Orthogonality principle

A.21 One-dimensional linear prediction over time

A.21.1 The innovations process

When white noise with power σ_n^2 is passed through an LTI filter $M(z)$, the output PSD is

$$S(e^{j\Omega}) = \sigma^2 |M(e^{j\Omega})|^2. \tag{A.84}$$

Conversely, when a PSD $S(e^{j\Omega})$ satisfies the corresponding conditions,[13] it is possible to find a filter $M(z)$ that obeys Eq. (A.84).

As discussed in [BLM04] (page 71), a WSS random process $\mathcal{X}[n]$ with PSD $S_X(e^{j\Omega})$ allows the following monic[14] minimum-phase spectral factorization of the associated[15] "transfer function" $S_X(z)$:

$$S_X(z) = \sigma_i^2 M_x(z) M_x^*(1/z^*), \tag{A.85}$$

[13] The spectral factorization of Eq. (A.85) is valid when $S(z)$ satisfies the Paley-Wiener condition [BLM04] (page 33).

[14] In a monic filter, the highest order coefficient in the denominator is unitary.

[15] See [BLM04], page 25, for explanations on reflected transfer functions $H^*(1/z^*)$ and also Example 2-13 in page 31. Note that when $H(z)H^*(1/z^*)$ is evaluated on the unit circle, the result is $|H(e^{j\Omega})|^2$.

where $M_x(z)$ is a monic loosely[16] minimum-phase causal filter and σ_i^2 is the variance of the innovation process or, equivalently, the geometric mean[17] of $S_X(e^{j\Omega})$.

If $M_x(z)$ is strictly minimum phase, $M_x^{-1}(z)$ is stable and the following filtering process

$$\mathcal{X}[n] \rightarrow \boxed{M_x^{-1}(z)} \rightarrow \mathcal{I}[n] \rightarrow \boxed{M_x(z)} \rightarrow \mathcal{X}[n]$$

illustrates how to generate a white WSS process $\mathcal{I}[n]$ with power σ_i^2 from $\mathcal{X}[n]$ (the "whitening" operation) and how to recover $\mathcal{X}[n]$ from $\mathcal{I}[n]$.

The filter $M_x^{-1}(z)$ removes correlated components in $\mathcal{X}[n]$ and obtains $\mathcal{I}[n]$ with characteristics of white noise. Hence, it is called *whitening filter*. And because $\mathcal{I}[n]$ corresponds to the new and random part of the process, it is called the innovation sequence.

Linear prediction for noise decorrelation

The whitening filter $M_x^{-1}(z)$ is also called *prediction-error filter* because its action can be split into two stages, assuming linear prediction:

1. a prediction $\tilde{\mathcal{X}}[m] = \sum_{k=1}^{\infty} p_k \mathcal{X}[m-k]$ of the current (m-th) sample of $\mathcal{X}[n]$ is obtained from past samples of $\mathcal{X}[n]$ via the *prediction filter*

$$P(z) = \sum_{n=1}^{\infty} p_n z^{-n}, \tag{A.86}$$

   which is $P(z) = 1 - M_x^{-1}(z)$

2. and then this prediction is subtracted from the current sample to produce the prediction error $\mathcal{I}[m] = \mathcal{X}[m] - \tilde{\mathcal{X}}[m]$. It can be shown that this scheme is an optimum minimum mean square error (MMSE) estimator[18] [BLM04] (page 72).

Using the notation of "prediction", the whitening can be obtained with:

$$\mathcal{X}[n] \rightarrow \boxed{1 - P(z)} \rightarrow \mathcal{I}[n]$$

In summary, the filter $M_x^{-1}(z) = 1 - P(z)$ is often used to perform *decorrelation* of a signal while $M_x(z)$ is called the *synthesis filter* because it can generate a signal $\mathcal{X}[n]$ with a given PSD from white noise $\mathcal{I}[n]$.

When $\mathcal{X}[m]$ is a correlated noise, the decorrelation procedure decreases its power and is typically beneficial for decoding purposes.

---

[16] A loosely minimum-phase filter can have zeros on unit circle while a strictly minimum phase has all zeros within the unit circle.

[17] The geometric mean can be easily calculated with `geomean` in Octave/Matlab, but as discussed in [BLM04], page 30, just before Example 2-12, the zeros of $S_X(e^{j\Omega})$ must be avoided by, e. g., adding a small constant. The following code provides an example: w=linspace(-pi,pi,10000); S=2-2*sin(w); S=S+eps; Sg=geomean(S), Sa=mean(S)

[18] MMSE linear estimation theory if further discussed in [CF97].

The power of the prediction error $\mathcal{I}[m]$ is σ_i^2, which is the geometric mean of the PSD $S_X(e^{j\Omega})$, while the power σ_x^2 of $\mathcal{X}[m]$ is the average of $S_X(e^{j\Omega})$. Hence, linear prediction reduces the noise power by

$$\text{prediction gain} = 10\log_{10}\left(\frac{\sigma_x^2}{\sigma_i^2}\right) \quad \text{dB}. \tag{A.87}$$

Autoregressive (AR) linear prediction

There are computationally efficient methods to estimate $M(z)$ when it is restricted to be and autoregressive (AR) filter $M(z) = 1/A(z)$. For example, the Yule-Walker equations consists of an efficient method to estimate $A(z)$ [Hay01]. In this case, it is assumed that $\mathcal{X}[n]$ is a realization of an ergodic autoregressive random process of order P, denoted as AR(P).

The problem can then be posed mathematically as finding the FIR filter $A(z) = 1 - \sum_{i=1}^{P} a_i z^{-i}$ that minimizes the energy $\sum_n |\mathcal{I}[n]|^2$ of the output signal $\mathcal{I}[n]$ in the filtering process:

$$\mathcal{X}[n] \rightarrow \boxed{A(z)} \rightarrow \mathcal{I}[n].$$

In this case, the prediction-error filter can be written as $M^{-1}(z) = A(z) = 1 - \tilde{P}(z)$, where $\tilde{P}(z) = \sum_{k=n}^{P} a_k z^{-k}$ predicts an estimate $\tilde{\mathcal{X}}[n]$ of $\mathcal{X}[n]$ as

$$\tilde{\mathcal{X}}[n] = \sum_{k=1}^{P} a_k \mathcal{X}[n-k], \tag{A.88}$$

based on the P past samples of $\mathcal{X}[n]$.

AR-based modeling using (A.88) is widely used in speech coding and, because $\mathcal{X}[n]$ in (A.88) is a linear combination of past inputs, it is called *linear predictive coding* (LPC).

codllinearPredictionExample provides two examples of linear prediction using or not an AR process.

The result of running codllinearPredictionExample with useARProcess = 1 is the following:

```
prediction gain = 14.0937 dB
Signal power (time-domain) = 422.8282
Signal power (via PSD arithmetic mean) = 412.0798
Innovation power (via PSD geometric mean) = 16.0552
Innovation power (via AR modeling) = 15.8997
Innovation power (time-domain) = 15.8994
```

Running codllinearPredictionExample with useARProcess = 0 leads to the following:

```
prediction gain = 2.9934 dB
Signal power (time-domain) = 72.5961
Signal power (via PSD arithmetic mean) = 71.8227
```

```
Innovation power (via PSD geometric mean) = 36.0512
Innovation power (via AR modeling) = 35.7824
Innovation power (time-domain) = 35.7821
```

In both cases the results were consistent. In the second case, an order 10 for the AR filter was adopted. Note that moving the poles closer to unit circle increases the prediction gain. This occurs because, by moving the poles closer to the unit circle, the frequency response (and the power spectral density) tends to present a shaper peak with higher magnitude. This, in turn, has produces a much more significant increase in the arithmetic mean than on the geometric mean of the PSD, which implies in an prediction gain increase, as inferred from Eq. (A.87).

A.22 Vector prediction exploring spatial correlation

Instead of exploring correlation over time, this section discusses methods to explore the so-called spatial correlation: an element of the random vector being estimated based on the other elements of this vector. As signal model, the Gaussian block or "packet" ISI channel [CF97], (page 84, Eq. 4.6) is adopted here, which is given by

$$\mathbf{Y} = H\mathbf{X} + \mathbf{N}, \tag{A.89}$$

where $\mathbf{X}$ is a zero-mean complex random input m-vector, $\mathbf{Y}$ is a zero-mean complex random output n-vector, $\mathbf{N}$ is a complex zero-mean Gaussian noise n-vector independent of $\mathbf{X}$ and H is the complex $n \times m$ channel matrix [CF97]. For these random vectors, the correlation matrices coincide with covariance matrices. [19]

The output covariance matrix is given by

$$R_{yy} = HR_{xx}H^* + R_{nn}, \tag{A.90}$$

where a matrix superscript $*$ denotes Hermitian (transpose conjugate).

A characteristic representation of a random m-vector $\mathbf{X}$ is given by the linear combination of the columns of a matriz F, whose determinant is equal to one,[20] weighted by a vector of uncorrelated random variables $\mathbf{V}$, that is:

$$\mathbf{X} = F\mathbf{V}. \tag{A.91}$$

Hence, the covariance matrix of $\mathbf{X}$ is given by:

$$R_{xx} = FR_{vv}F^* \tag{A.92}$$

where R_{vv} is diagonal (the random variables in $\mathbf{V}$ are uncorrelated).

There are two alternatives of interest for representing a vector in its characteristic form, the modal and the innovations representation. The first is derived from the

[19] Eq. (15) in [GP06] corresponds to Eq. (A.89), assuming the channel matrix H is diagonal because vectoring of the V lines already partitioned the channel.

[20] A matrix whose determinant is equal to one is said to be equi-areal and orientation-preserving.

eigendecomposition of R_{xx}. Given the factorization $R_{xx} = U\Lambda_x^2 U^* = (U\Lambda_x)(U\Lambda_x)^*$, by comparison to Eq. (A.92), F corresponds to the unitary matrix U from the eigendecomposition, while the uncorrelated vector $\mathbf{V}$ from Eq. (A.91) corresponds to $U^{-1}\mathbf{X}$. Meanwhile, the latter (innovations representation) is derived from the Cholesky decompostion. In a similar manner, given the factorization $R_{yy} = LD_y^2 L^* = (LD_y)(LD_y)^*$, F corresponds to the lower triangular matrix L, while $\mathbf{V}$ corresponds to $L^{-1}\mathbf{X}$.

The important conclusion yielded by these two representations is that the a given vector $\mathbf{X}$ whose random variables are correlated can be *whitened* by a forward section given by U^{-1}, the inverse of the unitary matrix from the eigendecomposition of its covariance matrix, or L^{-1}, the inverse of the lower triangular matrix from the Cholesky decomposition of its covariance matrix.

The innovations representation is a natural adaptation of linear prediction over time and is obtained with a Cholesky factorization of R_{yy}, while the modal representation can be obtained via eigenanalysis or SVD.

The optimum MMSE linear predictor in this scenario is

$$\tilde{\mathbf{Y}} = P\mathbf{Y}, \tag{A.93}$$

where the predictor matrix P is given by

$$P = I - L^{-1}, \tag{A.94}$$

with L being obtained from the innovations representation $R_{yy} = LD_y^2 L^*$ and I being the identity matrix. It was assumed that R_{yy} is nonsingular, otherwise the pseudo inverse can be used.

Because L is lower triangular and monic, its inverse is also lower triangular and monic. The subtraction of L^{-1} from I makes P to be lower triangular with zeros in the main diagonal. This structure imposes a causal relation among the elements of $\mathbf{Y}$, such that $\tilde{\mathbf{Y}}$ can be obtained recursively.

The error vector is

$$\mathbf{E} = \mathbf{Y} - \tilde{\mathbf{Y}} = \mathbf{Y} - P\mathbf{Y} = \mathbf{Y} - (I - L^{-1})\mathbf{Y} = L^{-1}\mathbf{Y}. \tag{A.95}$$

In general, the sum mean-squared prediction error is

$$\mathbb{E}[||\mathbf{E}||^2] = \mathbb{E}[||\mathbf{Y} - \tilde{\mathbf{Y}}||^2] = \text{trace}\{R_{ee}\}, \tag{A.96}$$

where $R_{ee} = \mathbb{E}[\mathbf{E}\mathbf{E}^*]$ is the autocorrelation matrix of $\mathbf{E}$. It can be proved (see, e. g., [BLM04]) that when the optimum linear predictor of Eq. (A.94) is adopted, the error power $\text{trace}\{R_{ee}\}$ achieves its minimum value given by $\text{trace}\{D_y^2\}$. This avoids the step of estimating R_{ee} to obtain the prediction gain, which is given by

$$\text{prediction gain} = 10\log_{10}\left(\frac{\text{trace}\{R_{xx}\}}{\text{trace}\{R_{ee}\}}\right) = 10\log_{10}\left(\frac{\text{trace}\{R_{xx}\}}{\text{trace}\{D_y^2\}}\right) \quad \text{dB}. \tag{A.97}$$

Hence, making an analogy with prediction over time, repeated here for convenience:

$$\mathcal{X}[n] \rightarrow \boxed{M_x^{-1}(z)} \rightarrow \mathcal{I}[n] \rightarrow \boxed{M_x(z)} \rightarrow \mathcal{X}[n]$$

the spatial prediction allows to obtain

$$\mathbf{Y} \rightarrow \boxed{L^{-1}} \rightarrow \mathbf{E} \rightarrow \boxed{L} \rightarrow \mathbf{Y},$$

which is expressed in matrix notation as $\mathbf{E} = L^{-1}\mathbf{Y}$ and $\mathbf{Y} = L\mathbf{E}$.

codlspatialLinearPredictionExample illustrates an example discussed in [BLM04].

In codlspatialLinearPredictionExample, the predictor matrix is

```
P =     [0,         0,         0;
    0.5000,         0,         0;
         0,    0.5000,         0]
```

and the prediction gain is 0.7463 dB. Adopting a new correlation matrix Rxx=[10, 8, 2; 8 10 10; 2 10 10] leads to

```
P =     [0,         0,         0;
    0.8000,         0,         0;
   -1.6667,    2.3333,         0]
```

and a prediction gain of 9.2082 dB.

Note that the first element $\tilde{y}_1$ of $\tilde{\mathbf{Y}} = [y_1, \ldots, y_V]$ in $\tilde{\mathbf{Y}} = P\mathbf{Y}$ is always zero due to the structure of P. Then, the second element $\tilde{y}_2$ is a scaled version of the first element y_1 of $\mathbf{Y}$, and so on.

A.23 Spatial whitening applied to interference mitigation

Assume that a communication channel has been partitioned into independent parallel "subchannels" by DMT or SVD (SVD-based partitioning is summarized in Application **??**). Then, the predictor of Eq. (A.94) can be used as follows.

Assume a single "subchannel", which would correspond to a single tone k in DMT. Also, as depicted in figlalien_crosstalk, assume that M interferers provoke crosstalk on the V copper lines of interest of the vectored group through a channel matrix H of dimension $V \times M$ as in Eq. (A.89). For DMT, the matrix H corresponds to the frequency response (eventually complex-valued) at tone k. The vectors corresponding to transmission over tone k in a DMT system are [GP06] (Eq. (15)):

$$\mathbf{Z}_k = T_k\mathbf{W}_k + \mathbf{N}_k, \tag{A.98}$$

where T_k is a diagonal channel matrix and $\mathbf{N}_k$ is the noise corresponding to both thermal (background) noise and alien crosstalk. Hence, the autocorrelation R_{nn} of $\mathbf{N}_k$ is nondiagonal and the goal is to reduce the noise power via decorrelation (whitening).

Using the innovations representation, R_{nn} is factored as $R_{nn} = LD_y^2L^*$

For simplicity, consider a noise free condition ($\mathbf{N} = 0$ in Eq. (A.89)) and that the interferers are i.i.d. zero-mean Gaussians with autocorrelation $R_{xx} = \sigma_x^2\mathbf{I}$. From Eq. (A.90),

$$R_{yy} = HR_{xx}H^* = \sigma_x^2 HH^*.$$

The predictor of Eq. (A.94) can be applied to $\mathbf{Y}$ to decrease...

codlto-be-done performs the following experiment: calculates the prediction gain over frequency for a set measured crosstalk channels.

A.24 Space-time prediction

the Gaussian MIMO channel [BLM04] (page 475, Fig. 10-5).

A.25 Non-linear decorrelation

This is related with blind identification and ICA.

Non-linear decorrelation algorithms are discussed at [urlоooi].

Examples:

Nonlinear decorrelator for multiuser detection in non-Gaussian impulsive environments Author(s): T.C. Chuah; B.S. Sharif; O.R. Hinton Source: Electronics Letters, Volume 36, Issue 10, 11 May 2000, p. 920 - 922

A. Cichocki and R. Unbehauen. Robust neural networks with on-line learning for blind identification and blind separation of sources. IEEE Trans. on Circuits and Systems, 43(11):894-906, 1996.

A.26 noise prediction and DFE

"ideal DFE assumption": inputs to "past" subchannels are available to the receiver when decoding the current subchannel [CF97], page 82.

from Proakis

A.27 Decibel (dB) and Related Definitions

The decibel (dB) expresses a ratio between two powers $\mathcal{P}_1$ and $\mathcal{P}_2$:

$$T_{\text{dB}} = 10\log_{10}\frac{\mathcal{P}_1}{\mathcal{P}_2}. \tag{A.99}$$

Hence, dB is a relative, not an absolute quantity. If $T_{\text{dB}} = 0$, then the powers are equal. If T_{dB} is positive, $\mathcal{P}_1$ is larger than $\mathcal{P}_2$ and vice-versa if $T_{\text{dB}} < 0$.

There are other definitions related to dB such as dBW, which is the relative power in dB of a signal with respect to 1 Watt (W). An important distinction is that, while dB should always be a "relative" measure indicating a ratio of powers, dBW is an "absolute" measure that indicates one specific power value. For example, assuming that

$\mathcal{P}$ is power in Watts, its conversion to $\hat{\mathcal{P}}$ in dBW is given by

$$\hat{\mathcal{P}} = 10 \log_{10} \mathcal{P} \qquad (\mathcal{P} \text{ in W and } \hat{\mathcal{P}} \text{ in dBW}).$$

For example, 100 W corresponds to 20 dBW.

Similarly, dBm is defined with respect to 1 milliwatt (mW). Assuming that $\mathcal{P}$ is power in Watts,

$$\hat{\mathcal{P}} = 10 \log_{10} \frac{\mathcal{P}}{10^{-3}} \qquad (\mathcal{P} \text{ in W and } \hat{\mathcal{P}} \text{ in dBm}).$$

For example, if a receiver detects a signal level of −13 dBm, it means this signal has a power that is 13 dB smaller than a milliwatt. For example, on a 3G cell phone using five "bars" to indicate signal strength, the 1-bar may correspond to −113 dBm and 5-bars to −100 dBm.

Another definition derived from dB is dBc, which is related to the presence of a carrier, such as in radio communications. In this case, the reference is the power of the strongest carrier. Therefore, typically, the dBc value of a signal component is negative.

A more complicated definition is dBm0, used for example in audio and telephony. It means the level compared to a milliwatt after the value is adjusted to make a reference ("correct") value be 0 dBm.

One unfortunate fact is that sometimes dB is erroneously interpreted as the absolute value of a signal power. For example, in sound engineering, the dB *sound pressure level* or dB SPL is widely used and it represents the ratio between the measured sound pressure level and the reference point. However, the "SPL" suffix and the reference power value are often omitted and dB may be incorrectly understood as an absolute power value in these cases.

In digital signal processing, most of the time the correct unit is unknown and, consequently, there is no interest on specifying the reference value. If the signals are assumed to be in Volts and obtained from a resistance of 1 Ohm, in many cases the correct unit would be dBW instead of dB. For example, in spectral analysis, it is common to convert a power spectral density $S(f)$ in W/Hz to $10 \log_{10} S(f)$, which should be interpreted as dBW/Hz. For example, the periodogram.m function in Matlab/Octave shows graphs in dB/Hz but the informed unit could be dBW/Hz.

Sometimes dB is used to express voltage ratios, not power ratios. For example, assuming purely resistive impedance R, the power associated to a voltage V is $\mathcal{P} = V^2/R$. In this case Eq. (A.99) can be written as

$$T_{\text{dB}} = 10 \log_{10} \left(\frac{V_1^2/R}{V_2^2/R} \right) = 20 \log_{10} \left(\frac{V_1}{V_2} \right). \tag{A.100}$$

Both voltage values in Eq. (A.100) should be measured over the same impedance. As an example of a possible mistake, consider an amplifier that requires 1 V from a 1,000 Ohms source to output 40 V over a 10 Ohms speaker. In this case, it is not strictly correct to

use Eq. (A.100) and state that this amplifier has a "gain" of $20\log_{10}(40/1) \approx 32$ dB. In fact, the power ratio in this case is $10\log_{10}((40^2/10)/(1/1000)) \approx 52$ dB.

There are also definitions of absolute values for voltage ratios such as dBmV, which for the cable industry represents decibels relative to 1 millivolt across 75 Ohms (the impedance of a coaxial cable). Hence, 0 dBmV corresponds to $10\log_{10}((10^{-3})^2/75) \approx -78.75$ dBW or -48.75 dBm.

Some other examples of dB usage:

- To convert from dBW to dBm, simply add 30.
- If a signal is transmitted with power 5 dBm and goes through a channel that attenuates it by 3 dB, it arrives at the receiver with 2 dBm of power.
- A sinusoid with amplitude $A = 10$ Volts has average power $\mathcal{P} = A^2/2 = 50$ W, which corresponds to ≈ 16.99 dBW.
- A DC signal with amplitude $A = 10$ volts, has average power 100 watts or, equivalently, 20 dBW.
- Assume one wants to design a (polar) signal that assumes only two amplitude values $-A$ and A with power -9 dBm. The first step is to convert the power -9 dBm to $10^{-9/10} = 0.126$ mW. Assuming A is in Volts, this signal has power A^2 Watts and the amplitude should be $A = \sqrt{0.126 \times 10^{-3}} = 0.011$ V.

A.28 Insertion loss and insertion frequency response

A system is often composed by the interconnection of several blocks and it is of interest to know how a signal is attenuated (or amplified!) as it passes through the blocks. The *insertion loss* (IL) is widely used to characterize the loss in power, for example, of a transmission line or a circuit. This section discusses the IL of a generic *two-port network* (or *quadripole*), which is any four-terminal network called here *device under test* (DUT). Material about IL in specific applications can be found elsewhere. For example, in [ANS03] (more specifically, "C.3.1.7 Relationship of transfer function and *insertion loss*") there is an interesting discussion of IL in the DSL systems context.

IL estimation is based on a smart strategy: measure the system output with and without the DUT, with the IL being the attenuation imposed by the DUT.

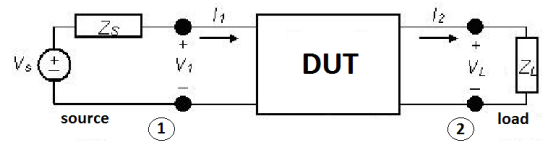


Figure A.33: *Basic setup for measuring the insertion loss of a device under test (DUT).*

Figure A.33 depicts a setup for measuring the insertion loss of a DUT. The behavior without the DUT is obtained by removing it (i. e., directly connecting the pairs of terminals 1 and 2, and measuring the power over the load, which in this case will be denoted as $\mathcal{P}_L^{\text{no}}$. Then, the behavior with the DUT is obtained by inserting it and again measuring the power over the load, which will be denoted as $\mathcal{P}_L^{\text{yes}}$. The insertion loss

in dB is defined as

$$\mathrm{IL} \stackrel{\mathrm{def}}{=} \frac{\mathcal{P}_L^{\mathrm{no}}}{\mathcal{P}_L^{\mathrm{yes}}} \tag{A.101}$$

and is often given in dB:

$$\mathrm{IL}_{\mathrm{dB}} \stackrel{\mathrm{def}}{=} 10\log_{10}\left(\frac{\mathcal{P}_L^{\mathrm{no}}}{\mathcal{P}_L^{\mathrm{yes}}}\right). \tag{A.102}$$

For example, a passive optical device that splits the input into two output signals could have an IL of 4 dB per output, consisting of the ideal 3 dB IL for splitting the power in halves plus an *excess loss* of 1 dB per port.

If the system behavior varies with frequency, $\mathrm{IL}(f)$ can be measured and reported over the frequency range of interest. Besides, in case the DUT amplifies its input, one can use the insertion gain $\mathrm{IG}(f) = 1/\mathrm{IL}(f)$ instead.

Another alternative definition of IL uses the voltage V_L over the load. Repeating the procedure of removing and then inserting the DUT, the measured values of V_L are denoted as V_L^{no} and V_L^{yes}, respectively. In this case and assuming frequency-dependence, IL can be written as

$$\mathrm{IL}(f) = 10\log_{10}\frac{|V_L^{\mathrm{no}}(f)|^2}{|V_L^{\mathrm{yes}}(f)|^2} = 20\log_{10}\frac{|V_L^{\mathrm{no}}(f)|}{|V_L^{\mathrm{yes}}(f)|}. \tag{A.103}$$

The "insertion" trick (measuring with and without the DUT) can be used to obtain parameters other than the IL. For example, in some situations it is inconvenient to deal with (or difficult to measure) the frequency response $H(f)$ directly because it depends on the source and load impedances. In these cases, it may be advantageous to measure the *insertion frequency response*

$$H_{\mathrm{I}}(f) = \frac{V_L^{\mathrm{yes}}(f)}{V_L^{\mathrm{no}}(f)} \tag{A.104}$$

and convert it to $H(f)$ when convenient and the source Z_s and load Z_l impedances are known. Note that Eq. (A.104) does not discard the phases of the measured voltages as in Eq. (A.103), and can be obtained for example with a network analyzer equipment.

Having $H_{\mathrm{I}}(f)$, the frequency response $H(f)$ relating the output voltage $V_L^{\mathrm{yes}}(f)$ (with the DUT) to the input source voltage $V_S(f)$ can be obtained as

$$H(f) \stackrel{\mathrm{def}}{=} \frac{V_L^{\mathrm{yes}}(f)}{V_s(f)} = \frac{V_L^{\mathrm{no}}(f)}{V_s(f)}\frac{V_L^{\mathrm{yes}}(f)}{V_L^{\mathrm{no}}(f)} = \frac{Z_L(f)}{Z_S(f)+Z_L(f)}H_{\mathrm{I}}(f). \tag{A.105}$$

When $Z_L(f)$ and $Z_S(f)$ can be considered constant over the frequency of interest, Eq. (A.105) is conveniently expressed as

$$H(f) = \frac{Z_L}{Z_S+Z_L}H_{\mathrm{I}}(f). \tag{A.106}$$

$H_{\mathrm{I}}(f)$ itself is of interest in some applications. For example, when estimating the channel capacity of a DSL copper loop, $H_{\mathrm{I}}(f)$ is adopted instead of $H(f)$ [ANS03].

The "insertion" procedure is used in many distinct scenarios but people tend to keep together the terms "insertion" and "loss", instead of using only the former. For example, some authors (e. g. [Cio10] and in Figure **??**) call insertion *loss* to $\mathrm{IG}(f)$ and report an attenuation as a negative value in dB. Others call the insertion frequency response of Eq. (A.106) as "insertion loss", as in the FTW software, or "loop insertion gain transfer function" as in [ANS03].

A.29 Discrete and Continuous-Time Impulses

A.29.1 Discrete-time impulse function

The discrete-time impulse function $\delta[n]$, also called *unit impulse* and Kronecker's delta, is a sequence that has amplitude equal to one at $n = 0$ and zero otherwise. When $\delta[n]$ is multiplied by another (arbitrary) sequence $x[n]$, the result $y[n] = x[n]\delta[n]$ can be denoted by $y[n] = x[0]\delta[n]$, indicating that $x[0]$ is the only non-zero value of the new sequence. Hence, $\delta[n]$ allows to analyze (or "sift") $x[n]$ when $n = 0$. Delaying the impulse by n_0 samples (the amplitude is one only at $n = n_0$ now) and multiplying by $x[n]$, one obtains

$$y[n] = x[n]\delta[n - n_0] = x[n - n_0]\delta[n - n_0]. \tag{A.107}$$

The capability of "sifting"[21] a signal $x[n]$ using impulses is very useful as discussed in Section 1.4.2. The continuous-time version $\delta(t)$ of $\delta[n]$ is discussed in the sequel. Many properties are shared by $\delta(t)$ and $\delta[n]$, but $\delta(t)$ is more mathematically involved.

A.29.2 Why defining the continuous-time impulse? Some motivation

Before describing its mathematical properties, some motivation for using continuous-time impulses, also called Dirac's delta, is presented.

A good motivation for the impulse $\delta(t)$ is the representational power of impulses in "density functions". In many situations it is useful to represent properties of discrete variables using densities. For example, how could one represent the discrete random variable (r. v.) corresponding to the result of a dice roll as a continuous r.v. Z and use a probability density function (PDF) instead of a probability mass function (PMF)? An impulse is capable of representing some "mass probability" in a PDF! Figure A.34 indicates the handy representation provided by impulses with area 1/6.

For illustrative purposes, it is common to plot the impulse length proportional to its area as in Figure A.34, but one should not take the area as the amplitude value. For example, as discussed in Section 1.13.1, the autocorrelation of a power signal at the origin is the average power. For AWGN with bilateral PSD $\mathcal{N}_0/2$, as indicated in Eq. (**??**), the power is infinite and its autocorrelation at the origin is an impulse with area $\mathcal{N}_0/2$. In other words, looking carelessly at the autocorrelation $R(\tau) = \mathcal{N}_0/2\delta(\tau)$ one could think that the average power is $\mathcal{N}_0/2$, but in fact $R(0) = \infty$.

[21] Note that it is not shifting but sifting. Sift means to analyze, as mentioned.

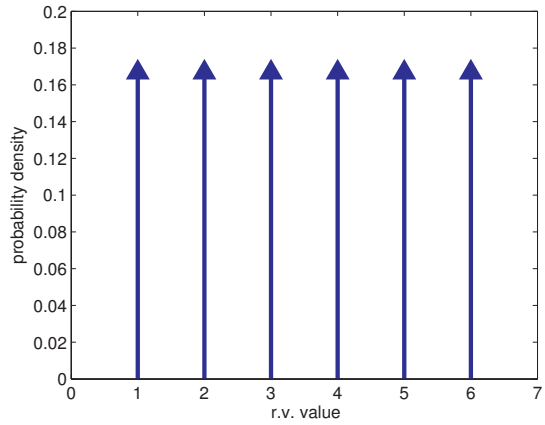


Figure A.34: *PDF for a dice roll result when the random variable is assumed to be continuous.*

Another example of using impulses in "densities" is when dealing with a Fourier transform $X(f)$. As indicated in Table 2.5, if $x(t)$ is in Volts, $X(f)$ has units of Volts/Hz. Hence, if $x(t)$ is periodic and has a representation as a Fourier series (analog to a PMF), one needs to use impulses to represent the spectrum of $x(t)$ as a Fourier transform (analog to a PDF). The following sections focus in briefly describing $\delta(t)$.

A.29.3 Definition of the continuous-time impulse as a limit

The impulse $\delta(t)$ can be obtained by considering a pulse $p(t)$ of duration Δ and amplitude $1/\Delta$:

$$\delta(t) = \lim_{\Delta \to 0} p(t). \tag{A.108}$$

Note that the area of $p(t)$ is always 1 and, as Δ goes to 0, its amplitude goes to ∞. It is also possible to obtain $\delta(t)$ as the limit of other functions, such as a Gaussian (see, e.g., [urlBMimp]).

A.29.4 Continuous-time impulse is a distribution, not a function

It is relatively tricky to deal with continuous-time impulses because they are not functions, but *distributions*, also called *generalized functions* [urlBMdis]. Impulses can be manipulated via linear operations such as derivative and integral, but nonlinear operations such as $\log(\delta(t))$ and $\delta^2(t)$ cannot be defined to have the usual properties and require specific algebras such as Colombeau's [urlBMcol].

While $\delta(t)$ is not a function, in most signal processing tasks it can be manipulated as such. In contrast, the discrete-time impulse $\delta[n]$ has no mathematical difficulties in its definition or manipulation. It is a regular sequence that has all 0 amplitude for all values of n but $n = 0$, which has amplitude 1. In other words, while the amplitude of $\delta(t)$ is not defined (infinite), in discrete-time the amplitude is simply $\delta[0] = 1$.

A.29.5 Mathematical properties of the continuous-time impulse

Sifting property

The sifting property of Eq. (A.107) also applies in continuous-time. This property states that multiplying $x(t)$ by a continuous-time impulse $\delta(t-t_0)$ results in $y(t) = x(t)\delta(t-t_0) = x(t_0)\delta(t-t_0)$ that is non-zero only at t_0 and $x(t_0)$ is the area of the impulse at t_0.

The sifting property is typically quoted as

$$x(t_0) = \int_{-\infty}^{\infty} x(t)\delta(t-t_0)dt. \tag{A.109}$$

when one wants to emphasize that $x(t_0)$ is a scalar and there is no more connection to time in this case. It is also possible to write

$$x(t) = \int_{-\infty}^{\infty} x(\tau)\delta(\tau-t)d\tau.$$

when one wants to emphasize that Eq. (A.109) can be repeatedly invoked, for all values of t_0 (represented as t in this case) and recover the complete signal $x(t)$.

In most applications of the sifting property, the integral is not used and the property is quoted as

$$x(t)\delta(t-t_0) = x(t_0)\delta(t-t_0),$$

which can be interpreted as: when a function $x(t)$ is multiplied by an impulse at t_0, all values of $x(t)$ other than $x(t_0)$ become zero and non-relevant, such that the final result is a single impulse of area $x(t_0)$ located at t_0.

Scaling property

Assuming a is a non-zero scalar, the scaling property of the impulse is

$$\delta(ax) = \frac{\delta(x)}{|a|}.$$

For example,

$$\delta(-3x) = \frac{\delta(x)}{3}.$$

The scaling property is important to understand the different areas of impulses when using Hz or rad/s as the unit for frequency (the independent variable). For example, the Fourier transform of a cosine $x(t) = 10\cos(6\pi t)$ of frequency 3 Hz and amplitude 10 is represented by

$$X(f) = 5\delta(f+3) + 5\delta(f-3)$$

or, equivalently,

$$X(\omega) = 10\pi\delta(\omega+3) + 10\pi\delta(\omega-3).$$

The areas are different by a factor of 2π for $X(f)$ and $X(\omega)$ because $\omega = 2\pi f$ and, by

the scaling property, $\delta(\omega) = \delta(f)/(2\pi)$. This can cause some confusion because when there are no impulses involved, it is trivial to convert a graph from $X(f)$ to $X(\omega)$ or vice-versa by scaling the abscissa (e. g., multiplying by 2π when converting from f to ω). However, all impulses should be scaled when the abscissa is modified.

A.29.6 Convolution with an impulse

When a signal $x(t)$ is convolved with an impulse $a\delta(t-t_0)$, the rule of thumb is to shift the origin of $x(t)$ to t_0 and scale it by the area a of the impulse. Mathematically:

$$x(t)*a\delta(t-t_0) = ax(t-t_0).$$

This is similar to $x[n]*a\delta[n-n_0]$ in discrete-time, but in discrete-time the scaling is by an amplitude a, not area.

A.30 System Properties

It is useful to characterize systems according to well-defined properties. Some of the most important properties are: linearity, time invariance, causality, stability, invertibility and memory. In the next subsections, it is assumed that the inputs x, x_1 and x_2 lead to the outputs $y = \mathcal{H}\{x\}$, $y_1 = \mathcal{H}\{x_1\}$ and $y_2 = \mathcal{H}\{x_2\}$, respectively (note it is not assumed the system $\mathcal{H}$ is LTI).

A.30.1 Linearity (additivity and homogeneity)

Linearity is a composition of two properties: additivity and homogeneity. A system is additive if it obeys

$$y_1 + y_2 = \mathcal{H}\{x_1 + x_2\}, \qquad \textbf{additive}$$

which indicates that its output to the sum of two signals $x_1 + x_2$ is equal to the sum of the two outputs y_1 and y_2 obtained independently for x_1 and x_2, respectively.

If a system is described by the relation $y[n] = 3x[n] + 2x[n-1]$ it is additive. The proof is as follows. When $x[n] = x_1[n] + x_2[n]$ one has

$$\begin{aligned} y[n] &= 3(x_1[n] + x_2[n]) + 2(x_1[n-1] + x_2[n-1]) \\ &= (3x_1[n] + 2x_1[n-1]) + (3x_2[n] + 2x_2[n-1]) = y_1[n] + y_2[n]. \end{aligned}$$

On the other hand, a system with $y[n] = 3(x[n])^2$ is not linear because

$$y[n] = 3(x_1[n] + x_2[n])^2 = 3(x_1[n])^2 + 3(x_2[n])^2 + 6x_1[n]x_2[n] \neq y_1[n] + y_2[n].$$

A system is homogeneous in case

$$\alpha y = \mathcal{H}\{\alpha x\}, \qquad \textbf{homogeneous}$$

where $\alpha \in \mathbb{C}$. In this case, if the input is multiplied by α, the output is multiplied by the same factor α. For example, the system $y[n] = 3(x[n])^2$ does not obey homogeneity

because

$$y[n] = 3(\alpha x[n])^2 = 3\alpha^2(x[n])^2 \neq \alpha 3(x[n])^2.$$

On the other hand, $y[n] = 3x[n] + 2x[n-1]$ can be proved to be homogeneous. Because it is also additive, $y[n] = 3x[n] + 2x[n-1]$ is a linear system.

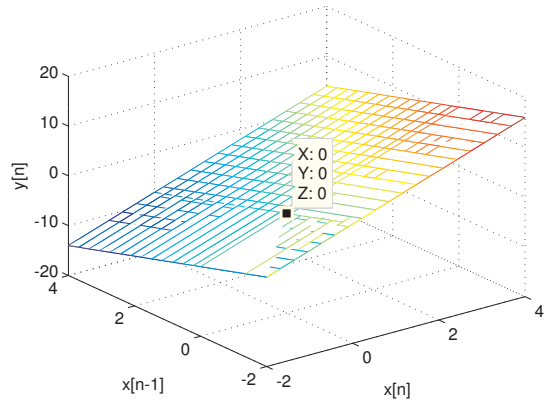


Figure A.35: *A 3-d representation of the system* $H(z) = 3 - 2z^{-1}$. *When* $x[n] = x[n-1] = 0$, *then* $y[n] = 3x[n] - 2x[n-1] = 0$, *as required for all linear systems.*

To provide a geometrical description of linearity, Figure A.35 depicts the possible outputs of the system $y[n] = 3x[n] - 2x[n-1]$ as tripes $(x[n], x[n-1], y[n])$. The plot shows the output when $x[n]$ and $x[n-1]$ are obtained from a grid of 400 Cartesian points in the range $[-2, 4] \times [-2, 4]$. It can be seen that, as required for a linear system, the input $(x[n], x[n-1]) = (0, 0)$ leads to $y[n] = 0$. In contrast, the system $y[n] = 3x[n] + 2x[n-1] + 1$ is not linear (neither additive nor homogeneous). A linear system is required to respond to an all zero input $x[n] = 0, \forall n$ with an all zero output $y[n] = 0, \forall n$.

Linearity is also called the principle of superposition and can be written as

$$\alpha y_1 + \beta y_2 = \mathcal{H}\{\alpha x_1 + \beta x_2\}, \qquad \text{(linearity or superposition)}$$

A.30.2 Time-invariance (or shift-invariance)

A system is time-invariant in case

$$y(t - t_0) = \mathcal{H}\{x(t - t_0)\}, \qquad \text{(time-invariant)}$$

or, in the discrete-time case: $y[n - n_0] = \mathcal{H}\{x[n - n_0]\}$, which is also called *shift-invariant*.

Time-invariance is equivalent to having the output delayed (or anticipated) by the same amount that the input was. A valid analogy is to consider that someone has a strict personal routine, which consists in waking-up at 7am, having breakfast at 8am, brushing teeth at 9am and so on. This person would be time-invariant if, in case

it wakes-up late, at 10am, it would keep the same routine with a delay of 3 hours: breakfast at 11am, brushing teeth at noon, etc.

A system $y[n] = nx[n]$ is not time-invariant. To prove it, one can observe that for an input $x[n] = \delta[n-2]$, the output is $y[n] = 2\delta[n-2]$ (because $x[n] = 0, n \neq 2$). Now, if one delays the input by $n_0 = 4$, to create a new input $x'[n] = x[n-n_0] = \delta[n-6]$ the output is $y[n] = 6\delta[n-6]$, which is different from having the output delayed by the same amount $n_0 = 4$ and get $y[n] = 2\delta[n-6]$, which should be expected for a time-invariant system.

It is easier to prove that a property is not respected because it suffices to give a single example that violates the property. However, to prove that a system obeys a property, this must be done in a general setting, to prove that it works for all signals. To practice this kind of general proof, let us continue with the system $y[n] = nx[n]$. The output to $x[n-n_0]$ is $(n-n_0)x[n-n_0]$ while time-invariance would require the output to be $nx[n-n_0]$. Before stating a general proof, it is convenient to work out a simple example.

As another example, the system $y(t) = 3x(t) + 2x(t-1)$ is time-invariant because for $x(t-t_0)$ the output is $y(t-t_0) = 3x(t-t_0) + 2x(t-1-t_0)$.

A.30.3 Memory

A memoryless system has the output at time t_0 depending only on the input at time t_0. This means that the output of a memoryless system does not depend of any past or future history of the input. For example, in analog filters, the resistor is a component that does not have memory, while the capacitor and inductor have. The current-voltage relation for a resistor is $i(t) = v(t)/R$ (memoryless) while for a capacitor it is $i(t) = Cdv(t)/dt$, which depends of the variation of $v(t)$ with time.

It is relatively easy to identify memory in the discrete-time domain because it can be related to the actual storage space called "memory" in computing systems. For example, the system $y[n] = \sqrt{x[n]} + \exp^{x[n]}$ is memoryless, while $y[n] = x[n] + 2x[n-1] - y[n-15]$ has to store one previous value ($x[n-1]$) of the input and 15 previous values ($y[n-1], y[n-2], \ldots, y[n-15]$) of the output. As illustrated in Listing **??**, all these values need to be stored in "memory" positions. The storage space for $x[n]$ and $y[n]$ is not accounted, even if in practice they require space. For example, assuming the values are represented as 4-bytes "floats", the previous system would require $4(1+15) = 64$ bytes of RAM "memory" to store the system memory.

A.30.4 Causality

A system is causal if the output at instant t_0 depends only on $x(t), t \leq t_0$, i. e., does not depend on future values of the input. All physical systems are causal because they cannot react to something that will occur in future.[22] It is possible to implement a non-causal system if the signal is processed "off-line", i. e., the samples are pre-stored and the processing is done without concerns to the real life time. The function filtfilt in

[22] A non-causal system could make a prediction in future, similar to a Nostradamus' prophecy!

Matlab/Octave is an example of non-causal processing that is very useful because can filter the magnitude of a signal without influencing its phase.

The system $y[n] = x[n+1]$ is non-causal because $y[n_0]$ depends on $x[n_0+1]$. On the other hand, the system $y[n] = x[n+1] - 0.5y[n+1]$ is causal because the time index can be rearranged (via a change in variable: $n' = n+1$) and the system rewritten as $y[n] = 2x[n] - 2y[n-1]$. As a general rule to check causality, one can look at the output value $y[n+n_{\max}]$ most advanced in time ($n_{\max} = 1$ in previous example), change variables with $n' = n + n_{\max}$, isolate $y[n]$ and check if it depends on future values of the input. For example, the system $y[n-3] = x[n-2] + y[n-4]$ is non-causal. This is clear if it is rewritten as $y[n] = x[n+1] + y[n-1]$.

A.30.5 Invertibility

A system is invertible if the input signal can be uniquely determined from the output. This is similar to the concept of an injective function. For example, $y(t) = kx(t)$ is invertible because $x(t) = (1/k)y(t)$, but $y(t) = x^2(t)$ is not. In general, a system $\mathcal{H}$ is invertible if it is possible to find another system $\mathcal{H}^{-1}$ such that their cascade replicates the input, as illustrated below:

$$x(t) \rightarrow \boxed{\mathcal{H}} \rightarrow y(t) \rightarrow \boxed{\mathcal{H}^{-1}} \rightarrow x(t).$$

A.30.6 Stability

The output of an unstable system can eventually grow unbounded and go to infinite. This is typically undesirable and suggests the so-called bounded-input, bounded-output (BIBO) stability. A system is BIBO stable if the output is bounded whenever the input is bounded. Note that if the input grows unbounded, then a stable system can have its output converging to infinite. In other words, looking only at the output one cannot state whether or not a system is stable. For example, the system $y(t) = 10x(t)$ is BIBO stable, but if $x(t) \to \infty$, then $y(t) \to \infty$. On the other hand, the system $y(t) = 1/x(t)$ is not BIBO stable, because the bounded input $x(t) = 0, \forall t$, leads to $y(t) \to \infty$.

A.30.7 Properties of Linear and time-invariant (LTI) systems

If an LTI system $\mathcal{H}$ is memoryless, its impulse response is non-zero only at $n = 0$ (or $t = 0$). In other words, $h[n] = 0, n \neq 0$. $\mathcal{H}$ is causal if and only if $h[n] = 0, n < 0$ (i. e., the output does not effectively start until the impulse $\delta[n]$ is applied at $n = 0$). If an LTI system $\mathcal{H}$ with impulse response $h[n]$ is invertible, the inverse system is also LTI and has an impulse response $h_i[n]$ that obeys $h[n] * h_i[n] = \delta[n]$. To be BIBO stable, the LTI must have the impulse response *absolutely summable* or *integrable*, i. e.

$$\sum_{n=-\infty}^{\infty} |h[n]| < \infty \quad \text{or} \quad \int_{-\infty}^{\infty} |h(t)| < \infty, \tag{A.110}$$

depending if the system is discrete or continuous in time, respectively.

Appendix B Useful Softwares and Programming Tricks

B.1 The GNU Radio (GR) Project

B.1.1 For those willing to stop reading and run code

Running GR on Windows is easier nowadays than when GR started, as discussed at [urlagrw]. But if it is an option, using GR on Linux is advised.[1]

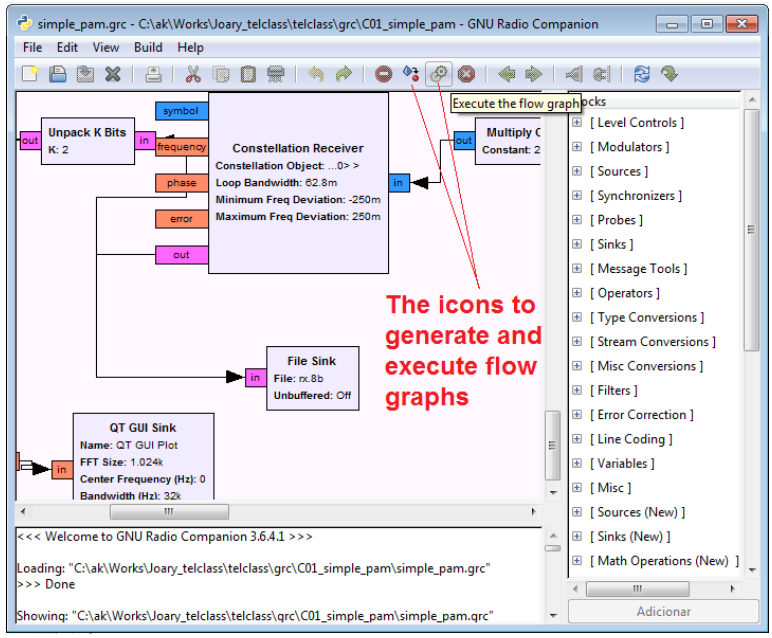


Figure B.1: *GNU Radio Companion interface highlighting the icons to execute the simulation.*

The easiest way to explore GNU Radio (GR) is to execute gnuradio-companion.py, the GNU Radio Companion (or GRC). The GRC is a GUI for GR that aims to make easier to get started with GR. On Windows, it is often located at `c:\Program Files\gnuradio\bin` and assuming this folder is part of your PATH and Python is properly installed, the script can be invoked with:

[1] In case you need (or want) to run GR on Windows, the main difficulty during the installation process is to keep the dependencies (Python libraries, etc.) compatible among themselves. For example, assume that you have Python version 2.7 installed, then you cannot use a library for Python version 3.3. The problem is that it is sometimes hard to find the file you are looking for on the Web. Try, for example, finding PyQwt-5.2.1-py2.7-x32-pyqt4.9.6-numpy1.6.2-1.exe, which is suggested at the Ettus Windows install page. If not found, you may end up going for PyQwt-5.2.1-py2.7-x32-pyqt4.9.6-numpy1.7.1.exe instead. But then, you must use the NumPy version 1.7.1 (numpy-1.7.1-win32-superpack-python2.7.exe) instead of the originally suggested 1.6.2. Pay attention on how the versions are encoded in the file names to be installed and make sure they match.

```
python gnuradio-companion.py
```

from a command prompt. An alternative is to use Windows Explorer to navigate to its folder and double-click the script icon. Figure B.1 illustrates GRC after loading a file (with extension grc).[2]

For those who are serious about using GR, the official web site is [urlagro]. A great source of information and cool projects is the *Comprehensive GR Archive Network* [urlagra].

B.1.2 Basic facts about GR

GR is a free software development toolkit that provides the signal processing runtime and processing blocks to implement software radios. It supports, Linux, Mac OS and Windows.

GR applications are primarily written using the Python programming language, while the supplied, performance-critical signal processing path is implemented in C++.

As used in this text, GR supports the development of signal processing algorithms using pre-recorded or generated data, avoiding the need for actual RF hardware.

A *block* in GR is actually a class implemented in C++. For example, a block called gr.quadrature_demod_cf corresponds to the class gr_quadrature_demod_cf implemented in C++. SWIG (*Simplified Wrapper and Interface Generator*) is a software layer that creates an interface between C++ and Python.

GR uses the following conventions for files:

- .h file for class declaration
- .cc file for class definition
- .i file defining how SWIG generates glue code binding the C++ class into Python

and for variables names:

- f - IEEE 754 single precision number in floating points (or *floats*)
- c - complex (pair of 32-bit floats)
- i - integer (32-bit signed integer)
- s - short (16-bit signed integer)
- b - byte (8-bit signed integer)

Much more information about code conventions is available at [urlagrc].

A very important aspect of GR is that it has a *Scheduler* that controls the flow of information. The user connects the blocks using a graph and the Scheduler takes care of sending information from a block to the next block(s) in the graph.

Blocks are connected together using the connect() method of the flow graph. The connect() method takes two parameters, the source endpoint and the destination endpoint, and creates a connection from the source to the destination. An endpoint

[2] One interesting thing is that, as indicated in Figure B.1, the icon to execute the flow graph has a gray color that sometimes leads novice users to consider it is disabled.

has two components: a signal processing block and a port number. The port number specifies which input or output port of the specified block is to be connected. In the general form, an endpoint is represented as a Python tuple like this: (block, port_number). When port_number is zero, the block may be used alone. For instance the following two expressions are equivalent:

```
fg.connect ((src1, 0), (dst, 1))
fg.connect (src1, (dst, 1))
```

The start() method forks one or more threads to run the computation described by the graph and returns control immediately to the caller.

Among other tasks, the Scheduler has to allocate buffers to accommodate the data that flows among the blocks. When developing simple applications, which rely only on existing blocks, most of the times the user does not need to understand much about the Scheduler. However, when developing new blocks in C++ or sophisticated applications, it is important to study the Scheduler and related tasks.

B.2 GR-Based SDR Hardware (USRP, RTL-SDR Dongle, etc.)

The availability of flexible and relatively low cost platforms has been increasing and can also be explored in advanced engineering projects, for teaching at schools or self-learning. This text describes just few platforms that rely on GNU Radio. The respective communities generated great material that is freely available and the reader is invited to use this text as an introduction and pointer to the web sites and discussion groups.

A typical hardware interface can be divided in three main system parts: the analog-to-digital (ADC) / digital-to-analog (DAC) conversions; the analog tuner, capable of up/down converting signals; and finally the transport, responsible for streaming the signal to/from the converter, constrained by the stream capability (S_c) of the connection interface (typically, USB or Ethernet).

Table B.1 compares four distinct SDR hardware platforms. In the past years, the Universal Software Radio Peripheral (USRP) has been the most popular hardware used with GNU Radio. It can be purchased from National Instruments (NI) or directly from Ettus (acquired by NI). NI commercializes USRP kits that include educational material. Table B.1 includes only two models: *N210* and the*B100* and assumes the current prices suggested by Ettus. Nowadays, there are cheaper yet powerful alternatives to USRPs such as the *Jawbreaker* board developed within the HackRF project. If only a SDR receiver (without a transmitter) is useful, a DVB-T Dongle is the cheapest alternative. Table B.1 lists the *Ezcap EZTV645* dongle, but there is a large number of models and a list of devices compatible with GNU Radio discussed at [urlartl] and [urlagra].

B.2.1 Universal Software Radio Peripheral (USRP)

The *Universal Software Radio Peripheral* (USRP) is a hardware device conceived to work with the GNU Radio project. The large majority of the USRP is *open source*, including schematics and layouts, for example. Figure B.2 illustrates the USRP through a block diagram.

Table B.1: *Examples of SDR Hardware Platforms.*

	U. N210	U. B100	Jawbreaker	EZTV645
F_s (Hz or Msps)	100	64	22	4
b (bits)	14	14	8	8
Δf (GHz)	0-6*	0-6*	0.03-6	0.064-1.7
S_c (Msps)	50	8	20	2.4
Price (US$)	1700	650	300	20
Obs.	full-duplex	full-duplex	half-duplex	just receiver

* Δf depends on the daughter boards used.

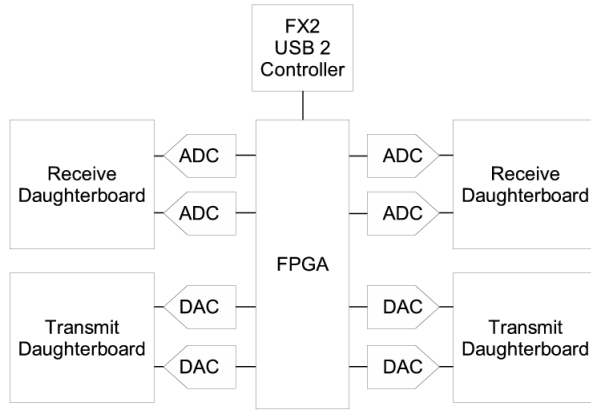


Figure B.2: *USRP block diagram.*

The USRP consists of a small motherboard containing up to four 12-bit 64M sample/sec analog-to-digital converters (ADCs), four 14-bit, 128M sample/sec digital-to-analog converters (DACs), a million gate-field programmable gate array (FPGA) and a programmable USB 2.0 controller. Each fully populated USRP motherboard supports four daughterboards, two for receive and two for transmit. RF front ends are implemented on the daughterboards. A variety of daughterboards is available to handle different frequency bands.

All ADCs and DACs are connected to the FPGA. In its original configuration, the FPGA implements *digital down-converters* (DDC) and *cascaded integrator-comb* (CIC) filters, while *digital up-converters* (DUC) are located in the AD9862 chips.

The USRP interfaces with the world through daughter boards, which are connected to the mentioned slots. There are several USRP daughter boards, with operating frequencies from 0 to 2.5 GHz. It is also possible to develop new daughter boards, for example, to connect baseband signals to the USRP.

B.3 Using GNU Radio and USRP

An experiment is described here to illustrate. Two USRP mother boards were used, each one connected to a FLEX 2400 daughter board, which work in the frequency range from 2.4 to 2.5 GHz. The goal is to transmit a given file (for example, a figure in the BMP format) using the GMSK modulation with a carrier frequency of 2.5 GHz. The second USRP receives the signal, demodulates, writes to disk and compares with the original file. The data flow graph depicted in Figure B.3 lists the processing blocks, as well as their corresponding output. Figure B.4 provides a illustration of the involved equipments. The first transmitter block reads the file, while its successor block ignores the first bytes (file header) and passes the remaining bytes to an "unpacker", which converts bytes into right-justified bits in each output byte. The GMSK modulator then sends complex symbols to the USRP, which converts them into a modulated signal. The inverse process takes place at the receiver. The extra Packed to Unpacked block is necessary due to the format adopted by the demodulator.

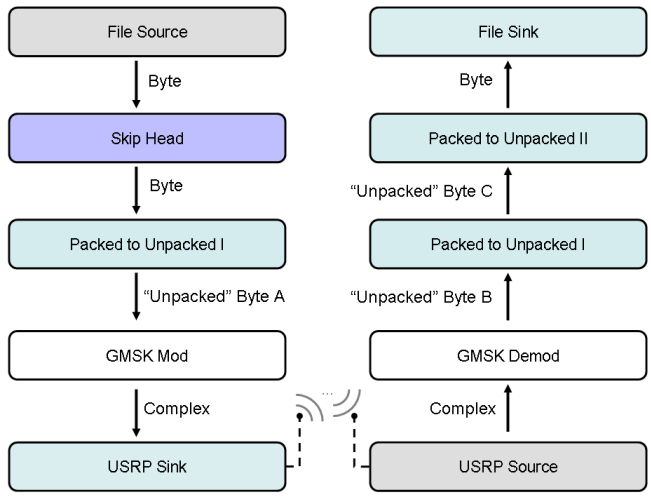


Figure B.3: *Block diagram describing a wireless system using GMSK. The blocks correspond to objects in Python that interface with GNU Radio code in C++ language.*

B.4 Matlab and Octave

For using the companion code, add the respective directories to the Matlab/Octave PATH with commands such as `addpath(genpath('C:/mydir/Code/'),'-end')` and `savepath`.

In spite of many similarities, Matlab and Octave are of course two distinct softwares. In fact, compatibility with Matlab is not the top priority in Octave's development. Hence, even if a function has the same name and syntax, the algorithms are typically implemented differently and may lead to discrepant results. Besides, Mathwords is strategically substituting functions by classes in new versions, such that functions such

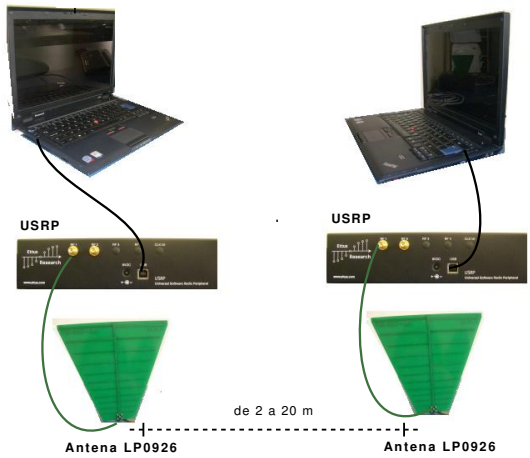


Figure B.4: *Experimental setup using two USRPs for getting familiar with GNU Radio.*

as butter (for designing a filter) are invoked after an object is created. This has been making Matlab and Octave increasingly different.

As another example of distinct behavior, fir1 in Matlab normalizes the filter using:

```
if First_Band
    b = b / sum(b);  % unity gain at DC
else
    ...
```

while fir1 in Octave uses:

```
  ## normalize filter magnitude
  if scale == 1
    ## find the middle of the first band edge
    if m(1) == 1, w_o = (f(2)-f(1))/2;
    else w_o = f(3) + (f(4)-f(3))/2;
    endif
    ## compute |h(w_o)|^-1
    renorm = 1/abs(polyval(b, exp(-1i*pi*w_o)));
    ## normalize the filter
    b = renorm*b;
  endif
```

Therefore, the command B=fir1(10,0.4) in Matlab generates a filter that has a gain of 1 at DC, while this command in Octave generates a filter with gain of 0.7557 dB at DC.

The bilinear function is a representative example of different syntaxes, which is a major problem when dealing simultaneously with Matlab and Octave. While Matlab uses the sampling frequency F_s, Octave uses the sampling period $T_s = 1/F_s$. Hence, for obtaining approximately the same results, the call [NUMd,DENd] = bilinear(NUM,DEN,Fs) in Matlab must be mapped to [NUMd,DENd] = bilinear(NUM,DEN,1/Fs) in Octave. Another example of discrepancy is the third argument of pwelch.m). Octave assumes it

is a percentage while Matlab assumes it is the number of samples. The Web has good sites comparing the two softwares, such as [urlomoc].

B.4.1 Octave Installation

The primary source for Octave is [urlooct]. But, for better organization, most of Octave's packages (called toolboxes in Matlab's jargon) are kept separated, at Octave Forge: [urloofo]. Therefore, after installing Octave itself, it may be necessary to download and install its packages (toolboxes).

After installing Octave, from its prompt, the command ver can be used to verify not only the Octave version but also its toolboxes. For example, the toolbox signal is required to execute several of the companion scripts. When using Linux, it is possible to use a command such as [urloins]:

```
sudo apt-get install $( apt-cache search octave-forge | awk '{printf
    $1; printf " "}' )
```

to install the packages.

An alternative to this two-steps installation procedure is to obtain a complete "distribution" of Octave, with all (or most) packages of interest already incorporated. For example, installing Octave on Windows can be simplified by following the guidelines at [urlowin], which indicate how to install the Octave executable from a single file and then most of its packages from a second compressed file.

It is suggested to add the folder with the companion functions in your Octave PATH using addpath. For example, to add the folder

```
C:\Code\MatlabOctaveFunctions
```

in a Windows machine, avoid the backslash using

```
addpath("C:/Code/MatlabOctaveFunctions","-end")
```

or

```
addpath(["C:" filesep "Code" filesep "MatlabOctaveFunctions"],"-end")
```

and use savepath to save the modifications.

Make sure the command sound (or soundsc) is working. You may need to install a sound player and inform Octave of its PATH using an Octave's global environment variable called sound_play_utility. A good strategy is to setup this variable at the .octaverc file with something like global sound_play_utility = "c:/Program Files/VideoLan/VLC/vlc", but changing the PATH to your installed audio player [url1rec].

Then keep your Octave installation up to date. For installing a package, for example, from the Octave prompt it is possible to run pkg install -forge signal to install the signal package.

Another point worth mentioning is that some packages, such as the Fixed Point, are not part of the ones supported by Octave Forge and need to be installed individually (taking in account their compatibility with the installed version of Octave).

B.5 Manipulating signals stored in files

B.5.1 Hex / Binary File Editors

It is important to be able to visualize the contents of files. For this task one can use hex (hexadecimal) editors such as [urlBMhex]. This is especially useful for binary files, but it is interesting for text files too.

The companion software FileViewer can also be used. It is a Java program that can be invoked, for example, from command line with

```
java -jar FileViewer.jar
```

This software was used to analyze a Windows (or DOS) text file called `fileviewer_test.txt`, which has 52 bytes and the following contents:

```
A simple text
with 52 bytes:
To be or not to be?
```

Figure B.5 is a screenshot of the file, showing each byte in hexadecimal in the main panel. One can see that the first letter, a capital 'A' is represented by the byte 0x41 in hexadecimal (the prefix 0x indicates an hexadecimal number). Clicking on the button "Decimal" (at the left of "Hexa"), one can note that 0x41 is 65 in decimal.

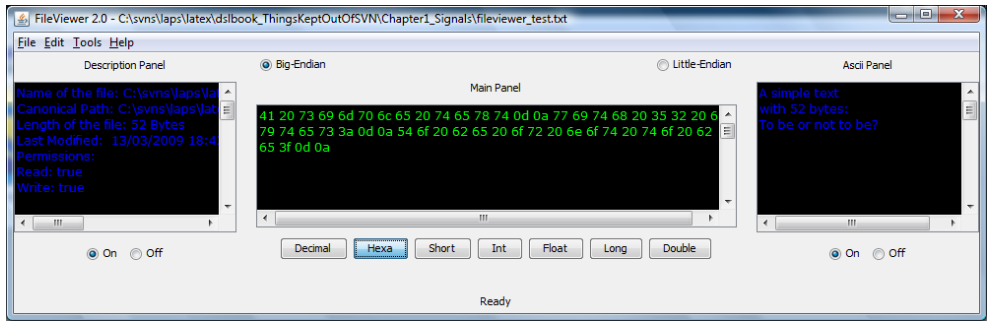


Figure B.5: *Screenshot of the FileViewer software.*

These plain text files are known as ASCII files, where ASCII stands for American Standard Code for Information Interchange. These files do not have the many extra bytes that a modern text editor (e.g, Microsoft Word and Open Office Writer) inserts to represent extra information, such as font size, color, etc.

B.5.2 ASCII Text Files: Unix/Linux versus Windows

There is an important distinction between how Windows and Unix'es (e.g., Linux) interpret text files. Details are provided in [urlBMlfc] and other sites. In summary, Windows uses two ASCII characters (CR and LF) to indicate a new line, while Linux uses LF alone. In hexadecimal these ASCII characters are represented by 0x0A (LF or line feed) and 0x0D (CR or carriage return). Note in Figure B.5 that each line is terminated by a pair of bytes 0x0D 0x0A because `fileviewer_test.txt` is a Windows

text file. In fact, when using a regular text editor, the number of visible letters in this file is 46, but each of the three new lines demand 2 extra (invisible) bytes, so the total file size is $46 + 3 \times 2 = 52$ bytes.

The FileViewer software can be used to convert between Linux and Windows text files. In menu Tools / Converter one can find the dialog window depicted in Figure B.6.

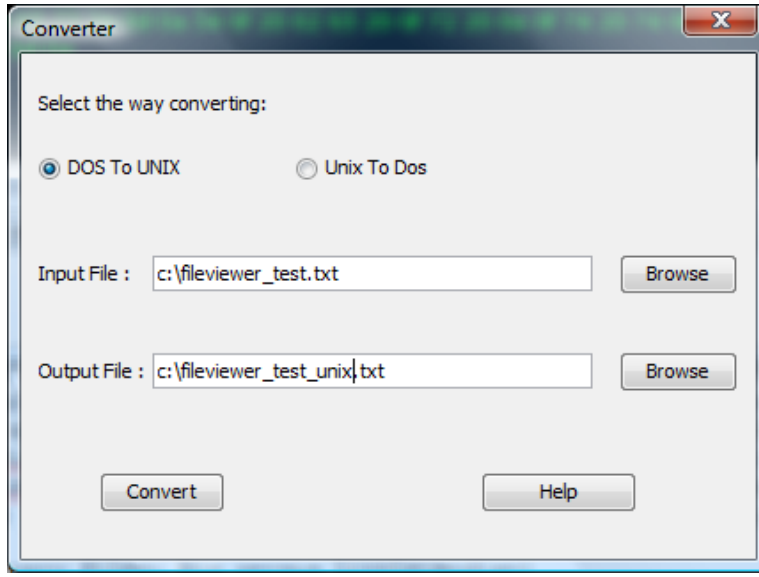


Figure B.6: *Screenshot of the FileViewer dialog window that allows to convert between Linux and Windows text files.*

The result of executing the instruction indicated in Figure B.6 is indicated in Figure B.7, which shows in the Description Panel that the file size was reduced to 49 bytes. The Main Panel also indicates that the pair 0x0D 0x0A was substituted by the byte 0x0A, which is enough to represent a new line on Linux.

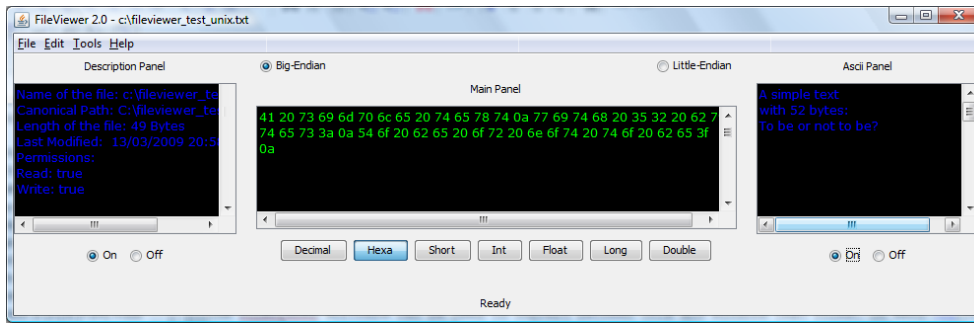


Figure B.7: *Screenshot of the FileViewer software showing the result of converting the Windows file of Figure B.5 to the Linux format. Note that the new lines are signaled by the byte 0x0A.*

B.5.3 Binary Files: Big versus Little-endian

Another important distinction is the little-endian versus the big-endian (endian, not indian) representation of variables composed of more than one byte. In the big-endian representation, the most significant byte (MSB) is the first. In little-endian, the least significant byte (LSB) is the first. For example, suppose one is dealing with a binary file that stores values represented by short variables. A short is represented by two bytes in most languages, such as C and Java. The first two bytes of this file are 0x02 and 0x01. Assuming big-endian, they would be interpreted as the number 513 in decimal (0x0201 in hexa), while in little-endian this pair of bytes corresponds to 258 in decimal (0x0102 in hexa). Note that in this example both numbers are positive. When dealing with negative numbers, one needs to pay attention to the adopted representation. Negative integers are typically represented in two's-complement arithmetic [urlBMwtc].

Figure B.8 illustrates the result of interpreting the bytes in the file of Figure B.7 as short. From Figure B.7 one can see that the first two bytes are 0x41 0x20, which (in this order 0x4120) corresponds to 16,672 in decimal. Using the option in FileViewer to interpret these bytes as little-endian would lead to 8,256 (i.e., 0x2041).

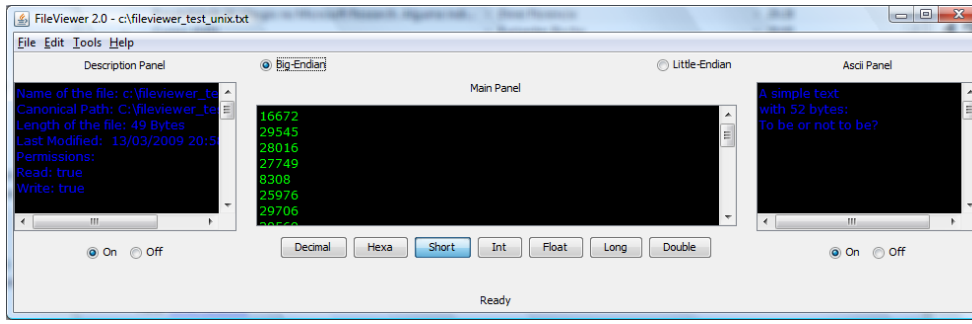


Figure B.8: *Interpretation of the file in Figure B.7 as short (2-bytes) big-endian elements.*

The companion C code laps_swap.c can swap bytes and create little-endian files from big-endian files and vice-versa. Note that the code simply reverses the order of the bytes. Its usage requires knowing the number of bytes of each element. For example, short corresponds to 2 bytes, float to 4, etc.

It is worth considering the following situation: Java always uses big-endian, even in computers that adopt a little-endian architecture such as the PC. After compiled, laps_swap.c can be used to convert binary big-endian files generated by a Java program into little-endian files to be interpreted by C programs executed in a PC. Assuming double (8 bytes per element), the command

laps_swap.exe inputFile.bin outputFile.bin 8

will swap the bytes of inputFile.bin (assumed to be big-endian) and create a file called outputFile.bin. FileViewer can be used to check the conversion.

B.5.4 Some Useful Code to Manipulate Files

Matlab/Octave language

Assume we have a vector **z** in Matlab/Octave. The command save can be used to create a file somefile.txt and store **z** as follows

```
save -ascii 'somefile.txt' z
```

However, save does not allow much control on formatting the output numbers. For improved formatting, one can use the fprintf function:

```
z=rand(10,1); %generate a random vector
fp = fopen('somefile.txt','wt'); %open the file for writing
fprintf(fp,'%12.8f\n',z); %save according to the specified format
fclose(fp);  %close the file
```

The command load can be used to read in data. For example:

```
x = load('anotherfile.txt');
```

would create a variable **x** with the contents of anotherfile.txt.

Text (ASCII) files have a relatively large size when compared to binary files and require parsing. Hence, the interface between other programs and Matlab/Octave is often more efficient when done via binary files. The following code illustrates how to read a binary file with samples stored as 16 bits.

```
fp=fopen('somefile.bin','rb'); %open for reading in binary
x=fread(fp,Inf,'int16'); %read all samples as signed 16-bits
fclose(fp);  %close the file
```

When dealing with binary files one should inform the endianness when opening the file. When Matlab/Octave is running on PCs, its default is little-endian. For example, fid = fopen('littleendianfile.bin','rb'); opens a file and prepares to interpret its data in little-endian. In contrast, fid = fopen('bigendianfile.bin','rb','b'); assumes the file is big-endian. In any case, the command [filename, permission, machineformat, encoding]=fopen(fid) retrieves the information associated to the opened fid variable.

In case of dealing with matrices representing images, one may need some extra commands such as, for example:

```
% Read image of size rows x cols stored in unsigned chars (no header)
 rows = 256; cols = 256; %number of pixels in rows / columns
 fid = fopen('myimage.bin','r'); %open the file
 out = fread(fid,[rows,cols],'uchar'); %read the file
 out = flipud(rot90(out)); %reorganize the data
 fclose(fid); %close the file
```

C language

The goal of this section is to illustrate how to read into Matlab/Octave data stored in files that were written by C programs and vice-versa. The example below shows how to generate a cosine signal in C, write it into a file and load in Matlab/Octave.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
/* Program to generate a cosine of 200 Hz and amplitude 10 */
#define D 5 /* duration in seconds */
int main() {
  FILE *fp;
  char name[2048]="cosine.txt";
  double sample;
  int i; /*counter*/
  double freq_cosine = 200;
  double Fs = 8000; /*sampling frequency*/
  int n; /* total number of samples */
  double Ts; /*  sampling period */
  double t; /* current time */

  n = (int) (Fs * D);
  Ts = 1/Fs;
  fp = fopen(name, "w"); /* open the output file */
  if (fp == NULL) {
    printf("Error opening file %s\n", name);
    exit(-1); /* exit if error */
  }
  /* write all samples */
  for (i=0; i<n; i++) {
    t = i * Ts; /* calculate current time */
    sample = 10.0*cos(2.0*M_PI*freq_cosine*t);
    fprintf(fp, "%f\n", sample);
  }
  fclose(fp); /* close file */
  printf("Wrote %d samples in file %s\n",n,name);
}
```

After compiling, executing the program and generating the file cosine.txt, it is possible to load it in Matlab/Octave, listen to the signal and observe its spectrogram with the commands:

```
x = load('cosine.txt');
soundsc(x)
specgram(x,[],8000);
colorbar
```

Note that the sampling frequency 8,000 Hz is informed to the specgram function for proper labeling of the axes.

Considering that the text file somefile.txt (see Section B.5.4) should be processed in a C program, the following code illustrates how to read it. Note that the value of N should be changed according to your needs.

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10 /*number of samples in file*/
int main() {
  FILE *fp; /*file pointer*/
```

```
  char name[2048]="somefile.txt";
  double signal[N];
  int i; /*counter*/
  fp = fopen(name, "r"); /* open file for reading */
  if (fp == NULL) {
    printf("Error opening file %s\n", name);
    exit(-1); /* exit if error */
  }
  for (i=0; i<N; i++) {
    /* note the use of "%lf" because the elements are double: */
    fscanf(fp, "%lf\n", &signal[i]); /*read N samples*/
  }
  /*from here you can manipulate signal*/
  fclose(fp); /* close file */
  printf("Finished reading %d samples from file %s",N,name);
}
```

Java language

Java is a modern language and has many features to deal with files. Listing B.1 illustrates writing and reading a file with float samples using a small header, which informs the number of samples. Recall that Java always uses big-endian. The output file has a size of 22 bytes, corresponding to 5 float (4-bytes) samples plus the 2-bytes header.

Listing B.1: *Java_Language/FileManipulation.java*

```
/** Example of reading and writing binary (float) files.*/
import java.io.*;

public class FileManipulation {

  public static void main(String[] args) {
  String fileName = "floatsamples.bin";
  float[] x = {-2,-1,0,1,2};
  writeVectorToBinaryFile(fileName, x);
  float[] y = readVectorFromBinaryFile(fileName);
  System.out.println("Samples obtained from file:");
  for (int i=0;i<y.length;i++) {
    System.out.print(y[i] + " ");
  }
  System.out.println("\nFinished writing and reading "+ fileName);
  }

  /** Method to write a binary file*/
  public static void writeVectorToBinaryFile(String fileName,
                                             float[] vector) {
    try {
      File file = new File(fileName);
      FileOutputStream fileOutputStream = new FileOutputStream(file);
      DataOutputStream dataOutputStream = new DataOutputStream(
          fileOutputStream);
```

```
      //create a small header with the number of samples
      dataOutputStream.writeShort( (short) vector.length);
      for (int i = 0; i < vector.length; i++) {
        dataOutputStream.writeFloat(vector[i]);
      }
      dataOutputStream.close();
      fileOutputStream.close();
    }
    catch (IOException e) {
      e.printStackTrace();
      System.err.println("Problem writing file " + fileName);
    }
  }

  /** Method to read a binary file*/
  public static float[] readVectorFromBinaryFile(String fileName) {
    float[] x = null;
    try {
      File file = new File(fileName);
      FileInputStream fileInputStream = new FileInputStream(file);
      DataInputStream dataInputStream = new DataInputStream(
          fileInputStream);
      //read the number of samples (in header)
      int numSamples = (int) dataInputStream.readShort();
      x = new float[numSamples];
      for (int i = 0; i < numSamples; i++) {
          x[i] = dataInputStream.readFloat();
      }
      fileInputStream.close();
    }
    catch (IOException e) {
      e.printStackTrace();
      System.err.println("Problem reading file " + fileName);
    }
    return x;
  }

}
```

B.5.5 Interpreting binary files with complex headers

Some binary files are *headerless* (also called *raw* files), i.e., they do not contain any extra information in a header, just the data (for example, the samples of a signal). However, most binary files start with a header. If the header information is encoded in the same format as the data itself, it is relatively easy to inspect the file contents using a software such as FileViewer. However, if the header is composed by a mix of float, short, char variables, etc., then the interpretation is trickier.

As an example, assume the file floatsamples.bin generated with Listing B.1. Figure B.9 shows the floatsamples.bin file contents interpreted as hexadecimal bytes.

The data is in float format but the header is a 2-bytes short. Hence, reading the file

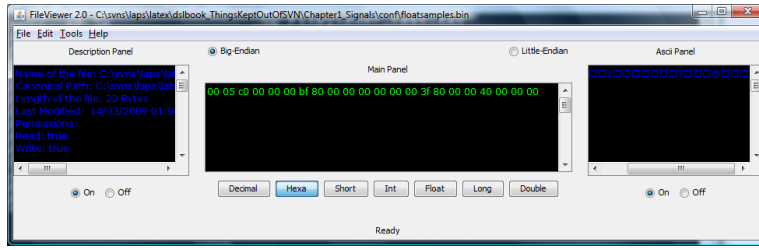


Figure B.9: *Contents in hexadecimal of file* floatsamples.bin *generated with Listing B.1.*

as float will get all numbers wrong as shown in Figure B.10. The correct would be to skip the header and then read the data.

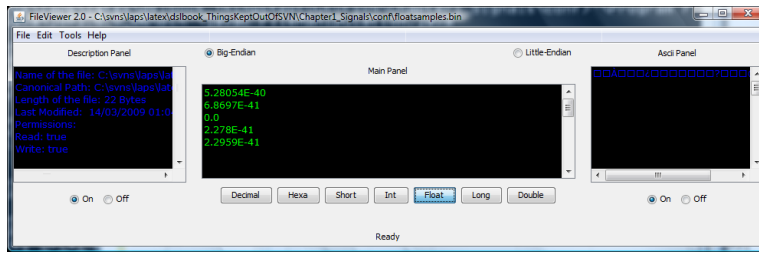


Figure B.10: *Interpretation as big-endian floats of file* floatsamples.bin *generated with Listing B.1. The numbers are wrong (the data after the header is* $-2, -1, 0, 1, 2$*) because the reading is not aligned with the end of the 2-bytes header.*

The companion code laps_dump.c can be helpful in situations where the header is heterogeneous. One can study its source code to understand how to simultaneously visualize all interpretations (short, float, etc.). For example, the interpretation of the big-endian file floatsamples.bin generated with Listing B.1, which is not trivial on a little-endian PC, can be performed as follows.

Recall that the useful information is the sequence $5, -2.0, -1.0, 0.0, 1.0, 2.0$. The command

laps_dump.exe floatsamples.bin swap

provides the output below. Each line indicates distinct interpretations (char, short, etc.) assuming the element starts at the corresponding byte. For example, the second line below indicates that the corresponding byte is 5, its ASCII code (fourth column) is a special character, and the fifth column interprets this byte as the beginning of a short variable composed by the pair of bytes 0x05C0, which corresponds to 1472 in decimal. Note that the numbers of interest in this case are inside frame boxes. The file size is 22 bytes, but after reading 16 bytes, the code tries to read the next double (8 bytes) and aborts the execution because there are only 6 bytes remaining. This is the reason for not observing the last element 2.0. Typically the last bytes will be missed.

Each columns represents:

```
<byte counter> <decimal char> <hexa char> <(ASCII)> ...
  <short> <int> <long int> <float> <double>
```

1 0 0 ( ) 5 376832 376832 5.280541e-40 7.996359e-309
2 5 5 (?) 1472 96468992 96468992 1.805559e-35 5.509016e-281
3 192 c0 ( ) -16384 -1073741824 -1073741824 -2.000000e+00 -2.000001e+00
4 0 0 ( ) 0 191 191 2.676480e-43 4.063622e-312
5 0 0 ( ) 0 49024 49024 6.869726e-41 1.040287e-309
6 0 0 ( ) 191 12550144 12550144 1.758650e-38 4.485749e-305
7 191 bf ( ) -16512 -1082130432 -1082130432 -1.000000e+00 -7.812500e-03
8 128 80 (?) -32768 -2147483648 -2147483648 -0.000000e+00 -3.112614e-322
9 0 0 ( ) 0 0 0 0.000000e+00 8.031531e-320
10 0 0 ( ) 0 0 0 0.000000e+00 2.056072e-317
11 0 0 ( ) 0 0 0 0.000000e+00 5.263544e-315
12 0 0 ( ) 0 63 63 8.828180e-44 1.347467e-312
13 0 0 ( ) 0 16256 16256 2.277951e-41 3.449516e-310
14 0 0 ( ) 63 4161536 4161536 5.831554e-39 1.752246e-307
15 63 3f (?) 16256 1065353216 1065353216 1.000000e+00 7.812502e-03
16 128 80 (?) -32768 -2147483584 -2147483584 -8.968310e-44
16 bytes read from file floatsamples.bin

Another example is the interpretation of a binary file generated by HCopy, which is part of the HTK package [urlBMhtk], widely used in speech recognition. Using the HList tool to interpret a file htk_file.bin with the command HList -t htk_file.bin leads to the output:

```
---------------------- Target -----------------------
  Sample Bytes:  96       Sample Kind:   MFCC
  Num Comps:     24       Sample Period: 6439.9 us
  Num Samples:   855      File Format:   HTK
```

HTK assumes the file content is a multi-variate time series with 855 samples of dimension 24 each. Because each of these 24 elements is represented by a 4-bytes float, the total size of a sample is 96 bytes. The sample period of 6439.9 microseconds indicates that the sampling frequency was $F_s = 10^6/6439.9 = 155.28$ Hz. MFCC is an identifier that is internally represented by a 2-bytes short of value 6 in this case (MFCC is typically represented by the short 838 in HTK) and indicates the kind of parameters in this time series. Figure B.11 shows the first bytes of the file htk_file.bin. The HTK manual informs that this header is composed by 12 bytes and the file is big-endian (by default). The first element is a four-bytes integer (int) that indicates the number of samples (855 in this case). The second header element is again an int that indicates the sample period in the unit of 10^{-7} seconds. The third element is a 2-bytes short that informs the sample size in bytes (96 in this case) and the last element is another short that specifies the kind of parameters (6 in this case).

Using the command
laps_dump.exe htk_file.bin swap
allows to investigate the contents of the file. The elements of interest are surrounded by frame boxes in the output below and can be compared to the previous output of the HList command.

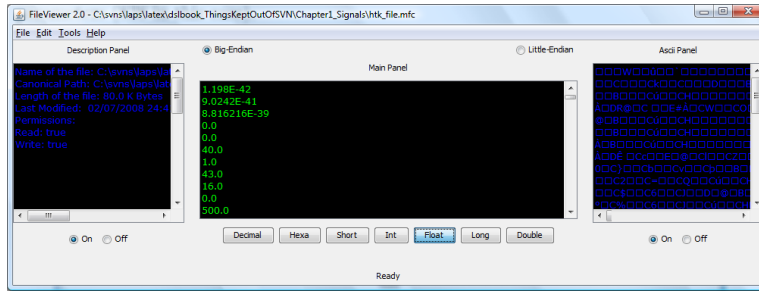


Figure B.11: *Contents interpreted as floats of big-endian file* htk_file.bin *generated with HTK. Note the first 12 bytes (3 floats) correspond to the header.*

Each columns represents:

```
<byte counter> <decimal char> <hexa char> <(ASCII)> ...
  <short> <int> <long int> <float> <double>
```

1 0 0 ( ) 0 855 855 1.198110e-42 1.814306e-311
2 0 0 ( ) 3 218880 218880 3.067162e-40 4.644624e-309
3 3 3 (?) 855 56033280 56033280 6.318282e-37 1.440497e-292
4 87 57 (W) 22272 1459618043 1459618043 1.407417e+14 1.202742e+111
5 0 0 ( ) 0 64399 64399 9.024222e-41 1.366544e-309
6 0 0 ( ) 251 16486144 16486144 2.310201e-38 6.279160e-304
7 251 fb ( ) -1137 -74514336 -74514336 -1.485012e+36 -1.475190e+287
8 143 8f ( ) -28928 -1895800832 -1895800832 -6.329376e-30 -2.011753e-236
9 0 0 ( ) 96 6291462 6291462 8.816216e-39 7.120277e-307
10 96 60 (‘) 24576 1610614272 1610614272 3.690024e+19 2.685490e+154
11 0 0 ( ) 6 393216 393216 5.510130e-40 8.344027e-309
12 6 6 (?) 1536 100663296 100663296 2.407412e-35 8.814426e-280
13 0 0 ( ) 0 0 0 0.000000e+00 0.000000e+00
14 0 0 ( ) 0 0 0 0.000000e+00 3.260833e-322
15 0 0 ( ) 0 0 0 0.000000e+00 8.363543e-320
16 0 0 ( ) 0 0 0 0.000000e+00 2.141067e-317
17 0 0 ( ) 0 0 0 0.000000e+00 5.481132e-315
18 0 0 ( ) 0 66 66 9.248570e-44 1.403170e-312
19 0 0 ( ) 0 16928 16928 2.372118e-41 3.592114e-310
20 0 0 ( ) 66 4333568 4333568 6.072622e-39 2.016473e-307
21 66 42 (B) 16928 1109393408 1109393408 4.000000e+01 3.435975e+10
22 32 20 ( ) 8192 536870975 536870975 1.084210e-19 1.491758e-154
23 0 0 ( ) 0 16256 16256 2.277951e-41 3.449516e-310
24 0 0 ( ) 63 4161536 4161536 5.831554e-39 1.752246e-307
25 63 3f (?) 16256 1065353216 1065353216 1.000000e+00 7.812502e-03

Reading and writing HTK files. By default, HTK writes all files as big-endian. Matlab/Octave reads files according to the native architecture of the computer. Hence,

on PCs Matlab reads/writes little-endian. For example, if you open a binary file with fid = fopen('testhtk.mfc','rb');, Matlab on a PC will read it as little-endian. On the other hand, fid = fopen('testhtk.mfc','rb','b');, reads as big-endian. A file in HTK format has a header with 12 bytes. You can write Matlab functions to read and write a file with MFCC parameters according to HTK's format. To test, use a file generated by HCopy and compare the results of your functions with HList. To help you with parsing the header, the code below shows how to read a HTK file with Java, which always uses big-endian (even on PCs).

```
public static float[][] getParametersFromFile(String hTKFileName)
    {
    float[][] fparameters = null;
    try {
        FileInputStream fileInputStream =
            new FileInputStream(new File(hTKFileName));
        DataInputStream dataInputStream =
            new DataInputStream(fileInputStream);

        int nSamples = dataInputStream.readInt(); //4 bytes
        int sampPeriod = dataInputStream.readInt(); //4 bytes
        short sampSize = dataInputStream.readShort(); //2 bytes
        short parmKind = dataInputStream.readShort(); //2 bytes
        //parmKind = 838 corresponds to MFCC_E_D_A

        //allocate space (assume float, 4 bytes per number)
        int nspaceDimension = sampSize / 4;
        fparameters = new float[nSamples][nspaceDimension];
        for (int i = 0; i < fparameters.length; i++) {
            for (int j = 0; j < nspaceDimension; j++) {
                fparameters[i][j] = dataInputStream.readFloat();
            }
        }
        fileInputStream.close();
        dataInputStream.close();
    }
    catch (IOException e) {
        e.printStackTrace();
        return null;
    }
    return fparameters;
}
```

Glossary

Text Conventions

1. Definitions are indicated by $\triangleq$.
2. When discussing digital modulation, N is the number of dimensions (e.g., $N = 2$ for QAM), M is the number of constellation symbols, S is the number of samples per symbol (also called *oversampling factor*) and $b = \log_2 M$ is the number of bits per symbol.
3. Random variables are upper-case mathsf in Latex, e.g. X, Y and Z.
4. Random signals are lower-case mathsf in Latex, e.g. $\mathsf{x}[n]$
5. Random processes use cal in Latex, e.g., $\mathcal{X}[n]$ and $\mathcal{X}(t)$
6. Sets also use cal in Latex, e.g., $\mathcal{C} = \mathcal{A} \cup \mathcal{B}$. The context should be enough to disambiguate random processes and sets.
7. We call D/C (discrete/continuous) and C/D (continuous/discrete) the theoretical models used as stages of the practical D/A and A/D conversions, respectively.
8. Analog (unquantized continuous-time) signals $x(t)$. A sampled signal is indicated with a subscript s, e.g., $x_s(t)$. Discrete-time $x[n]$ and digital $x^q[n]$, sampled and quantized $x_s^q(t)$
9. Distinguish the signal $x(t)$ (or x[n]) from a specific sample $x(t_0)$. At least in the beginning of the text.
10. Units are taken from the international system: length in meters, time in seconds, $x(t)$ and $x[n]$ in Volts, $X(f)$ in Volts/Hz and PSDs in Watts/Hz.
11. $\mathbb{E}$ is the expected value and Var is the variance.
12. The superscripts *, T and H denote complex conjugate, transpose and Hermitian, respectively. For matrices, $\mathbf{A}^H = (\mathbf{A}^*)^T$.
13. Two Latin sentences will be used: i.e. (id est, which means "that is") and e.g. (exempli gratia, which means "for example")
14. Whenever possible, constants and variables will be represented by upper and lower case letters, respectively

15. Vectors are bold lower-case (e.g., $\mathbf{x}$) and matrices are bold upper-case (e.g., $\mathbf{X}$). When dealing with transforms, a bold upper-case letter is also used. For example, the vector $\mathbf{X}$ is the transform of $\mathbf{x}$. The context should be enough to disambiguate transforms and matrices.

16. As in Matlab/Octave, the column m of a $M \times N$ matrix A is represented by $A(:, m)$, with $m = 0, \ldots, M-1$. Similarly, $A(n, :)$ denotes the n-th row, with $n = 0, \ldots, N-1$. Note that the first index in Matlab/Octaveis 1.

17. Lower case can eventually denote frequency-domain variables

18. $|\cdot|$ magnitude or absolute value of a complex scalar.

19. ang$(\cdot)$ angle of a complex scalar

20. $\|\mathbf{x}\|$ is the norm of vector $\mathbf{x}$

21. Estimates are indicated by a hat over the symbol (e.g., $\hat{x}$)

22. diag is the main diagonal of a matrix

23. A matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ can be denoted using Matlab/Octave syntax as $[a, b; c, d]$.

24. The (i, j)-th element of a matrix $\mathbf{A}$, at row i and column j, is represented by $a_{i,j}$.

25. When the value of a variable x is given in dB it will be denoted with a subscript x_{dB}.

26. In multicarrier systems, subscripts denote the user n while the superscripts denote the tone k

Table 2: *Nomenclature of special frequencies.*

Symbol	Description	Reference
ω_c or f_c	Cutoff frequency: where gain falls by $1/\sqrt{2}$ (-3 dB)	Figure ??
ω_n	Natural frequency, for example, of a resonator	Figure ??
ω_0	Center frequency of a pole	Figure ??
f_p	Passband frequency	Figure ??
f_r	Stopband (or rejection) frequency	Figure ??
F_s	Sampling frequency (typically in Hz or samples per sec.)	Eq. (1.5)
$F_s/2$	Nyquist (or folding) frequency	Eq. (2.31)

Main Abbreviations

- ADSL - Asymmetric digital subscriber line
- AGC - Automatic gain control
- AM - Amplitude modulation
- AR - Auto-regressive
- BER - Bit error rate
- BW - Channel bandwidth
- CO - Central office
- CPE - Customer premises equipment
- CR - Cognitive radio
- DCT - Discrete cosine transform
- DFT - Discrete Fourier transform
- DSB - Double-sideband
- DSP - Digital signal processing
- DTFS - Discrete-time Fourier series
- DTFT - Discrete-time Fourier transform
- ESD - Energy spectral density
- FA - Fixed-margin
- FB - Frequency burst
- FDD - Frequency division duplex
- FDM - Frequency division multiplexing
- FDMA - Frequency division multiple access
- GMSK - Gaussian minimum-shift keying
- GR - GNU Radio
- ISI - Intersymbol interference
- LPC - Linear predictive coding
- LSB - Least significant bit
- MA - Margin-adaptive
- MAP - Maximum a posteriori
- MF - Matched filter
- ML - Maximum likelihood
- MSB - Most significant bit
- NRZ - Non-return-to-zero
- PA - Power-adaptive
- PAM - Pulse amplitude modulation
- PBX - Private branch exchange
- PHY - Physical layer
- PLL - Phase-locked loop
- POTS - Plain old telephony system
- PSD - Power spectral density
- RA - Rate-adaptive
- RF - Radio-frequency signals
- ROC - Region of convergence

- Rx - Receiver
- RZ - Return-to-zero
- SER - Symbol error rate
- SNR - Signal to noise ratio: the signal power divided by the noise power.
- SSB - Single-sideband
- STFT - Short-time Fourier transform
- SVD - Singular value decomposition
- TDD - Time-division duplex
- TDM - Time-division multiplex
- TDMA - Time-division multiple access
- Tx - Transmitter
- USRP - Universal software radio peripheral
- ZFE - Zero-forcing equalization

Main Symbols

$\bar{\varepsilon}_x$	Average energy per tone
$\bar{C}$	Capacity of the discrete-time AWGN channel
$\mathcal{G}(f)$	Energy spectral density (ESD)
$\mathcal{M}$	Set of quantizer outputs and set of constellation symbols
$\mathcal{P}$	Power (Watts), calligraphic distinguishes from probabilities P
η	Efficiency
$\mathbb{E}[\mathsf{X}]$	Expected value of a random variable
F_s	Sampling frequency, $F_s = 1/T_s$
γ	Noise margin
γ_t	Target margin
$\hat{m}(t)$	Decoded symbol, receiver output
$\hat{P}(e^{j\Omega})$	Estimated periodogram
$\hat{R}_x[k]$	Autocorrelation for a finite-duration signal
$\hat{S}_x(e^{j\Omega})$	Periodogram
$\mathcal{P}$	Average power
Ω	Discrete-time angular frequency in radians
$\overline{E}_c$	Average energy of a constellation (Joules)
R_{sym}	$R_{\text{sym}} = 1/T_{\text{sym}}$, Which is called the *symbol rate*
R_{sym}	Signaling frequency or symbol rate $R_{\text{sym}} = 1/T_{\text{sym}}$
σ	Standard deviation
T_s	Sampling interval
T_{sym}	Symbol interval
b	Number of bits per sample
BW	Bandwidth
C	Correlation between two random variables X and Y
D	Signal space or constellation dimension (e. g., $D = 2$ for QAM)
$c(t)$	Carrier signal, typically a sinusoid $\cos(2\pi f_c t + \phi_c)$
E_p	Energy of the shaping pulse
f_0	Fundamental frequency (Hz)
f_c	Cutoff frequency
f_p	Bandpass frequency
f_r	Stopband frequency

$H(e^{j\Omega})$	Frequency response of a discrete-time system
$H(f)$	Frequency response of a continuous-time system
$H(s)$	Transfer function of a continuous-time system
$h(t)$ or $h[n]$	Impulse response of a system
$H(z)$	Transfer function of a discrete-time system
L	Oversampling factor, given by $L = T_{\text{sym}}/T_s$
L_{CP}	Prefix-cyclic length
M	Number of symbols in M-ary modulation
$m(t)$	Information source
m_i	Symbol of a constellation of M symbols
MS	Mean-square
N	FFT length, and other uses, e. g., number of samples per frame
$\mathcal{N}$	Gaussian (normal) distribution $\mathcal{N}(\mu, \sigma^2)$
ν	Noise in continuous $\nu(t)$ and discrete-time $\nu[n]$
$P(\omega)$	Spectrum of a shaping pulse in continuous-time
$p(t)$	Shaping pulse (such as the raised cosine)
Q	Quality factor or Q-factor
$r(t)$	Received signal
$R_X(s,t)$	Autocorrelation function
$S(f)$	Continuous-time power spectral density (PSD)
$S(e^{j\Omega})$	Discrete-time PSD
$s(t)$	Transmitter output, which is the channel input
$S_{ms}[k]$	Mean-square spectrum
$W(e^{j\Omega})$	Discrete-time Fourier transform of $w[n]$
$w[n]$	Rectangular window
$X(\tau, f)$	Short-time Fourier transform of $x(t)$
$X(e^{j\Omega})$	Discrete-time Fourier transform of $x[n]$
$X(f)$	Continuous-time Fourier transform of $x(t)$ with f in Hertz
$X(\omega)$	Continuous-time Fourier transform of $x(t)$ with ω in rad/s
$X(s)$	Laplace transform of $x(t)$
$x(t)$	Continuous-time signal

$X(z)$	Z transform of $x[n]$
$x[n]$	Discrete-time signal
$x_b[n]$	Binary representation of a signal
$x_i[n]$	Integer-valued signal
$x_q(t)$	Quantized continuous-time signal
$x_q[n]$	Digital signal
$x_s(t)$	Sampled signal
$X_N(e^{j\Omega})$	DTFT of windowed version of $x[n]$ with N samples
$X_T(f)$	Continuous-time Fourier transform of $x_T(t)$
$x_T(t)$	Truncated version of $x(t)$ with duration T

Bibliography

[AM07] J. Anderson and S. Mohan. *Source and Channel Coding: An Algorithmic Approach*. Springer, 2007.

[ANS03] ANSI. T1.417 - Spectrum Management fo Loop Transmission Systems, Issue 2, 2003.

[Ant07] Jérôme Antoni. Cyclic spectral analysis in practice. *Mechanical Systems and Signal Processing*, 21:597–630, 2007.

[ASS11] Tulay Adali, Peter J. Schreier, and Louis L. Scharf. Complex-valued signal processing: The proper way to deal with impropriety. *IEEE Transactions on Signal Processing*, 59:5101–25, Nov. 2011.

[BF01] J. Benedetto and P. Ferreira. *Modern Sampling Theory*. Birkhauser Boston, 2001.

[BLM04] John R. Barry, Edward A. Lee, and David G. Messerschmitt. *Digital Communication*. Kluwer, 3rd edition, 2004.

[BS92] P. Butzer and R. Stens. Sampling theory for not necessarily band-limited functions: A historical overview. *SIAM Review*, 34:40–53, 1992.

[CF97] John M. Cioffi and G. D. Forney Jr. Generalized decision feedback equalization for packet transmission with ISI and Gaussian noise. In *Communications, Computation, Control and Signal Processing: A Tribute to Thomas Kailath*, pages 79–127. Kluwer Academy, Boston, 1997.

[Cio10] J. Cioffi. Stanford class notes – `http://www.stanford.edu/group/cioffi/`, 2010.

[CLSG02] P. Ciblat, P. Loubaton, E. Serpedin, and G. B. Giannakis. Performance of blind carrier offset estimation for noncircular transmissions through frequency-selective channels. *IEEE Transactions on Signal Processing*, 50(1):130–140, Jan 2002.

[Cor68] C. Corduneanu. *Almost Periodic Functions*. Interscience Publishers (John Wiley & Sons), 1968.

[CV03] P. Ciblat and L. Vandendorpe. Blind carrier frequency offset estimation for noncircular constellation-based transmissions. *IEEE Transactions on Signal Processing*, 51(5):1378–89, May 2003.

[DdSN10] P. S. R. Diniz, E. A. B. da Silva, and S. L. Netto. *Digital Signal Processing: System Analysis and Design*. Cambridge University Press, 2nd ed. edition, 2010.

[Fra80] L. Franks. Carrier and bit synchronization in data communication: A tutorial review. *IEEE Transactions on Communications*, 28:1107–1121, Aug. 1980.

[Gar91] W. A. Gardner. Exploitation of spectral redundancy in cyclostationary signals. *IEEE Signal Processing Mag.*, 8:14–36, Apr. 1991.

[GG98] Fulvio Gini and Georgios B. Giannakis. Frequency offset and symbol timing recovery in flat-fading channels: A cyclostationary approach. *IEEE Transactions on Communications*, 46:400–411, March 1998.

[Gia99] G. Giannakis. *Digital Signal Processing Handbook*, chapter 17 - Cyclostationary Signal Analysis. Chapman & Hall, CRC Press, 1999.

[GLAK94] M. Genossar, H. Lev-Ari, and T. Kailath. Consistent estimation of the cyclic autocorrelation. *IEEE Transactions on Signal Processing*, 42(3):595–603, March 1994.

[GP06] George Ginis and Chia-Ning Peng. Alien crosstalk cancellation for multipair digital subscriber line systems. *Applied Signal Processing, EURASIP Journal on*, 2006:1–12, 2006.

[Hay01] Simon O. Haykin. *Adaptive Filter Theory*. Prentice Hall Information and System Sciences Series, 4 th edition, 2001.

[JN84] N. Jayant and P. Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video*. Prentice Hall, 1984.

[Kay06] S. Kay. *Intuitive Probability and Random Processes Using Matlab*. Springer, 2006.

[Lat78] B. Lathi. *Signal Processing and Linear Systems*. Berkeley Cambridge Press, 1978.

[Luk99] H. Luke. The origins of the sampling theory. *IEEE Communications Magazine*, 37:106–108, 1999.

[Lyo10] Richard G. Lyons. *Understanding Digital Signal Processing*. Prentice Hall, 3rd edition, 2010.

[Mal92] H. Malvar. *Signal Processing with Lapped Transforms*. Artech House, 1992.

[Mar01] F. Marvasti. *Nonuniform Sampling: Theory and Practice*. Springer, 2001.

[MdJ94] M. Moeneclaey and G. de Jonghe. Ml-oriented nda carrier synchronization for general rotationally symmetric signal constellations. *IEEE Transactions on Communications*, 42(8):2531–3, Aug. 1994.

[Mit10] Sanjit K. Mitra. *Digital Signal Processing, A Computer-Based Approach*. McGraw-Hill, 4th edition, 2010.

[NM93] Fredy D. Neeser and James L. Massey. Proper complex random processes with applications to information theory. *IEEE Transactions on Information Theory*, 39(4):1293–1302, July 1993.

[Opp75] A. Oppenheim. *Digital Signal Processing*. Prentice-Hall, 1975.

[OS09] A. Oppenheim and R. Schafer. *Discrete-time signal processing*. Prentice-Hall, 3rd edition, 2009.

[OWH96] A. Oppenheim, A. Willsky, and S. Hamid. *Signals and Systems*. Prentice Hall, 2nd edition, 1996.

[Pee86] P. Peebles. *Digital Communication Systems*. Prentice Hall, 1986.

[PS07] John Proakis and Masoud Salehi. *Digital Communications*. McGraw-Hill, 5th edition, 2007.

[RNT84] J. Rodgers, W. Nicewander, and L. Toothaker. Linearly independent, orthogonal, and uncorrelated variables. *The American Statistician*, 38(2):133–134, May 1984.

[RY90] K. Rao and P. Yip. *Discrete Cosine Transform—Algorithms, Advantages, Applications*. Academic Press, 1990.

[SS10] P. J. Schreier and L. L. Scharf. *Statistical Signal Processing of Complex-Valued Data: The Theory of Improper and Noncircular Signals*. Cambridge Univ. Press, 2010.

[TMK00] T. Thaiupathump, C. Murphy, and S. Kassam. Asymmetric signaling constellations for phase estimation. In *Proceedings of the Tenth IEEE Workshop on Statistical Signal and Array Processing*, pages 161 – 165, 2000.

[url] url. All references starting with prefix *url* are listed at `http://www.aldebaro.ufpa.br`.

Index

ADC chip, 3
Almost periodic, 35
Almost-periodic signals, 34
Analog signal, 2
Autocorrelation function, 187
AWGN, 49

Bias of an estimator, 212
Block, in GR, 231
Block-based processing, 84

Circular random variable, 198
Circularly symmetric, 198
Coefficients, transform, 85
Complex-valued signals, 7
Correlation, 41
Correlation coefficient, 41
Covariance, 41, 190
Cramér-Rao bound, 212
Cross-correlation, 47
Cyclic frequency, 199

D transform, 117
dBc, 220
Decorrelation, 214
Digital signal, 2
Dot product, 82
DSP, 73
Dynamic range, 17

Ensemble average, 43
Ergodic, 193
Ergodicity, 43
Estimation theory, 212
Estimator, 212
Even and odd parts of a signal, 157

FFT algorithms, 91
Fixed-point representation, 23
Floating-point representation, 26
Folding frequency, 107
Frame-based processing, 84
Fundamental period, 33

Gibbs phenomenon, 97
Granular region of a quantizer, 19

Harmonics, 100
Hermitian symmetry, 44, 102

i. i. d. - Independent and identically distributed, 51
i. i. d. random process, 191
Impulses, 5
Initial value theorem, 121
Inner product, 82
Insertion frequency response, 222
Insertion loss, 221
Inverse transform, 85

Leakage, spectrum, 137
Likelihood function, 33
Linear predictive coding, 215

Matched Z-transform, 118
MMSE estimator, 213
Moore-Penrose pseudoinverse, 172

Nyquist frequency, 16, 107

Offset binary output coding, 22
Outlier, 18
Overload region of a quantizer, 19

Partial fraction decomposition, 161
PCM, 53
Probability density function, 182
Probability mass function, 182
Proper random variable, 196

Pseudoinverse, 172
PSNR, 145

Quantization, 17
Quantization noise, 51
Quantization SNR, 52
Quasi-periodic, 35

Ramp signal, 116
Real-valued signals, 7
Rectangle method, 165
Region of convergence (ROC), Laplace transform, 112
RMS value, 38

Sampled signal, 8
Samples per second (SPS), 74
Sampling, 9
Sampling theorem, 10
Saturation region of a quantizer, 19
Scheduler, in GR, 231
Signal-to-noise ratio (SNR), 50
Spectrum, 102
Standard Gaussian, 31
Stationarity, 191
Superposition, 227
Support (signal), 6
Support of a PDF, 33

Twiddle factor, 90

Unbalanced constellation, 198
Unbiased estimator, 212
Uncorrelated random variables, 42
Unilateral Laplace transform, 117
Unilateral spectrum, 133
Unitary DFT, 89
Unitary matrix, 85

Vector quantization, 128

Wavelets, 92
Where-to-shift rule, 4
White Gaussian noise (WGN), 46
White signal, 46

Zero-padding, 44, 107, 127
Zeros and poles, 113