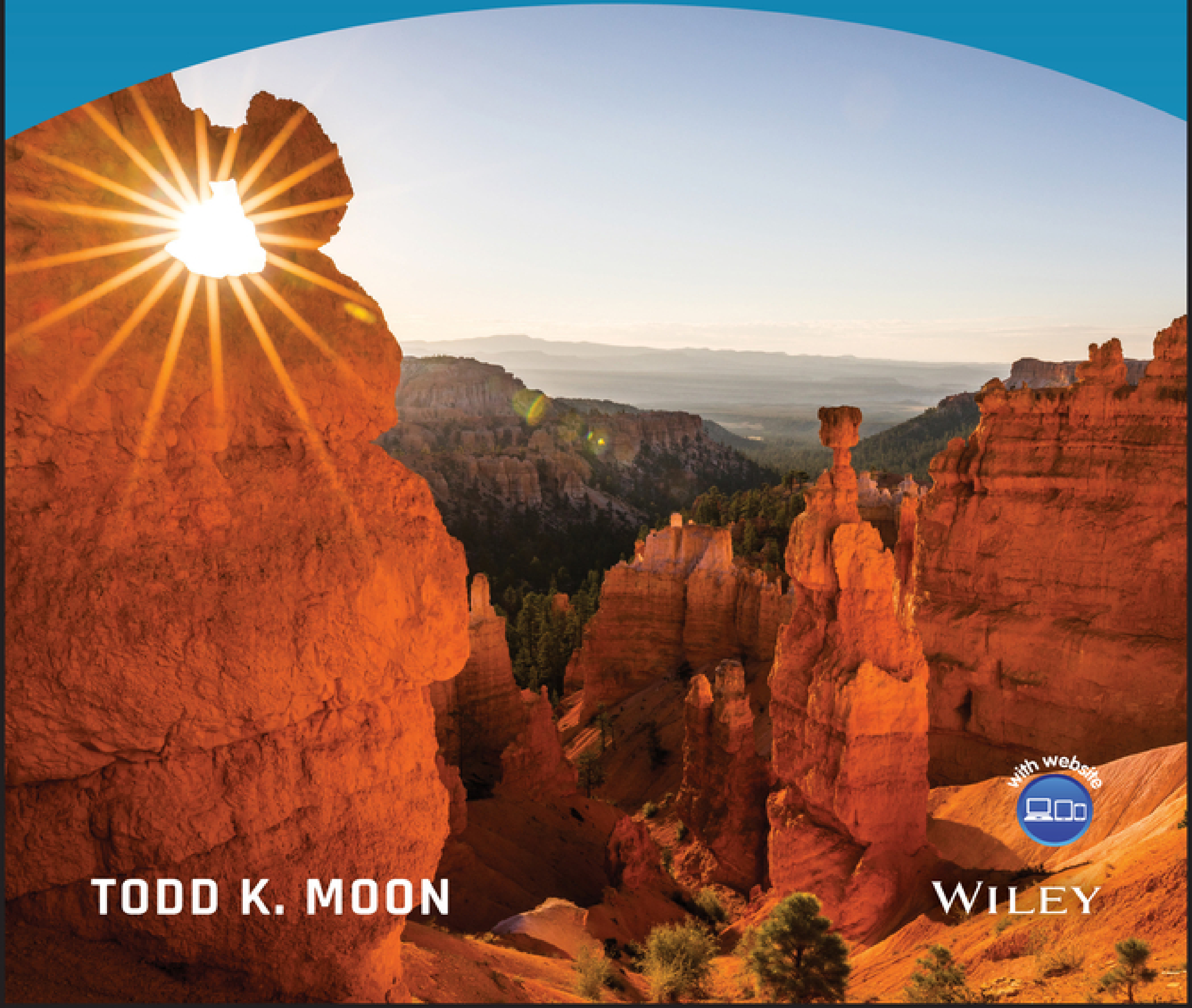
ERROR CORRECTION CODING

MATHEMATICAL METHODS AND ALGORITHMS

FIRST EDITION



TODD K. MOON

WILEY

Error Correction Coding

Error Correction Coding

Mathematical Methods and Algorithms

Second Edition

Todd K. Moon
Utah State University, Utah, USA

WILEY

This second edition first published 2021
© 2021 by John Wiley and Sons, Inc.

Edition History
John Wiley and Sons, Inc. (1e, 2005)

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by law. Advice on how to obtain permission to reuse material from this title is available at http://www.wiley.com/go/permissions.

The right of Todd K. Moon to be identified as the author of this work has been asserted in accordance with law.

Registered Office
John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, USA

Editorial Office
111 River Street, Hoboken, NJ 07030, USA

For details of our global editorial offices, customer services, and more information about Wiley products visit us at www.wiley.com.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Some content that appears in standard print versions of this book may not be available in other formats.

Limit of Liability/Disclaimer of Warranty
MATLAB® is a trademark of The MathWorks, Inc. and is used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This work's use or discussion of MATLAB® software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB® software.

While the publisher and authors have used their best efforts in preparing this work, they make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives, written sales materials or promotional statements for this work. The fact that an organization, website, or product is referred to in this work as a citation and/or potential source of further information does not mean that the publisher and authors endorse the information or services the organization, website, or product may provide or recommendations it may make. This work is sold with the understanding that the publisher is not engaged in rendering professional services. The advice and strategies contained herein may not be suitable for your situation. You should consult with a specialist where appropriate. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Neither the publisher nor authors shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Library of Congress Cataloging-in-Publication Data

Names: Moon, Todd K., author.
Title: Error correction coding : mathematical methods and algorithms / Todd K. Moon.
Description: Second edition. | Hoboken, NJ : Wiley, 2021. | Includes bibliographical references and index.
Identifiers: LCCN 2020027697 (print) | LCCN 2020027698 (ebook) | ISBN 9781119567479 (cloth) | ISBN 9781119567486 (adobe pdf) | ISBN 9781119567493 (epub)
Subjects: LCSH: Engineering mathematics. | Error-correcting codes (Information theory)
Classification: LCC TA331 .M66 2021 (print) | LCC TA331 (ebook) | DDC 621.3820285/572–dc23
LC record available at https://lccn.loc.gov/2020027697
LC ebook record available at https://lccn.loc.gov/2020027698

Cover Design: Wiley
Cover Image: © Putt Sakdhnagool/Getty Images

Set in 10/12pt NimbusRomNo9L by SPi Global, Chennai, India

10 9 8 7 6 5 4 3 2 1

Contents

Preface xvii

List of Program Files xxiii

List of Laboratory Exercises xxix

List of Algorithms xxxi

List of Figures xxxiii

List of Tables xli

List of Boxes xliii

About the Companion Website xlv

Part I Introduction and Foundations 1

1 A Context for Error Correction Coding 3

1.1 Purpose of This Book 3
1.2 Introduction: Where Are Codes? 3
1.3 The Communications System 4
1.4 Basic Digital Communications 10
1.4.1 Binary Phase-Shift Keying 10
1.4.2 More General Digital Modulation 11
1.5 Signal Detection 14
1.5.1 The Gaussian Channel 14
1.5.2 MAP and ML Detection 16
1.5.3 Special Case: Binary Detection 18
1.5.4 Probability of Error for Binary Detection 19
1.5.5 Bounds on Performance: The Union Bound 21
1.5.6 The Binary Symmetric Channel 23
1.5.7 The BSC and the Gaussian Channel Model 24
1.6 Memoryless Channels 24
1.7 Simulation and Energy Considerations for Coded Signals 25
1.8 Some Important Definitions and a Trivial Code: Repetition Coding 27
1.8.1 Detection of Repetition Codes Over a BSC 27
1.8.2 Soft-Decision Decoding of Repetition Codes Over the AWGN 31
1.8.3 Simulation of Results 32
1.8.4 Summary 32
1.9 Hamming Codes 32
1.9.1 Hard-Input Decoding Hamming Codes 34
1.9.2 Other Representations of the Hamming Code 36
1.10 The Basic Questions 38
1.11 Historical Milestones of Coding Theory 39
1.12 A Bit of Information Theory 39
1.12.1 Information- Theoretic Definitions for Discrete Random Variables 39
1.12.2 Data Processing Inequality 43
1.12.3 Channels 44
1.12.4 Channel Capacity 46
1.12.5 Information Theoretic Definitions for Continuous Random Variables 47
1.12.6 The Channel Coding Theorem 48

1.12.7 "Proof" of the Channel Coding Theorem 49
1.12.8 Capacity for the Continuous-Time AWGN Channel 53
1.12.9 Transmission at Capacity with Errors 54
1.12.10 The Implication of the Channel Coding Theorem 55
1.12.11 Non-Asymptotic Information Theory 56
Lab 1 Simulating a Communications Channel 60
Objective 60
Background 60
Use of Coding in Conjunction with the BSC 61
Assignment 61
Programming Part 61
Resources and Implementation Suggestions 62
1.13 Exercises 64
1.14 References 68

Part II Block Codes **69**

2 Groups and Vector Spaces **71**
2.1 Introduction 71
2.2 Groups 71
2.2.1 Subgroups 74
2.2.2 Cyclic Groups and the Order of an Element 75
2.2.3 Cosets 76
2.2.4 Lagrange's Theorem 77
2.2.5 Induced Operations; Isomorphism 78
2.2.6 Homomorphism 81
2.3 Fields: A Prelude 82
2.4 Review of Linear Algebra 84
2.5 Exercises 89
2.6 References 91

3 Linear Block Codes **93**
3.1 Basic Definitions 93
3.2 The Generator Matrix Description of Linear Block Codes 94
3.2.1 Rudimentary Implementation 96
3.3 The Parity Check Matrix and Dual Codes 96
3.3.1 Some Simple Bounds on Block Codes 98
3.4 Error Detection and Correction Over Hard-Output Channels 99
3.4.1 Error Detection 100
3.4.2 Error Correction: The Standard Array 100
3.5 Weight Distributions of Codes and Their Duals 104
3.6 Binary Hamming Codes and Their Duals 106
3.7 Performance of Linear Codes 108
3.7.1 Error Detection Performance 108
3.7.2 Error Correction Performance 109
3.7.3 Performance for Soft-Decision Decoding 112
3.8 Erasure Decoding 113
3.8.1 Binary Erasure Decoding 114
3.9 Modifications to Linear Codes 114
3.10 Best-Known Linear Block Codes 116
3.11 Exercises 116
3.12 References 121

4 Cyclic Codes, Rings, and Polynomials **123**
4.1 Introduction 123
4.2 Basic Definitions 123
4.3 Rings 124
4.3.1 Rings of Polynomials 125
4.4 Quotient Rings 126
4.5 Ideals in Rings 128
4.6 Algebraic Description of Cyclic Codes 130
4.7 Nonsystematic Encoding and Parity Check 131
4.8 Systematic Encoding 134
4.9 Some Hardware Background 136
4.9.1 Computational Building Blocks 136
4.9.2 Sequences and Power Series 137
4.9.3 Polynomial Multiplication 137
4.9.4 Polynomial Division 138
4.9.5 Simultaneous Polynomial Division and Multiplication 141
4.10 Cyclic Encoding 143
4.11 Syndrome Decoding 147
4.12 Shortened Cyclic Codes 153
4.12.1 Method 1: Simulating the Extra Clock Shifts 154
4.12.2 Method 2: Changing the Error Pattern Detection Circuit 155
4.13 Binary CRC Codes 155
4.13.1 Byte-Oriented Encoding and Decoding Algorithms 157
4.13.2 CRC Protecting Data Files or Data Packets 161
Appendix 4.A Linear Feedback Shift Registers 161
Appendix 4.A.1 Basic Concepts 162
Appendix 4.A.2 Connection With Polynomial Division 165
Appendix 4.A.3 Some Algebraic Properties of Shift Sequences 167
Lab 2 Polynomial Division and Linear Feedback Shift Registers 168
Objective 168
Preliminary Exercises 168
Programming Part: `BinLFSR` 168
Resources and Implementation Suggestions 169
Programming Part: `BinPolyDiv` 169
Follow-On Ideas and Problems 170
Lab 3 CRC Encoding and Decoding 170
Objective 170
Preliminary 170
Programming Part 170
Resources and Implementation Suggestions 171
4.14 Exercise 172
4.15 References 178

5 Rudiments of Number Theory and Algebra **179**
5.1 Motivation 179
5.2 Number Theoretic Preliminaries 182
5.2.1 Divisibility 182
5.2.2 The Euclidean Algorithm and Euclidean Domains 184
5.2.3 An Application of the Euclidean Algorithm: The Sugiyama Algorithm 189
5.2.4 Congruence 192
5.2.5 The ϕ Function 193
5.2.6 Some Cryptographic Payoff 193

5.3 The Chinese Remainder Theorem 196
5.3.1 The CRT and Interpolation 198
5.4 Fields 200
5.4.1 An Examination of $\mathbb{R}$ and $\mathbb{C}$ 201
5.4.2 Galois Field Construction: An Example 204
5.4.3 Connection with Linear Feedback Shift Registers 207
5.5 Galois Fields: Mathematical Facts 207
5.6 Implementing Galois Field Arithmetic 211
5.6.1 Zech Logarithms 211
5.6.2 Hardware Implementations 212
5.7 Subfields of Galois Fields 213
5.8 Irreducible and Primitive Polynomials 214
5.9 Conjugate Elements and Minimal Polynomials 216
5.9.1 Minimal Polynomials 218
5.10 Factoring $x^n - 1$ 222
5.11 Cyclotomic Cosets 223
Appendix 5.A How Many Irreducible Polynomials Are There? 224
Appendix 5.A.1 Solving for I_m Explicitly: The Moebius Function 227
Lab 4 Programming the Euclidean Algorithm 228
Objective 228
Preliminary Exercises 228
Background 228
Programming Part 229
Lab 5 Programming Galois Field Arithmetic 229
Objective 229
Preliminary Exercises 229
Programming Part 230
5.12 Exercise 231
5.13 References 239

6 BCH and Reed–Solomon Codes: Designer Cyclic Codes **241**
6.1 BCH Codes 241
6.1.1 Designing BCH Codes 241
6.1.2 The BCH Bound 244
6.1.3 Weight Distributions for Some Binary BCH Codes 246
6.1.4 Asymptotic Results for BCH Codes 248
6.2 Reed–Solomon Codes 249
6.2.1 Reed–Solomon Construction 1 249
6.2.2 Reed–Solomon Construction 2 250
6.2.3 Encoding Reed–Solomon Codes 251
6.2.4 MDS Codes and Weight Distributions for RS Codes 252
6.3 Decoding BCH and RS Codes: The General Outline 253
6.3.1 Computation of the Syndrome 254
6.3.2 The Error Locator Polynomial 255
6.3.3 Chien Search 255
6.4 Finding the Error Locator Polynomial 256
6.4.1 Simplifications for Narrow-Sense Binary Codes; Peterson's Algorithm 258
6.4.2 Berlekamp–Massey Algorithm 261
6.4.3 Characterization of LFSR Length in Massey's Algorithm 262
6.4.4 Simplifications for Binary Codes 266
6.5 Nonbinary BCH and RS Decoding 268
6.5.1 Forney's Algorithm 268

6.6 Euclidean Algorithm for the Error Locator Polynomial 272
6.7 Erasure Decoding for Nonbinary BCH or RS Codes 274
6.8 Galois Field Fourier Transform Methods 277
6.8.1 Equivalence of the Two Reed–Solomon Code Constructions 281
6.8.2 Frequency-Domain Decoding 282
6.9 Variations and Extensions of Reed–Solomon Codes 283
6.9.1 Simple Modifications 283
6.9.2 Generalized Reed–Solomon Codes and Alternant Codes 283
6.9.3 Goppa Codes 285
6.9.4 Decoding Alternant Codes 286
6.9.5 Cryptographic Connections: The McEliece Public Key Cryptosystem 286
Lab 6 Programming the Berlekamp–Massey Algorithm 287
Background 287
Assignment 287
Preliminary Exercises 287
Programming Part 288
Resources and Implementation Suggestions 288
Lab 7 Programming the BCH Decoder 289
Objective 289
Preliminary Exercises 289
Programming Part 289
Resources and Implementation Suggestions 290
Follow-On Ideas and Problems 290
Lab 8 Reed–Solomon Encoding and Decoding 290
Objective 290
Background 290
Programming Part 290
Appendix 6.A Proof of Newton's Identities 291
6.11 Exercise 293
6.12 References 297

7 Alternate Decoding Algorithms for Reed–Solomon Codes 299
7.1 Introduction: Workload for Reed–Solomon Decoding 299
7.2 Derivations of Welch–Berlekamp Key Equation 299
7.2.1 The Welch–Berlekamp Derivation of the WB Key Equation 300
7.2.2 Derivation from the Conventional Key Equation 304
7.3 Finding the Error Values 306
7.4 Methods of Solving the WB Key Equation 307
7.4.1 Background: Modules 308
7.4.2 The Welch–Berlekamp Algorithm 309
7.4.3 A Modular Approach to the Solution of the WB Key Equation 315
7.5 Erasure Decoding with the WB Key Equation 326
7.6 The Guruswami–Sudan Decoding Algorithm and Soft RS Decoding 326
7.6.1 Bounded Distance, ML, and List Decoding 326
7.6.2 Error Correction by Interpolation 327
7.6.3 Polynomials in Two Variables 328
7.6.4 The GS Decoder: The Main Theorems 333
7.6.5 Algorithms for Computing the Interpolation Step 340
7.6.6 A Special Case: $m = 1$ and $L = 1$ 351
7.6.7 An Algorithm for the Factorization Step: The Roth–Ruckenstein Algorithm. 353
7.6.8 Soft-Decision Decoding of Reed–Solomon Codes 358
7.7 Exercises 367
7.8 References 370

8 Other Important Block Codes 371
8.1 Introduction 371
8.2 Hadamard Matrices, Codes, and Transforms 371
8.2.1 Introduction to Hadamard Matrices 371
8.2.2 The Paley Construction of Hadamard Matrices 373
8.2.3 Hadamard Codes 376
8.3 Reed–Muller Codes 376
8.3.1 Boolean Functions 377
8.3.2 Definition of the Reed–Muller Codes 378
8.3.3 Encoding and Decoding Algorithms for First-Order RM Codes 380
8.3.4 The Reed Decoding Algorithm for $RM(r,m)$ Codes, $r \geq 1$ 385
8.3.5 Other Constructions of Reed–Muller Codes 392
8.4 Building Long Codes from Short Codes: The Squaring Construction 393
8.5 Quadratic Residue Codes 397
8.6 Golay Codes 399
8.6.1 Decoding the Golay Code 400
8.7 Exercises 403
8.8 References 405

9 Bounds on Codes 407
9.1 The Gilbert–Varshamov Bound 410
9.2 The Plotkin Bound 411
9.3 The Griesmer Bound 412
9.4 The Linear Programming and Related Bounds 413
9.4.1 Krawtchouk Polynomials 415
9.4.2 Character 415
9.4.3 Krawtchouk Polynomials and Characters 417
9.5 The McEliece–Rodemich–Rumsey–Welch Bound 419
9.6 Exercises 420
9.7 References 424

10 Bursty Channels, Interleavers, and Concatenation 425
10.1 Introduction to Bursty Channels 425
10.2 Interleavers 425
10.3 An Application of Interleaved RS Codes: Compact Discs 428
10.4 Product Codes 429
10.5 Reed–Solomon Codes 431
10.6 Concatenated Codes 431
10.7 Fire Codes 432
10.7.1 Fire Code Definition 432
10.7.2 Decoding Fire Codes: Error Trapping Decoding 434
10.8 Exercises 436
10.9 References 437

11 Soft-Decision Decoding Algorithms 439
11.1 Introduction and General Notation 439
11.2 Generalized Minimum Distance Decoding 441
11.2.1 Distance Measures and Properties 441
11.3 The Chase Decoding Algorithms 444
11.4 Halting the Search: An Optimality Condition 445
11.5 Ordered Statistic Decoding 447
11.6 Soft Decoding Using the Dual Code: The Hartmann Rudolph Algorithm 449

11.7 Exercises 451
11.8 References 452

Part III Codes on Graphs 453

12 Convolutional Codes 455
12.1 Introduction and Basic Notation 455
12.1.1 The State 459
12.2 Definition of Codes and Equivalent Codes 460
12.2.1 Catastrophic Encoders 464
12.2.2 Polynomial and Rational Encoders 466
12.2.3 Constraint Length and Minimal Encoders 467
12.2.4 Systematic Encoders 470
12.3 Decoding Convolutional Codes 471
12.3.1 Introduction and Notation 471
12.3.2 The Viterbi Algorithm 474
12.3.3 Some Implementation Issues 483
12.4 Some Performance Results 488
12.5 Error Analysis for Convolutional Codes 491
12.5.1 Enumerating Paths Through the Trellis 494
12.5.2 Characterizing Node Error Probability P_e and Bit Error Rate P_b 499
12.5.3 A Bound on P_d for Discrete Channels 501
12.5.4 A Bound on P_d for BPSK Signaling Over the AWGN Channel 503
12.5.5 Asymptotic Coding Gain 505
12.6 Tables of Good Codes 505
12.7 Puncturing 506
12.7.1 Puncturing to Achieve Variable Rate 508
12.8 Suboptimal Decoding Algorithms for Convolutional Codes 509
12.8.1 Tree Representations 510
12.8.2 The Fano Metric 510
12.8.3 The Stack Algorithm 514
12.8.4 The Fano Algorithm 516
12.8.5 Other Issues for Sequential Decoding 523
12.8.6 A Variation on the Viterbi Algorithm: The M Algorithm 524
12.9 Convolutional Codes as Block Codes and Tailbiting Codes 524
12.10 A Modified Expression for the Path Metric 525
12.11 Soft Output Viterbi Algorithm (SOVA) 528
12.12 Trellis Representations of Block and Cyclic Codes 533
12.12.1 Block Codes 533
12.12.2 Cyclic Codes 533
12.12.3 Trellis Decoding of Block Codes 534
Lab 9 Programming Convolutional Encoders 535
Objective 535
Background 535
Programming Part 536
Lab 10 Convolutional Decoders: The Viterbi Algorithm 537
Objective 537
Background 537
Programming Part 538

12.13 Exercises 538
12.14 References 543

13 Trellis-Coded Modulation 545
13.1 Adding Redundancy by Adding Signals 545
13.2 Background on Signal Constellations 545
13.3 TCM Example 547
13.3.1 The General Ungerboeck Coding Framework 553
13.3.2 The Set Partitioning Idea 553
13.4 Some Error Analysis for TCM Codes 555
13.4.1 General Considerations 555
13.4.2 A Description of the Error Events 556
13.4.3 Known Good TCM Codes 560
13.5 Decoding TCM Codes 562
13.6 Rotational Invariance 562
13.6.1 Differential Encoding 566
13.6.2 Constellation Labels and Partitions 567
13.7 Multidimensional TCM 568
13.7.1 Some Advantages of Multidimensional TCM 569
13.7.2 Lattices and Sublattices 570
13.8 Multidimensional TCM Example: The V.34 Modem Standard 578
13.9 Exercises 585
Lab 11 Trellis-Coded Modulation Encoding and Decoding 586
Objective 586
Background 586
Programming Part 586
13.10 References 586

Part IV Iteratively Decoded Codes 589

14 Turbo Codes 591
14.1 Introduction 591
14.2 Encoding Parallel Concatenated Codes 593
14.3 Turbo Decoding Algorithms 595
14.3.1 The MAP Decoding Algorithm 596
14.3.2 Notation 596
14.3.3 Posterior Probability 598
14.3.4 Computing α_t and β_t 600
14.3.5 Computing γ_t 601
14.3.6 Normalization 602
14.3.7 Summary of the BCJR Algorithm 603
14.3.8 A Matrix/Vector Formulation 605
14.3.9 Comparison of the Viterbi Algorithm and the BCJR Algorithm 605
14.3.10 The BCJR Algorithm for Systematic Codes 606
14.3.11 Turbo Decoding Using the BCJR Algorithm 607
14.3.12 Likelihood Ratio Decoding 609
14.3.13 Statement of the Turbo Decoding Algorithm 612
14.3.14 Turbo Decoding Stopping Criteria 612
14.3.15 Modifications of the MAP Algorithm 614
14.3.16 Corrections to the Max-Log-MAP Algorithm 616
14.3.17 The Soft-Output Viterbi Algorithm 616

14.4 On the Error Floor and Weight Distributions 616
14.4.1 The Error Floor 616
14.4.2 Spectral Thinning and Random Permuters 618
14.4.3 On Permuters 621
14.5 EXIT Chart Analysis 622
14.5.1 The EXIT Chart 624
14.6 Block Turbo Coding 626
14.7 Turbo Equalization 628
14.7.1 Introduction to Turbo Equalization 628
14.7.2 The Framework for Turbo Equalization 629
Lab 12 Turbo Code Decoding 631
Objective 631
Background 631
Programming Part 631
14.8 Exercise 631
14.9 References 634

15 Low-Density Parity-Check Codes: Introduction, Decoding, and Analysis 637
15.1 Introduction 637
15.2 LDPC Codes: Construction and Notation 637
15.3 Tanner Graphs 641
15.4 Decoding LDPC Codes 642
15.4.1 Decoding Using Log-Likelihood Ratios 642
15.4.2 Decoding Using Probabilities 652
15.4.3 Variations on Decoding Algorithms: The Min-Sum Decoder 659
15.4.4 Variations on Decoding Algorithms: Min-Sum with Correction 662
15.4.5 Hard-Decision Decoding 669
15.4.6 Divide and Concur Decoding 674
15.4.7 Difference Map Belief Propagation Decoding 681
15.4.8 Linear Programming Decoding 682
15.4.9 Decoding on the Binary Erasure Channel 691
15.4.10 BEC Channels and Stopping Sets 692
15.5 Why Low-Density Parity-Check Codes? 694
15.6 The Iterative Decoder on General Block Codes 695
15.7 Density Evolution 696
15.8 EXIT Charts for LDPC Codes 699
15.9 Irregular LDPC Codes 702
15.9.1 Degree Distribution Pairs 703
15.9.2 Density Evolution for Irregular Codes 704
15.9.3 Computation and Optimization of Density Evolution 706
15.9.4 Using Irregular Codes 708
15.10 More on LDPC Code Construction 708
15.11 Encoding LDPC Codes 708
15.12 A Variation: Low-Density Generator Matrix Codes 710
Lab 13 Programming an LDPC Decoder 710
Objective 710
Background 710
Assignment 711
Numerical Considerations 711
15.13 Exercise 712
15.14 References 715

16 Low-Density Parity-Check Codes: Designs and Variations **717**
16.1 Introduction 717
16.2 Repeat-Accumulate Codes 717
16.2.1 Decoding RA Codes 720
16.2.2 Irregular RA Codes 721
16.2.3 RA Codes with Multiple Accumulators 722
16.3 LDPC Convolutional Codes 723
16.4 Quasi-Cyclic Codes 725
16.4.1 QC Generator Matrices 725
16.4.2 Constructing QC Generator Matrices from QC Parity Check Matrices 727
16.5 Construction of LDPC Codes Based on Finite Fields 730
16.5.1 I. Construction of QC-LDPC Codes Based on the Minimum-Weight Codewords of a Reed–Solomon Code with Two Information Symbols 731
16.5.2 II. Construction of QC-LDPC Codes Based on a Special Subclass of RS Codes 733
16.5.3 III. Construction of QC-LDPC Codes Based on Subgroups of a Finite Field . 734
16.5.4 IV. Construction of QC-LDPC Codes Based on Subgroups of the Multiplicative Group of a Finite Field 735
16.5.5 Construction of QC-LDPC Codes Based on Primitive Elements of a Field . . 736
16.6 Code Design Based on Finite Geometries 736
16.6.1 Rudiments of Euclidean Geometry 736
16.6.2 A Family of Cyclic EG-LDPC Codes 740
16.6.3 Construction of LDPC Codes Based on Parallel Bundles of Lines 741
16.6.4 Constructions Based on Other Combinatoric Objects 742
16.7 Ensembles of LDPC Codes 742
16.7.1 Regular ensembles 743
16.7.2 Irregular Ensembles 743
16.7.3 Multi-edge-type Ensembles 745
16.8 Constructing LDPC Codes by Progressive Edge Growth (PEG) 746
16.9 Protograph and Multi-Edge-Type LDPC Codes 749
16.10 Error Floors and Trapping Sets 750
16.11 Importance Sampling 751
16.11.1 Importance Sampling: General Principles 752
16.11.2 Importance Sampling for Estimating Error Probability 754
Conventional Sampling (MC) 755
Importance Sampling (IS) 756
16.11.3 Importance Sampling for Tanner Trees 765
16.12 Fountain Codes 771
16.12.1 Conventional Erasure Correction Codes 772
16.12.2 Tornado Codes 772
16.12.3 Luby Transform Codes 774
16.12.4 Raptor Codes 774
16.13 References 774

Part V Polar Codes **777**

17 Polar Codes **779**
17.1 Introduction and Preview 779
17.2 Notation and Channels 781
17.3 Channel Polarization, $N = 2$ Channel 782
17.3.1 Encoding 782
17.3.2 Synthetic Channels and Mutual Information 783

17.3.3 Synthetic Channel Transition Probabilities 786
17.3.4 An Example with $N = 2$ Using the Binary Erasure Channel 786
17.3.5 Successive Cancellation Decoding 787
17.3.6 Tree Representation 792
17.3.7 The Polar Coding Idea 793
17.4 Channel Polarization, $N > 2$ Channels 793
17.4.1 Channel Combining: Extension from $N/2$ to N channels 793
17.4.2 Pseudocode for Encoder for Arbitrary N 797
17.4.3 Transition Probabilities and Channel Splitting 798
17.4.4 Channel Polarization Demonstrated: An Example Using the Binary Erasure Channel for $N > 2$ 802
17.4.5 Polar Coding 805
17.4.6 Tree Representation 807
17.4.7 Successive Cancellation Decoding 807
17.4.8 SC Decoding from a Message Passing Point of View on the Tree 810
17.4.9 A Decoding Example with $N = 4$ 811
17.4.10 A Decoding Example with $N = 8$ 813
17.4.11 Pseudo-Code Description of Successive Cancellation Algorithm 823
17.5 Some Theorems of Polar Coding Theory 826
17.5.1 $I(W)$ and $Z(W)$ for general B-DMCs 826
17.5.2 Channel Polarization 827
17.5.3 The Polarization Theorem 830
17.5.4 A Few Facts About Martingales 830
17.5.5 Proof of the Polarization Theorem 831
17.5.6 Another Polarization Theorem 832
17.5.7 Rate of Polarization 835
17.5.8 Probability of Error Performance 835
17.6 Designing Polar Codes 837
17.6.1 Code Design by Battacharyya Bound 837
17.6.2 Monte Carlo Estimation of $Z(W_N^{(i)})$ 838
17.7 Perspective: The Channel Coding Theorem 839
17.8 Systematic Encoding of Polar Codes 839
17.8.1 Polar Systematic Encoding via the Encoder Graph 840
17.8.2 Polar Systematic Encoding via Arıkan's Method 842
17.8.3 Systematic Encoding: The Bit Reverse Permutation 843
17.8.4 Decoding Systematically Encoded Codes 843
17.8.5 Flexible Systematic Encoding 843
17.8.6 Involutions and Domination Contiguity 846
17.8.7 Polar Codes and Domination Contiguity 847
17.8.8 Modifications for Polar Codes with Bit-Reverse Permutation 850
17.9 List Decoding of Polar Codes 850
17.9.1 The Likelihood Data Structure P 851
17.9.2 Normalization 853
17.9.3 Code to Compute P 854
17.9.4 The Bits Data Structure C 855
17.9.5 Code to Compute C 856
17.9.6 Supporting Data Structures 857
17.9.7 Code for Polar List Decoding 857
17.9.8 An Example of List Decoding 862
17.9.9 Computational Complexity 864
17.9.10 Modified Polar Codes 865
17.10 LLR-Based Successive Cancellation List Decoding 866
17.10.1 Implementation Considerations 867

17.11 Simplified Successive Cancellation Decoding ... 867
17.11.1 Modified SC Decoding ... 870
17.12 Relations with Reed–Muller Codes ... 871
17.13 Hardware and Throughput Performance Results ... 871
17.14 Generalizations, Extensions, and Variations ... 871
Appendix 17.A B_N is a Bit-Reverse Permutation ... 872
Appendix 17.B The Relationship of the Battacharyya Parameter to Channel Capacity ... 873
Appendix 17.B.1 Error Probability for Two Codewords ... 873
Appendix 17.B.2 Proof of Inequality (17.59) ... 874
Appendix 17.B.3 Proof of Inequality (17.60) [16] ... 879
Appendix 17.C Proof of Theorem 17.12 ... 880
17.15 Exercises ... 882
17.16 References ... 883

Part VI Applications 885

18 Some Applications of Error Correction in Modern Communication Systems 887
18.1 Introduction ... 887
18.2 Digital Video Broadcast T2 (DVB-T2) ... 887
18.2.1 BCH Outer Encoding ... 887
18.2.2 LDPC Inner Encoding ... 888
18.3 Digital Cable Television ... 892
18.4 E-UTRA and Long-Term Evolution ... 895
18.4.1 LTE Rate 1/3 Convolutional Encoder ... 897
18.4.2 LTE Turbo Code ... 897
18.5 References ... 898

Part VII Space-Time Coding 899

19 Fading Channels and Space-Time Codes 901
19.1 Introduction ... 901
19.2 Fading Channels ... 901
19.2.1 Rayleigh Fading ... 904
19.3 Diversity Transmission and Reception: The MIMO Channel ... 905
19.3.1 The Narrowband MIMO Channel ... 907
19.3.2 Diversity Performance with Maximal-Ratio Combining ... 908
19.4 Space-Time Block Codes ... 909
19.4.1 The Alamouti Code ... 909
19.4.2 A More General Formulation ... 911
19.4.3 Performance Calculation ... 911
19.4.4 Complex Orthogonal Designs ... 916
19.5 Space-Time Trellis Codes ... 917
19.5.1 Concatenation ... 919
19.6 How Many Antennas? ... 919
19.7 Estimating Channel Information ... 921
19.8 Exercises ... 922
19.9 References ... 923

References 925

Index 939

Preface

The goals of this second edition are much the same as the first edition: to provide a comprehensive introduction to error correction coding suitable for the engineering practitioner and to provide a solid foundation leading to more advanced treatments or research. Since the first edition, the content has been modernized to include substantially more on LDPC code design and decoder algorithms, as well as substantial coverage of polar codes. The thorough introduction to finite fields and algebraic codes of the first edition has been retained, with some additions in finite geometries used for LDPC code design. As observed in the first edition, the sophistication of the mathematical tools used increases over time. While keeping with the sense that this is the first time most readers will have seen these tools, a somewhat higher degree of sophistication is needed in some places.

The presentation is intended to provide a background useful both to engineers, who need to understand algorithmic aspects for the deployment and implementation of error correction coding, and to researchers, who need sufficient background to prepare them to read, understand, and ultimately contribute to the research literature. The practical algorithmic aspects are built upon a firm foundation of mathematics, which are carefully motivated and developed.

Pedagogical Features

Since its inception, coding theory has drawn from richly interacting variety of mathematical areas, including detection theory, information theory, linear algebra, finite geometries, combinatorics, optimization, system theory, probability, algebraic geometry, graph theory, statistical designs, Boolean functions, number theory, and modern algebra. The level of sophistication has increased over time: algebra has progressed from vector spaces to modules; practice has moved from polynomial interpolation to rational interpolation; Viterbi makes way for SOVA and BCJR. This richness can be bewildering to students, particularly engineering students who may be unaccustomed to posing problems and thinking abstractly. It is important, therefore, to motivate the mathematics carefully.

Some of the major pedagogical features of the book are as follows.

- While most engineering-oriented error-correction-coding textbooks clump the major mathematical concepts into a single chapter, in this book the concepts are developed over several chapters so they can be put to more immediate use. I have attempted to present the mathematics "just in time," when they are needed and well-motivated. Groups and linear algebra suffice to describe linear block codes. Cyclic codes motivate polynomial rings. The design of cyclic codes motivates finite fields and associated number-theoretical tools. By interspersing the mathematical concepts with applications, a deeper and broader understanding is possible.
- For most engineering students, finite fields, the Berlekamp–Massey algorithm, the Viterbi algorithm, BCJR, and other aspects of coding theory are abstract and subtle. Software implementations of the algorithms bring these abstractions closer to a meaningful reality, bringing deeper understanding than is possible by simply working homework problems and taking tests. Even when students grasp the concepts well enough to do homework on paper, these programs provide a further emphasis, as well as tools to *help* with the homework. The understanding becomes *experiential*, more than merely conceptual.

  Understanding of any subject typically improves when the student him- or herself has the chance to teach the material to someone (or something) else. A student must develop an especially clear understanding of a concept in order to "teach" it to something as dim-witted and literal-minded as a computer. In this process the computer can provide feedback to the student through debugging and program testing that reinforces understanding.

  In the coding courses I teach, students implement a variety of encoders and decoders, including Reed–Solomon encoders and decoders, convolutional encoders, turbo code decoders, and LDPC decoders. As a result of these programming activities, students move beyond an on-paper understanding, gaining a perspective of what coding theory can do and how to put it to work. A colleague of mine observed that many students emerge from a first course in coding theory more

LabIntro.pdf

confused than informed. My experience with these programming exercises is that my students are, if anything, overconfident, and feel ready to take on a variety of challenges.

In this book, programming exercises are presented in a series of 13 Laboratory Exercises. These are supported with code providing most of the software "infrastructure," allowing students to focus on the particular algorithm they are implementing.

These labs also help with the coverage of the course material. In my course I am able to offload classroom instruction of some topics for students to read, with the assurance that the students will learn it solidly on their own as they implement it. (The Euclidean algorithm is one of these topics in my course.)

Research in error control coding can benefit from having a flexible library of tools for the computations, particularly since analytical results are frequently not available and simulations are required. The laboratory assignments presented here can form the foundation for a research library, with the added benefit that having written major components, the researcher can easily modify and extend them.

It is in light of these pedagogic features that this book bears the subtitle *Mathematical Methods and Algorithms.*

There is sufficient material in this book for a one- or two-semester course based on the book, even for instructors who prefer to focus less on implementational aspects and the laboratories.

Over 200 programs, functions and data files are associated with the text. The programs are written in MATLAB,[1] C, or C++. Some of these include complete executables which provide "tables" of primitive polynomials (over any prime field), cyclotomic cosets and minimal polynomials, and BCH codes (not just narrow sense), avoiding the need to tabulate this material. Other functions include those used to make plots and compute results in the book. These provide example of how the theory is put into practice. Other functions include those used for the laboratory exercises. The files are highlighted in the book by the icon

as in the marginal note above. The files are available at https://github.com/tkmoon/eccbook. Other aspects of the book include the following:

- Many recent advances in coding have resulted from returning to the perspective of coding as a detection problem. Accordingly, the book starts off with a digital communication framework with a discussion of **detection theory**.
- Recent codes such as polar codes or LDPC codes are capable of achieving capacity, or nearly so. It is important, therefore, to understand what capacity is and what it means to transmit at capacity. Chapter 1 also summarizes **information theory**, to put coding into its historical and modern context. Added in this second edition is an introduction to nonasymptotic information theory, which will be increasingly important in developing communication systems. The information theory informs EXIT chart analysis of turbo and LDPC codes.

[1] MATLAB is a registered trademark of The Mathworks, Inc.

- Pedagogically, Hamming codes are used to set the stage for the book by using them to demonstrate block codes, cyclic codes, trellises and Tanner graphs.
- Homework exercises are drawn from a variety of sources and are at a variety of levels. Some are numerical, testing basic understanding of concepts. Others provide the opportunity to prove or extend results from the text. Others extend concepts or provide new results. Because of the programming laboratories, exercises requiring decoding by hand of given bit sequences are few, since I am of the opinion that is better to know how to tell the computer than to do it by hand. I have drawn these exercises from a variety of sources, including problems that I faced as a student and those which I have given to students on homework and exams over the years.
- Number theoretic concepts such as divisibility, congruence, and the Chinese remainder theorem are developed at a point in the development where students can appreciate it.
- Occasionally connections between the coding theoretic concepts and related topics are pointed out, such as **public key cryptography** and **shift register sequences**. These add spice and motivate students with the understanding that the tools they are learning have broad applicability.
- There has been considerable recent progress made in decoding Reed–Solomon codes by re-examining their original definition. Accordingly, Reed–Solomon codes are defined both in this primordial way (as the image of a polynomial function) and also using a generator polynomial having roots that are consecutive powers of a primitive element. This sets the stage for several decoding algorithms for Reed–Solomon codes, including frequency-domain algorithms, **Welch–Berlekamp algorithm** and the **soft-input Guruswami–Sudan algorithm**.
- **Turbo codes**, including EXIT chart analysis, are presented, with both BCJR and SOVA decoding algorithms. Both probabilistic and likelihood decoding viewpoints are presented.
- **LDPC codes** are presented with an emphasis on the decoding algorithm. Density evolution analysis is also presented.
- **Polar Codes** are a family of codes developed since the first edition. They offer low complexity encode and decode with soft inputs and without iterative decoding. They also provably achieve channel capacity (as the code length goes to infinity). This book offers a deep introduction to polar codes, including careful description of encoding and decoding algorithms, with operational C++ code.
- Several **Applications** involving state-of-the-art systems illustrate how the concepts can be applied.
- **Space-time codes**, used for multi-antenna systems in fading channels, are presented.

Courses of Study

A variety of courses of study are possible. In the one-semester course I teach, I move quickly through principal topics of block, trellis, and iteratively–decoded codes. Here is an outline of one possible one-semester course:

Chapter 1: Major topics only.

Chapter 2: All.

Chapter 3: Major topics.

Chapter 4: Most. Leave CRC codes and LFSR to labs.

Chapter 5: Most. Touch on RSA.

Chapter 6: Most. Light on GFFT and variations.

Chapter 12: Most. Skip puncturing and stack-oriented algorithms.

Chapter 13: Most. Skip the V.34 material.

Chapter 14: Basic definition and the BCJR algorithm.

Chapter 15: Basic definition of LDPC codes and the sum-product decoder.

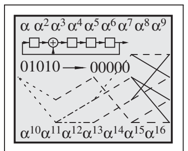
Chapter 17: Introduce the idea of

A guide in selecting material for this course is: follow the labs. To get through all 13 labs, selectivity is necessary.

An alternative two-semester course could be a semester devoted to block codes followed by a semester on trellis and iteratively decoded codes. A two semester sequence could move straight through the book, with possible supplements from the research literature on topics of particular interest to the instructor. A different two-semester approach would follow the outline above for one semester, followed by more advanced coverage of LDPC and polar codes.

Theorems, lemmas, corollaries, examples, and definitions are all numbered sequentially using the same counter in a chapter, which should make identifying these environments straightforward. Figures, tables, and equations each have their own counters. Definitions, examples, and proofs are terminated by the symbol □.

Use of Computers


comptut.pdf

The computer-based labs provide a means of working out some of the computational details that otherwise might require drudgery. These are primarily (with the exception of the first lab) to be done in C++, where the ability to overload operators and run at compiled speed make the language very well suited.

It may be helpful to have a "desk calculator" for homework and exploring ideas. Many tools exist now for this. The brief tutorial `comptut.pdf` provides an introduction to `gap` and `magma`, both of which can be helpful to students doing homework or research in this area. The `sage` package built on Python also provides considerable relevant capability.

Why This Book?

In my mind, the purposes of a textbook are these:

1. To provide a topographical map into the field of study, showing the peaks and the important roads to them. (However, in an area as rich as coding theory, it is impossible to be exhaustive.)
2. To provide specific details about important techniques.
3. To present challenging exercises that will strengthen students' understanding of the material and present new perspectives.
4. To have some reference value, so that practitioners can continue to use it.
5. To provide references to literature that will lead readers to make their own discoveries. With a rapidly-changing field, the references can only provide highlights; web-based searches have changed the nature of the game. Nevertheless, having a starting point in the literature is still important.
6. To present a breadth of ideas which will motivate students to innovate on their own.

A significant difficulty I have faced is selection. The terrain is so richly textured that it cannot be mapped out in a single book. Every communication or information theory conference and every issue of the *IEEE Transactions on Information Theory* yields new and significant results. Publishing restrictions and practicality limit this book from being encyclopedic. My role as author has been merely to select what parts of the map to include and to present them in a pedagogically useful way. In so doing, I have aimed to choose tools for the general practitioner and student (and of interest to me). Other than that selective role, no claim of creation is intended; I hope I have given credit as appropriate where it is due.

This book is a result of teaching a course in error correction coding at Utah State University for nearly three decades. Over that time, I have taught out of the books [45], [489], and [275], and my debt to these books is clear. Parts of some chapters grew out of lecture notes based on these books and the connections will be obvious. I have felt compelled to include many of the exercises from the first coding course I took out of [275]. These books have defined for me the *sine qua non* of error-correction coding texts. I am also indebted to [296] for its rich theoretical treatment, [401] for presentation of trellis coding material, [462] for discussion of bounds, [190] for exhaustive treatment of turbo coding methods, and to the many great researchers and outstanding expositors whose works have contributed to my understanding. More recently, the extensive coverage of LDPC codes in [391] has been extremely helpful, and motivated my own implementation of almost all of the decoding algorithms.

Acknowledgments

I am grateful for the supportive environment at Utah State University that has made it possible to undertake and to complete this task. Students in coding classes over the years have contributed to this material.

I have benefitted tremendously from feedback from careful readers and translators who have provided their own error correction from the first edition. Their negative acknowledgment protocols have improved the delivery of this packet of information. I thank you all!

To my six wonderful children — Leslie, Kyra, Kaylie, Jennie, Kiana, and Spencer — and presently twelve grandchildren, and my wife Barbara, who have seen me slip away too often and too long to write, I express my deep gratitude for their trust and patience. In the end, all I do is for them.

T.K.M
Logan, UT, October 2020

List of Program Files

`comptut.pdf`	Introduction to `gap` and `magma`	xx
`bpskprobplot.m`	Set up to plot prob. of error for BPSK	21
`bpskprob.m`	Plot probability of error for BPSK	21
`repcodeprob.m`	Error prob. for $(n, 1)$ repetition codes	31
`testrepcode.cc`	Test the repetition code performance	32
`repcodes.m`	Plot results for repetition code	32
`mindist.m`	Find minimum distance using exhaustive search	33
`hamcode74pe.m`	Probability of error for $(7, 4)$ Hamming code	35
`nchoosektest.m`	Compute $\binom{n}{k}$ and test for $k < 0$	35
`plotcapcmp.m`	Capacity of the AWGN and BAWGNC channels	49
`cawgnc2.m`	Compute capacity of AWGN channel	49
`cbawgnc2.m`	Compute the capacity of the BAWGN channel	49
`h2.m`	Compute the binary entropy function	49
`plotcbawn2.m`	Plot capacity for AWGN and BAWGN channels	54
`cbawgnc.m`	Compute capacity for BAWGN channel	54
`cawgnc.m`	Compute capacity for AWGN channel	54
`philog.m`	Log ϕ function for the BAWGNC	54
`phifun.m`	ϕ function for the BAWGNC	54
`gaussj2`	Gaussian elimination over $GF(2)$	95
`Hammsphere`	Number of points in a Hamming sphere	98
`genstdarray.c`	Generate a standard array for a code	101
`progdetH15.m`	Error detection prob. for $(15, 11)$ Hamming code	105
`progdet.m`	Error detection prob. for $(31, 21)$ Hamming code	105
`polyadd.m`	Add polynomials	126
`polysub.m`	Subtract polynomials	126
`polymult.m`	Multiply polynomials	126
`polydiv.m`	Polynomial quotient and remainder	126
`polyaddm.m`	Add polynomials modulo a number	126
`polysubm.m`	Subtract polynomials modulo a number	126
`polymultm.m`	Multiply polynomials modulo a number	126
`primitive.txt`	Table of primitive polynomials	163
`primfind.c`	Program to find primitive polynomials	163
`BinLFSR.h`	(lab, complete) Binary LFSR class	169
`BinLFSR.cc`	(lab, complete) Binary LFSR class	169
`testBinLFSR.cc`	(lab, complete) Binary LFSR class tester	169
`MakeLFSR`	(lab, complete) Makefile for tester	169
`BinPolyDiv.h`	(lab, complete) Binary polynomial division	170
`BinPolyDiv.cc`	(lab, incomplete) Binary polynomial division	170
`testBinPolyDiv.cc`	(lab, complete) Binary polynomial division test	170
`gcd.c`	A simple example of the Euclidean algorithm	177
`crtgamma.m`	Compute the gammas for the CRT	197
`fromcrt.m`	Convert from CRT representation to integer	197
`tocrt.m`	Compute the CRT representation of an integer	197
`testcrt.m`	An example of CRT calculations	197
`testcrp.m`	Test a polynomial CRT calculation	197
`tocrtpoly.m`	CRT representation of a polynomial	197
`fromcrtpolym.`	Polynomial CRT representation	197
`crtgammapoly.m`	Gammas for the CRT representation	197
`primfind`	Find primitive polynomials in $GF(p)[x]$	216
`cyclomin`	Cyclotomic cosets and minimal polynomials	223
`ModArnew.h`	Templatized Modulo arithmetic class	228
`testmodarnew1.cc`	Templatized Modulo arithmetic class tester	228

`polynomialT.h`	(lab, complete) Templatized polynomial class	229
`polynomialT.cc`	(lab, complete) Templatized polynomial class	229
`testpoly1.cc`	Demonstrate templatized polynomial class	229
`testgcdpoly.cc`	(lab, complete) Test polynomial GCD function	229
`gcdpoly.cc`	Polynomial GCD function (incomplete)	229
`GF2.h`	(lab, complete) GF(2) class	230
`GFNUM2m.h`	(lab, complete) Galois field $GF(2^m)$ class	230
`GFNUM2m.cc`	(lab, **incomplete**) Galois field $GF(2^m)$ class	230
`testgfnum.cc`	(lab, complete) Test Galois field class	230
`bchweight.m`	Weight dist. of BCH code from weight of dual	247
`bchdesigner`	Design a t-error correcting binary BCH code	248
`reedsolwt.m`	Weight distribution for an (n, k) RS code	253
`masseymodM.m`	Return the shortest LFSR for data sequence	265
`testRSerasure.cc`	RS with erasures using Euclidean alg.	275
`erase.mag`	Erasure decoding example in `magma`	276
`testBM.cc`	Test the Berlekamp–Massey algorithm	288
`Chiensearch.h`	(lab, complete) Chien Search class	289
`Chiensearch.cc`	(lab, **incomplete**) Chien Search class	289
`testChien.cc`	(lab, complete) Test the Chien Search class	289
`BCHdec.h`	(lab, complete) BCHdec decoder class	289
`BCHdec.cc`	(lab, **incomplete**) BCHdec decoder class	289
`testBCH.cc`	(lab, complete) BCHdec decoder class tester	289
`RSdec.h`	(lab, complete) RS decoder class header	290
`RSdec.cc`	(lab, **incomplete**) RS decoder class	291
`testRS.cc`	(lab, complete) Test RS decoder	291
`rsencode.cc`	(lab, complete) Encode a file using RS encoder	291
`rsdecode.cc`	(lab, complete) Decode a file using RS decoder	291
`bsc`	Simulate a binary symmetric channel	291
`testpxy.cc`	Concepts relating to 2-variable polynomials	328
`computekm.m`	Compute K_m for the Guruswami–Sudan decoder	336
`computeLm.cc`	Maximum list length for $GS(m)$ decoding	336
`computeLm.m`	Maximum list length for $GS(m)$ decoding	336
`testft.m`	Test the Feng–Tzeng algorithm	344
`fengtzeng.m`	Poly. s.t. the first $l + 1$ columns are lin. dep.	344
`invmodp.m`	Compute the inverse of a number modulo p	344
`testGS1.cc`	Test the GS decoder (Kotter part)	349
`kotter.cc`	Kotter interpolation algorithm	349
`testGS3.cc`	Test the GS decoder	350
`testGS5.cc`	Test the GS decoder	353
`kotter1.cc`	Kotter algorithm with $m = 1$	353
`testGS2.cc`	Test the Roth–Ruckenstein algorithm	353
`rothruck.cc`	Roth–Ruckenstein algorithm (find y-roots)	353
`rothruck.h`	Roth–Ruckenstein algorithm (find y-roots)	353
`Lbarex.m`	Average performance of a $GS(m)$ decoder	358
`computetm.m`	T_m error correction capability for GS decoder	358
`computeLbar.m`	Avg. no. of CWs around random pt.	358
`computeLm.m`	Compute max. length of list of decoder	358
`pi2ml`	Koetter–Vardy map reliability to multiplicity	363
`genrm.cc`	Create a Reed–Muller generator matrix	378

`rmdecex.m`	*RM*$(1,3)$ decoding example	383
`hadex.m`	Computation of H_8	384
`testfht.cc`	Test the fast Hadamard transform	384
`fht.cc`	Fast Hadamard transform	384
`fht.m`	Fast Hadamard transform	384
`rmdecex2.m`	*RM*$(2,4)$ decoding example	388
`testQR.cc`	Example of arithmetic for QR code decoding	398
`golaysimp.m`	Derive equations for algebraic Golay decoder	401
`testGolay.cc`	Test the algebraic Golay decoder	402
`golayrith.m`	Arithmetic Golay decoder	403
`plotbds.m`	Plot bounds for binary codes	408
`simplex1.m`	Linear program in standard form	408
`pivottableau.m`	Main function in `simplex1.m`	414
`reducefree.m`	Auxiliary linear programming function	414
`restorefree.m`	Auxiliary linear programming function	414
`krawtchouk.m`	Compute Krawtchouk polynomials recursively	415
`lpboundex.m`	Solve the linear programming for the LP bound	418
`utiltkm.cc`	Sort and random functions	440
`utiltkm.h`	Sort and random functions	440
`concodequant.m`	Quantization of the Euclidean metric	487
`chernoff1.m`	Chernoff bounds for conv. performance	502
`plotconprob.m`	Performance bounds for a conv. code	504
`finddfree`	**Executable:** Find d_{free}	506
`teststack.m`	Test the stack algorithm	514
`stackalg.m`	The stack algorithm for convolutional decoding	514
`fanomet.m`	Fano metric for convolutionally coded data	514
`fanoalg.m`	The Fano algorithm for convolutional decoding	516
`BinConv.h`	(lab, complete) Base class for bin. conv. enc.	531
`BinConvFIR.h`	(lab, complete) Bin. feedforward conv. enc.	536
`BinConvFIR.cc`	(**incomplete**) Bin. feedforward conv. encoder	536
`BinConvIIR.h`	(lab, complete) Binary recursive conv. enc.	536
`BinConvIIR.cc`	(lab, **incomplete**) Binary recursive conv. enc.	536
`testconvenc`	(lab, complete) Test convolutional encoders	536
`Convdec.h`	(lab, complete) Convolutional decoder class	533
`Convdec.cc`	(lab, **incomplete**) Convolutional decoder class	533
`BinConvdec01.h`	(lab, complete) Binary conv. decoder, BSC	533
`BinConvdecBPSK.h`	(lab, complete) Bin. conv. dec., BPSK	538
`BinConvdecBPSK.cc`	(lab, complete) Bin. conv. dec., BPSK	538
`testconvdec.cc`	(lab, complete) Test the convolutional decoder	538
`makeB.m`	Make the B matrix for an example code	557
`tcmt1.cc`	Test constellation for rotationally invariant code	557
`tcmrot2.cc`	Test constellation for rotationally invariant code	565
`lattstuff.m`	Gen. matrices for A_2, D_4, E_6, E_8, Λ_{16}, Λ_{24}	570
`voln.m`	Volume of n-dimensional unit-radius sphere	570
`latta2.m`	Plot A_2 lattice	576
`lattz2m`	Plot Z_2 lattice	576
`BCJR.h`	(lab, complete) BCJR algorithm class header	631
`BCJR.cc`	(lab, **incomplete**) BCJR algorithm class	631
`testbcjr.cc`	(lab, complete) Test BCJR algorithm class	631
`testturbodec2.cc`	(lab, complete) Test the turbo decoder	631
`makegenfromA.m`	Systematic gen. matrix from parity check matrix	638

`gaussj2.m`	Gauss–Jordan elimination over $GF(2)$	638
`Agall.m`	A parity check matrix for an LDPC code	640
`Agall.txt`	Sparse representation of the matrix	640
`writesparse.m`	Write a matrix into a file in sparse format	640
`ldpcdecoder.cc`	Many different LDPC decoding algorithms	642
`ldpctest2.cc`	ldpcdecoder runner	642
`decode()`	Decoding in class LDPCDECODER	649
`lldecode()`	LL BP decoding in class LDPCDECODER	649
`probdecode()`	Probability BP decoding	655
`ldpc.m`	Demonstrate LDPC decoding	656
`ldpcdecode.m`	LDPC decoder (nonsparse representation)	656
`ldpclogdecode.m`	LDPC decoder (log likelihood)	656
`phidecode()`	Belief propagation decoding using ϕ function	660
`qphidecode()`	Quantized ϕ decoding	660
`minsumdecode()`	Min sum decoder	661
`minsumcorrectdecode()`	Corrected min sum decoder	659
`aminstartdecode()`	Approximation min* decoder	662
`rcbpdecode()`	Reduce complexity box-plus decoder	665
`showchktable.m`	Compare RCBP quantization and $\boxplus$	667
`bitflip1decode()`	Bit Flipping LDPC decoder	670
`galAbitflipdecode()`	Gallager Algorithm A Bit Flipping Decoder	671
`weightedbitflip decode()`	Weighted bit flipping decoder	673
`modwtedbitflipdecode()`	Modified weighted bit flipping decoder	673
`grad1bitflipdecode()`	Gradient descent bit flipping	673
`grad2bitflipdecode()`	Multigradient bit flip	673
`DC1decode()`	LDPC decoding using divide and concur	674
`DMBPdecode()`	DMBP decoding	682
`LPdecode()`	Linear Programming LDPC Decoding	682
`ldpctest1`	LDPC err. prob. & iteration plots	695
`psifunc.m`	Plot the Ψ function used in density evolution	697
`densev1.m`	An example of density evolution	698
`densevtest.m`	Plot density evolution results	698
`Psi.m`	Plot the Ψ function used in density evolution	698
`Psiinv.m`	Compute Ψ^{-1} used in density evolution	698
`threshtab.m`	Convert threshold table to E_b/N_0	698
`ldpcsim.mat`	LDPC decoder simulation results	700
`exit1.m`	Plot histograms of LDPC decoder outputs	700
`loghist.m`	Find histograms	700
`exit3.m`	Plot mutual information as a function of iteration	700
`dotrajectory.m`	Mutual information as a function of iteration	700
`exit2.m`	Plot EXIT chart	700
`doexitchart.m`	Take mutual information to EXIT chart	700
`getinf.m`	Convert data to mutual information	700
`getinfs.m`	Convert multiple data to mutual information	700
`sparseHno4.m`	Make sparse check matrix without girth 4	708
`Asmall.txt`	H matrix example	711
`ldpcdec.h`	(lab, complete) LDPC decoder class header	711
`ldpcdec.cc`	(lab, **incomplete**) LDPC decoder class	711
`ldpctest.cc`	(lab, complete) LDPC decoder class tester	711
`ldpc.m`	(lab, complete) Demonstrate LDPC decoding	711
`ldpcdecode.m`	LDPC decoder (not sparse)	711

`ldpctest1.cc`	Prob. of err. plots for LDPC code	711
`A1-2.txt`	Rate 1/2 parity check matrix, sparse format	
`A1-3.txt`	Rate 1/3 parity check matrix, sparse format	711
`gf257ex.m`	LDPC check matrix min. wt. RS codewords	732
`ldpcencode.m`	Encode LDPC codes	732
`ldpccodesetup1.m`	Make parity check matrix for DVB-T2 IRA code	732
`fadeplot.m`	Realizations of the amplitude of a fading channel	903
`jakes.m`	Jakes method for amplitude of a fading channel	903
`fadepbplot.m`	BPSK performance over Rayleigh fading channel	904

List of Laboratory Exercises

Lab 1 Simulating a Communications Channel 60
Lab 2 Polynomial Division and Linear Feedback Shift Registers 168
Lab 3 CRC Encoding and Decoding 170
Lab 4 Programming the Euclidean Algorithm 228
Lab 5 Programming Galois Field Arithmetic 229
Lab 6 Programming the Berlekamp–Massey Algorithm 287
Lab 7 Programming the BCH Decoder 289
Lab 8 Reed–Solomon Encoding and Decoding 290
Lab 9 Programming Convolutional Encoders 535
Lab 10 Convolutional Decoders: The Viterbi Algorithm 537
Lab 11 Trellis-Coded Modulation Encoding and Decoding 586
Lab 12 Turbo Code Decoding 631
Lab 13 Programming an LDPC Decoder 710

List of Algorithms

1.1 Hamming Code Decoding 34
1.2 Outline for Simulating Digital Communications 60
1.3 Outline for Simulating (n, k)-coded Digital Communications 61
1.4 Outline for Simulating (n, k) Hamming-coded Digital Communications 61

4.1 Fast CRC encoding for a stream of bytes. 161
4.2 Binary Linear Feedback Shift Register 169
4.2 Binary Polynomial Division 170

5.1 Extended Euclidean Algorithm 189
5.2 Modulo Arithmetic 228
5.3 Templatized Polynomials 229
5.4 Polynomial GCD 229
5.5 $GF(2^m)$ Arithmetic 230

6.1 Massey's Algorithm (Pseudocode) 265
6.2 Massey's Algorithm for Binary BCH Decoding 266
6.3 Test Berlekamp–Massey Algorithm 288
6.4 Chien Search 289
6.5 BCH Decoder 289
6.6 Reed–Solomon Encoder Declaration 290
6.7 Reed–Solomon Decoder Declaration 291
6.8 Reed–Solomon Decoder Testing 291
6.9 Reed–Solomon File Encoder and Decoder 291
6.10 Binary Symmetric Channel Simulator 291

7.1 Welch–Berlekamp Interpolation 314
7.2 Welch–Berlekamp Interpolation, Modular Method v.1 321
7.3 Welch–Berlekamp Interpolation, Modular Method v.2 324
7.4 Welch–Berlekamp Interpolation, Modular Method v.3 325
7.5 The Feng–Tzeng Algorithm 343
7.6 Kötters Interpolation for Guruswami–Sudan Decoder 348
7.7 Guruswami–Sudan Interpolation Decoder with $m = 1$ and $L = 1$ 352
7.8 Roth–Ruckenstein Algorithm for Finding y-roots of $Q(x, y)$ 356
7.9 Koetter–Vardy Algorithm for Mapping from Π to M 362

8.1 Decoding for $RM(1, m)$ Codes 382
8.2 Arithmetic Decoding of the Golay $\mathcal{G}_{24}$ Code 403

11.1 Generalized Minimum Distance (GMD) Decoding 441
11.2 Chase-2 Decoder 445
11.3 Chase-3 Decoder 445
11.4 Ordered Statistic Decoding 448

12.1 The Viterbi Algorithm 477
12.2 The Stack Algorithm 514
12.3 The SOVA 532
12.4 Base Class for Binary Convolutional Encoder 535
12.5 Derived Classes for FIR and IIR Encoders 536
12.6 Test Program for Convolutional Encoders 536
12.7 Base Decoder Class Declarations 537
12.8 Convolutional Decoder for Binary (0,1) Data 537
12.9 Convolutional Decoder for BPSK Data 538
12.10 Test the Convolutional Decoder 538

14.1 The BCJR (MAP) Decoding Algorithm, Probability Form 603
14.2 The Turbo Decoding Algorithm, Probability Form 608
14.3 BCJR Algorithm 631
14.4 Test the turbo decoder Algorithm 631

15.1 Log-Likelihood Decoding Algorithm for Binary LDPC Codes 649
15.2 Probability Algorithm for Binary LDPC Codes 655
15.3 a-Min* Algorithm for Check Node Computation 665
15.4 Quantize the chk Function 667
15.5 Bit Flipping Decoder 670
15.6 Gallager Algorithm A Bit Flipping Decoder 671
15.7 Weighted Bit Flipping Decoder 673
15.8 Divide and Concur LDPC Decoder 678
15.9 Difference Map Belief Propagation Decoder 682
15.10 LDPC Decoder Class Declarations 711
15.11 MATLAB Code to Test LDPC Decoding 711
15.12 Make Performance Plots for LDPC Codes 711

16.1 Repeat Accumulate Decoding 720
16.2 Progressive Edge Growth 747

17.1 Polar Code Encoding 797
17.2 Successive Cancellation Decoder (Pseudocode) 824
17.3 Polar Code Design Using Bhattacharrya Parameter 838
17.4 Polar Code Design Using Monte Carlo Simulation 839
17.5 Polar Code Systematic Encoder 842
17.6 Polar List Decoder: Recursively Calculate P 854
17.7 Polar List Decoder: Recursively Calculate C 856
17.8 Polar List Decoder: Main Loop 857
17.9 Polar List Decoder: Build Data Structures 858
17.10 Polar List Decoder: Initialize Data Structures 859
17.11 Polar List Decoder: Assign Initial Path 859
17.12 Polar List Decoder: Clone a Path 859
17.13 Polar List Decoder: Kill a Path 860
17.14 Polar List Decoder: Get Pointer P_λ^ℓ 860
17.15 Polar List Decoder: Get Pointer C_λ^ℓ 860
17.16 Polar List Decoder: Continue Paths for Unfrozen Bit 861

List of Figures

1.1 The binary entropy function $H_2(p)$. 5
1.2 A general framework for digital communications. 6
1.3 Signal constellation for BPSK. 11
1.4 Juxtaposition of signal waveforms. 11
1.5 Two-dimensional signal space. 12
1.6 8-PSK signal constellation. 13
1.7 Correlation processing (equivalent to matched filtering). 15
1.8 Conditional densities in BPSK modulation. (a) Conditional densities; (b) weighted conditional densities. 18
1.9 Distributions when two signals are sent in Gaussian noise. 20
1.10 Probability of error for BPSK signaling. 21
1.11 Probability of error bound for 8-PSK modulation. 22
1.12 A binary symmetric channel. 23
1.13 Communication system diagram and BSC equivalent. 25
1.14 Energy for a coded signal. 26
1.15 Probability of error for coded bits, before correction. 26
1.16 A $(3, 1)$ binary repetition code. (a) The code as points in space; (b) the Hamming spheres around the points. 28
1.17 A representation of decoding spheres. 29
1.18 Performance of the $(3, 1)$ and $(11, 1)$ repetition code over BSC. 31
1.19 Performance of the $(7, 4)$ Hamming code in the AWGN channel. 35
1.20 The trellis of a $(7, 4)$ Hamming code. 37
1.21 The Tanner graph for a $(7, 4)$ Hamming code. 38
1.22 Illustration of data-processing inequality. 44
1.23 BSC and BEC models. (a) Binary symmetric channel with crossover; (b) binary erasure channel with erasure probability. 45
1.24 Capacities of AWGNC, BAWGNC, and BSC. 50
1.25 Relationship between input and output entropies for a channel. 51
1.26 Capacity lower bounds on P_b as a function of SNR. 55
1.27 Rate-blocklength tradeoff for the BSC with $\delta = 0.11$ and maximal block error rate $\epsilon = 10^{-3}$. 59
1.28 Rate-blocklength tradeoff for the Gaussian channel with $P = 1$ and maximal block error rate $\epsilon = 10^{-3}$. 60
1.29 Regions for bounding the Q function. 65

2.1 An illustration of cosets. 76
2.2 A lattice partitioned into cosets. 81

3.1 Error detection performance for a $(15, 11)$ Hamming code. 110
3.2 Demonstrating modifications on a Hamming code. 115

4.1 Addition and multiplication tables for $GF(2)[x]/\langle x^3 + 1\rangle$. 127
4.2 A circuit for multiplying two polynomials, last-element first. 138
4.3 High-speed circuit for multiplying two polynomials, last-element first. 138
4.4 A circuit for multiplying two polynomials, first-element first. 139
4.5 High-speed circuit for multiplying two polynomials, first-element first. 139
4.6 A circuit to perform polynomial division. 139
4.7 A circuit to divide by $g(x) = x^5 + x + 1$. 141
4.8 Realizing $h(x)/g(x)$ (first-element first), controller canonical form. 142
4.9 Realizing $h(x)/g(x)$ (first-element first), observability form. 142
4.10 Realization of $H(x) = (1 + x)/(1 + x^3 + x^4)$, controller form. 142
4.11 Realization of $H(x) = (1 + x)/(1 + x^3 + x^4)$, H observability form. 143

4.12 Nonsystematic encoding of cyclic codes. 144
4.13 Circuit for systematic encoding using $g(x)$. 145
4.14 Systematic encoder for $(7, 4)$ code with generator $g(x) = 1 + x + x^3$. 146
4.15 A systematic encoder using the parity-check polynomial. 146
4.16 A systematic encoder for the Hamming code using $h(x)$. 147
4.17 A syndrome computation circuit for a cyclic code example. 148
4.18 Cyclic decoder with $r(x)$ shifted in the left end of syndrome register. 149
4.19 Decoder for a $(7, 4)$ Hamming code, input on the left. 150
4.20 Cyclic decoder when $r(x)$ is shifted in right end of syndrome register. 151
4.21 Hamming decoder with input fed into right end of the syndrome register. 152
4.22 Meggitt decoders for the $(31, 26)$ Hamming code. (a) Input from the left end; (b) Input from the right end. 153
4.23 Multiply $r(x)$ by $\rho(x)$ and compute the remainder modulo $g(x)$. 154
4.24 Decoder for a shortened Hamming code. 155
4.25 Linear feedback shift register. 162
4.26 Linear feedback shift register with $g(x) = 1 + x + x^2 + x^4$. 162
4.27 Linear feedback shift register with $g(x) = 1 + x + x^4$. 164
4.28 Linear feedback shift register, reciprocal polynomial convention. 165
4.29 Another LFSR circuit. 176
4.30 An LFSR with state labels. 177

5.1 LFSR labeled with powers of α to illustrate Galois field elements. 207
5.2 Multiplication of β by α. 212
5.3 Multiplication of an arbitrary β by α^4. 212
5.4 Multiplication of β by an arbitrary field element. 213
5.5 Subfield structure of $GF(2^{24})$. 214

6.1 Chien search algorithm. 256

7.1 Remainder computation when errors are in message locations. 301
7.2 Comparing BD, ML, and list decoding. (a) BD decoding: codeword in same Hamming sphere as **r**; (b) ML decoding: nearest codeword to **r** decoding (possibly beyond Hamming sphere); (c) List decoding: all codewords within a given Hamming distance of **r**. 327
7.3 K_m as a function of m for a $(32, 8)$ Reed–Solomon code. 337
7.4 Fraction of errors corrected as a function of rate. 338
7.5 An example of the Roth–Ruckenstein Algorithm over $GF(5)$. 364
7.6 An example of the Roth–Ruckenstein Algorithm over $GF(5)$ (cont'd). 365
7.7 Convergence of $\bar{M}$ to Π. 366
7.8 Computing the reliability function. 366

8.1 An encoder circuit for a $RM(1, 3)$ code. 381
8.2 Signal flow diagram for the fast Hadamard transform. 385
8.3 Binary adjacency relationships in (a) three and (b) four dimensions. 389
8.4 Planes shaded to represent the equations orthogonal on bit m_{34}. 389
8.5 Parity check geometry for second-order vectors for the $RM(2, 4)$ code. 391
8.6 Parity check geometry for first-order vectors for the $RM(2, 4)$ code. 392

9.1 Comparison of lower bound and various upper bounds. 408
9.2 A linear programming problem. 414
9.3 Finding Stirling's formula. (a) Under bound; (b) over bound. 422

10.1 A 3×4 interleaver and deinterleaver. 426
10.2 A cross interleaver and deinterleaver system. 427
10.3 The CD recording and data formatting process. 428
10.4 The error-correction encoding in the compact disc standard. 429
10.5 The product code $C_1 \times C_2$. 430
10.6 A concatenated code. 431
10.7 Deep-space concatenated coding system. 432
10.8 Error trapping decoder for burst-error-correcting codes. 435

11.1 Signal labels for soft-decision decoding. 440

12.1 A rate $R = 1/2$ convolutional encoder. 456
12.2 A systematic $R = 1/2$ encoder. 457
12.3 A systematic $R = 2/3$ encoder. 458
12.4 A systematic $R = 2/3$ encoder with more efficient hardware. 458
12.5 Encoder, state diagram, and trellis for $G(x) = [1 + x^2, 1 + x + x^2]$. (a) Encoder; (b) state diagram; (c) one stage of trellis; (d) three stages of trellis. 461
12.6 State diagram and trellis for a rate $R = 2/3$ systematic encoder. 462
12.7 A feedforward $R = 2/3$ encoder. 463
12.8 A less efficient feedforward $R = 2/3$ encoder. 463
12.9 Processing stages for a convolutional code. 472
12.10 Notation associated with a state transition. 473
12.11 The Viterbi step: select the path with the best metric. 475
12.12 Path through trellis corresponding to true sequence. 478
12.13 Add-compare-select operation. 483
12.14 A 2-bit quantization of the soft-decision metric. 487
12.15 Quantization thresholds for four- and eight-level quantization. 488
12.16 Bit error rate for different constraint lengths. 489
12.17 Bit error rate for various quantization and window lengths. 490
12.18 Viterbi algorithm performance as a function of quantizer threshold spacing. 491
12.19 BER performance as a function of truncation block length. 491
12.20 Error events due to merging paths. 492
12.21 Two decoding examples. (a) Diverging path of distance 5; (b) diverging path of distance 6. 493
12.22 The state diagram and graph for diverging/remerging paths. (a) State diagram; (b) split state 0; (c) output weight irepresented by Di. 494
12.23 Rules for simplification of flow graphs. 495
12.24 Steps simplifying the flow graph for a convolutional code. 495
12.25 State diagram labeled with input/output weight and branch length. 496
12.26 A state diagram to be enumerated. 497
12.27 Performance of a $(2, 1)$ convolutional code with $d_{\text{free}} = 5$. 504
12.28 Trellises for a punctured code. (a) Trellis for initial punctured code; (b) trellis by collapsing two stages of the initial trellis into a single stage. 509
12.29 A tree representation for a rate $R = 1/2$ code. 511
12.30 Stack contents for stack algorithm decoding example. 515
12.31 Flowchart for the Fano algorithm. 517
12.32 Computations of SOVA at $t = 3$ (see Example 12.23). 529
12.33 Computations of SOVA at $t = 4$. 531
12.34 The trellis of a $(7, 4)$ Hamming code. 534
12.35 A systematic encoder for a $(7, 4, 3)$ Hamming code. 534

12.36 A trellis for a cyclically encoded $(7, 4, 3)$ Hamming code. 535
12.37 (a) State diagram and (b) trellis. 536

13.1 PSK signal constellations. 546
13.2 QAM signal constellations (overlaid). 546
13.3 Three communication scenarios. (a) QPSK, no coding; (b) QPSK, $R = 2/3$ coding; (c) 8PSK, $R = 2/3$ coding. 548
13.4 Set partitioning of an 8-PSK signal. 549
13.5 $R = 2/3$ trellis-coded modulation example. 549
13.6 A TCM encoder employing subset selection and a four-state trellis. 551
13.7 An eight-state trellis for 8-PSK TCM. 552
13.8 Block diagram of a TCM encoder. 553
13.9 Set partitioning on a 16-QAM constellation. 554
13.10 Partition for 8-ASK signaling. 554
13.12 A correct path and an error path. 557
13.11 Example trellis for four-state code. 557
13.13 Trellis coder circuit. 560
13.14 TCM encoder for QAM constellations. 564
13.15 Mapping of edge (i,j) to edge $(f_\phi(i),\ f_\phi(j))$. 564
13.16 Encoder circuit for rotationally invariant TCM code. 565
13.17 Trellis for the rotationally invariant code of Figure 13.16. 565
13.18 32-Cross constellation for rotationally invariant TCM code. 566
13.19 A portion of the lattice $\mathbb{Z}^2$ and its cosets. (a) $\mathbb{Z}^2$ lattice; (b) cosets of the lattice. 571
13.20 Hexagonal lattice. (a) Basic lattice; (b) the fundamental parallelotopes around the lattice points. 571
13.21 $\mathbb{Z}^2$ and its partition chain and cosets. (a) $\mathbb{Z}^2$; (b) cosets of $R\mathbb{Z}^2$; (c) Cosets of $R^2\mathbb{Z}^2$. 575
13.22 Block diagram for a trellis lattice coder. 575
13.23 Lattice and circular boundaries for various constellations. 576
13.24 16-State trellis encoder for use with V.34 standard. 579
13.25 Trellis diagram of V.34 encoder [?]. 580
13.26 The 192-point constellation employed in the V.34 standard. 581
13.27 Partition steps for the V.34 signal constellation. 582
13.28 Orbits of some of the points under rotation. 583

14.1 Decoding results for a $(37, 21, 65536)$ code. 592
14.2 Block diagram of a turbo encoder. 593
14.3 Block diagram of a turbo encoder with puncturing. 594
14.4 Example turbo encoder with $G(x) = 1/(1 + x^2)$. 594
14.5 Block diagram of a turbo decoder. 595
14.6 Processing stages for BCJR algorithm. 597
14.7 One transition of the trellis for the encoder. 598
14.8 A log-likelihood turbo decoder. 611
14.9 State sequences for an encoding example. 619
14.10 Arrangements of $n_1 = 3$ detours in a sequence of length $N = 7$. 620
14.11 A 6×6 "square" sequence written into the 120×120 permuter. 622
14.12 Variables used in iterative decoder for EXIT chart analysis. 623
14.13 Qualitative form of the transfer characteristic $I_E = T(I_A)$. 625
14.14 Trajectories of mutual information in iterated decoding. 625
14.15 Turbo BCH encoding. 626
14.16 Structure of an implementation of a parallel concatenated code. 627
14.17 A trellis for a cyclically encoded $(7, 4, 3)$ Hamming code. 628
14.18 Framework for a turbo equalizer. 629

14.19 Trellis associated with a channel with $L = 2$. 630
14.20 Example for a $(3, 2, 2)$ parity-check code. (a) A simple $(3, 2, 2)$ parity check code; (b) received values $L_c r_t$. 632

15.1 Bipartite graph associated with the parity check matrix H. 641
15.2 Structure of LDPC decoding. 642
15.3 A check node with four connections. 645
15.4 The message from check node C_m to variable node V_n. 646
15.5 The message from variable node V_n to check node C_m. 647
15.6 A parity-check tree associated with the Tanner graph. 652
15.7 A portion of the Tanner graph for the example code, with node for V_1 as a root, showing a cycle. 652
15.8 Processing information through the graph determined by H. 653
15.9 Probabilistic message from check node m to variable nodes n. 654
15.10 The function $\phi(x)$. 660
15.11 Comparison between LL-based computing and min-sum computing for check to variable node computation. (a) Original comparison; (b) scaled comparison. 661
15.12 $s(x, y)$, the approximation $\tilde{s}(x, y)$, $L_1 \boxplus L_2$, and the approximation using $\tilde{s}(x, y)$. (a) $s(x, y)$; (b) $\tilde{s}(x, y)$; (c) exact $L_1 \boxplus L_2$; (d) approximate $L_1 \boxplus L_2$. 664
15.13 Comparison of $\boxplus$ and its RCBP quantization. 668
15.14 Approximations to the $\phi(\beta)$ function for the Richardson–Novichkov decoder. 669
15.15 Steps in the DC algorithm. (a) Projection of $\mathbf{r}_{[0]}$ and $\mathbf{r}_{[1]}$ onto $\mathcal{P}_0$ and $\mathcal{P}_1$; (b) $\mathbf{r}_{(m)} + 2(P_D^m(\mathbf{r}_{(m)}) - \mathbf{r}_{(m)})$ (Overshoot); (c) concur projection, and results of difference map. 675
15.16 Linear programming problem. 683
15.17 Polytope of the codewords $(0, 0, 0), (0, 1, 1), (1, 1, 0), (1, 0, 1)$. 685
15.18 Tanner graph for the Hamming $(7, 4)$ code. 686
15.19 Number of constraints κ_m as a function of $|\mathcal{N}(m)|$. 690
15.20 Hamming code Tanner graph. Solid = 1, dotted = 0, heavy dashed = erasure. (a) Tanner graph; (b) graph with deleted edges; (c) after one erasure corrected; (d) after two erasures corrected. 692
15.21 Stopping sets. (a) Idea of stopping sets; (b) stopping set $\{v_0, v_4, v_6\}$ on a $(9, 3)$ example code [442]. 693
15.22 Illustration of the decoding performance of LPDC codes. 694
15.23 Comparison of hard-decision Hamming and iterative decoding. 695
15.24 The function $\Psi(x)$ compared with $tanh(x/2)$. 697
15.25 Behavior of density evolution for a $R = 1/3$ code. 699
15.26 Messages from bits to checks and from checks to bits. 700
15.27 Histograms of the bit-to-check information for various decoder iterations. 701
15.28 Decoder information at various signal-to-noise ratios. 701
15.29 EXIT charts at various signal-to-noise ratios. 702
15.30 Result of permutation of rows and columns on parity check matrix. 708

16.1 A repeat-accumulate encoder. 718
16.2 The Tanner graph for a $(3, 1)$ RA code with two input bits. 718
16.3 Decoding steps for an RA code. (a) Initialization/parity bit to check; (b) check to message; (c) message to check; (d) check to parity bit. 720
16.4 Tanner graph for an irregular repeat-accumulate code. 721
16.5 Example Tanner graph for an irregular repeat-accumulate code. 722
16.6 An IRAA encoder. 723
16.7 Encoding QC codes. (a) cyclic shift register adder accumulator (CSRAA); (b) parallelizing the CRSAA. 727
16.8 $(3, 6)$ Ensemble. 743

16.9 Illustrating an irregular ensemble. 743
16.10 An example MET ensemble. 745
16.11 Notation for the PEG algorithm. 747
16.12 Performance of a $(4000, 2000)$ LDPC code founding using PEG. 748
16.13 Graphs related to protograph description of LDPC code. (a) Initial protograph from adjacency matrix in 16.10; (b) repeating nodes $Q = 3$ times, and permutations of repeated edges; (c) detailed Tanner graph derived from protograph. 749
16.14 Subgraphs associated with trapping sets. ∘ = variable nodes. ■ = unsatisfied check nodes. □ = incorrectly satisfied check nodes. (a) $(12, 4)$ trapping set; (b) $(14, 4)$ trapping set; (c) $(5, 5)$ trapping set; (d) $(5, 7)$ trapping set. 751
16.15 Probability of error for a BPSK detection problem using MC and IS sampling. (a) Distribution for Monte Carlo integration; (b) optimal sampling distribution for this problem; (c) distribution for IS integration. 755
16.16 Best gain and shift ζ for an importance sampling example. (a) γ and P_E as a function of σ^2. P_E decreases when γ increases; (b) z as a function of σ^2. 758
16.17 Probability of error for three-symbols detection problem using MC and IS sampling. (a) Distribution for Monte Carlo integration; (b) optimal sampling distribution q^* for this problem; (c) distribution for IS integration. 759
16.18 Points chosen at random on a circle of radius $\rho = 2$ with Gaussian noise added of covariance $\tilde{K} = K/n$, where $K = \sigma^2 I_2$ with $\sigma^2 = 1$. (a) $n = 0.5$; (b) $n = 1$; (c) $n = 2$. 759
16.19 An example constraint set A. 760
16.20 Example constellation. (a) Constellation; (b) error region; (c) spherical offset. 761
16.21 Geometry of coded symbol points. (a) The sphere of codewords around $\mathbf{0}$, and the sphere of codewords around $\mathbf{s}_0$; (b) side view of two spheres; (c) plane in $\mathbb{R}^{N-1}$ on which intersecting hypersphere lies. 764
16.22 Decision boundary for $(3, 2)$ single parity-check coding ($\sigma = 0.1$). Squares denote codewords. 767
16.23 A $(3, 2, 2)$ tree and labeling for a particular boundary. (a) Tree; (b) indication of variable nodes for the $\partial E^{(2,4,6,8,14,16)}$ boundary. 768
16.24 Trees which are not symmetric. (a) Variable nodes 2 and 3 and nodes 4 and 5 isormophically interchangeable; (b) variable nodes 2 and 3 isormophically interchangeable. 770
16.25 Tanner graph for $(9, 3)$ regular code, and its unfolding into a tree. (a) Tanner graph for the code; (b) unfolded tree; (c) pruned unfolded tree. 771
16.26 Graph defining Tornado codes. (a) One stage of code graph; (b) decoding an erased value; (c) two stages of check bits; (d) overall Tornado code with $m + 1$ stages. 773

17.1 Error correction performance of polar codes compared with an LDPC code of H the same rate. (a) Results for a (2048,1024) polar code, with list decoding (see [431]); (b) comparison of $R = 0.84$ polar codes with LDPC code of same rate (see [391]). (Color versions of the page can be found at wileywebpage or `https://github.com/tkmoon/eccbook/colorpages`.). 780
17.2 Basic ideas of polarization. (a) Two uses of the channel W; (b) simple coding and two channel uses; (c) recursively applying the coding construction, $N = 4$; (d) recursively applying the coding construction, $N = 8$ with some bits frozen. 780
17.3 First stage of the construction of the polar transform. (a) The basic building block; (b) representation in terms of the channels $W_2^{(0)}$ and $W_2^{(1)}$. 783
17.4 $g(p)$ and $h(p)$ for $p \in [0, 1]$, showing that $g(p) > h(p)$. 788
17.5 Successive cancellation on $N = 2$ channels. (a) Upper log-likelihood ratio computation; (b) lower log-likelihood ratio computation. 790
17.6 Comparison of tanh rule (solid lines) with (17.26) (dashed lines). 791

17.7 Tree representation of the encoding. (a) Encoder $N = 2$; (b) converting to a tree; (c) converted to a tree; (d) rotated with the root node at the top. 792
17.8 Repeating the basic construction by duplicating W_2 to form W_4. (a) Duplication of W_2; (b) creation of $W_4^{(0)}$ and $W_4^{(1)}$; (c) creation of $W_4^{(2)}$ and $W_4^{(3)}$; (d) reordering the inputs of W_4. 794
17.9 Another representation of W_4. 795
17.10 General recursion to go from two $W_{N/2}$ channels to W_N. (a) Basic form of general recursion to go from two $W_{N/2}$ channels to W_N; (b) alternative form for the recursion. . . 797
17.11 The channel W_8. 798
17.12 The tree of recursively constructed channels. 801
17.13 Tree decomposition of the information in the W_4 channels. 802
17.14 Computing the synthetic BEC erasure probabilities on a tree. 804
17.15 Polarization effect for the BEC channel. 804
17.16 BEC probability for polarized channels, $p = 0.3$. (a) Unsorted probabilities, 8 iterations; (b) sorted probabilities, 8 iterations; (c) sorted probabilities, 16 iterations. 806
17.17 A tree representation of the decoder structure. (a) Basic encoding structure; (b) basic encoding structure, sequential bit order; (c) identifying frozen and unfrozen nodes, and the groupings of adders; (d) first stage of converting groupings into nodes, with nodes colored to match colorings of children; (e) next stage of converting groupings into nodes; (f) final stage of converting groupings into nodes. 808
17.18 Rotated tree and messages in the tree. (a) Rotating the tree to put the root node at the top; (b) a portion of a tree, showing messages passed in a local decoder. 809
17.19 Encoder for $N = 4$ and its associated tree. (a) Encoder illustrating node groupings; (b) decoding tree for $N = 4$ with messages labeled. 811
17.20 Decoding steps for $N = 4$. (a) $L_4^{(0)}$; (b) $L_4^{(1)}$; (c) $L_4^{(2)}$; (d) $L_4^{(3)}$. (Color versions of the page can be found at wileywebpage or `https://github.com/tkmoon/eccbook/colorpages`.) 814
17.21 Decoding tree for $N = 8$ with messages labeled. 814
17.22 The structure used for the decoding example. 815
17.23 Successive cancellation steps, $j = 0, 1, 2$, $i = 0, 4, 2$. (a) LLR path for bit $j = 0$. $i = 0$, `level1` = 3; (b) bit propagation, $j = 0, i = 0$; (c) LLR path for bit $j = 1$. $i = 4$, `level1` = 1; (d) bit propagation $j = 1$. $i = 4$, `level0` = 2; (e) LLR path for bit $j = 2, i = 2$. `level1` = 2; (f) bit propagation $j = 2, i = 2$. (Color versions of the page can be found at wileywebpage or `https://github.com/tkmoon/eccbook/colorpages`.). 816
17.24 Successive cancellation steps, $j = 3, 4, 5$, $i = 6, 1, 5$. (a) LLR path for bit $j = 3, i = 6$. `level1` = 1; (b) bit propagation for bit $j = 3, i = 6$. `level0` = 3; (c) LLR path for bit $j = 4, i = 1$. `level1` = 3; (d) bit propagation $j = 4, i = 1$; (e) LLR path for bit $j = 5, i = 5$. `level1` = 1; (f) bit propagation $j = 5, i = 5$. `level0` = 2. (Color versions of the page can be found at wileywebpage or `https://github.com/tkmoon/eccbook/colorpages`.). 820
17.25 Successive cancellation steps, $j = 6, 7$, $i = 3, 7$. (a) LLR path for bit $j = 6, i = 3$. `level1` = 2; (b) bit propagation $j = 6, i = 3$; (c) LLR path for bit $j = 7, i = 7$. `level1` = 1. (Color versions of the page can be found at wileywebpage or `https://github.com/tkmoon/eccbook/colorpages`.). 823
17.26 $Z(W)$, $I(W)$ and the bounds from 17.59 and 17.60. (a) BEC; (b) BSC. 827
17.27 Calculations for systematic polar encoding. 841
17.28 A systematic encoder for an $(8, 5)$ polar code [396]. 844
17.29 Evolution of paths in list decoding. Following [435]. 851
17.30 Encoder for $N = 4$. 852

17.31 Data structure P used in list decoding with $N = 8$, $n = 3$, $L = 4$. Numbers in the braces indicate the dimension of the array. 853
17.32 A two-step list decoder. 865
17.33 A tree labeled to illustrate the notation for simplified successive cancellation decoding. . 868
17.34 Identifying rate one nodes (black) and a rate zero nodes (white) in the tree, and the reduced tree. (a) Rate one nodes and rate zero nodes; (b) the reduced tree. 868
17.35 Plot of $E_0(\rho, Q)$. 876

18.1 FEC encoding structure (see [107, Figure 12]). 888
18.2 Concatenated Reed–Solomon and Trellis coding for television transmission (see [204]). 892
18.3 Trellis encoder ([204, Figure 2]). 893
18.4 Set partition of 64-QAM. At the last stage, different shapes are used to indicate the different subsets. (This decomposition is for example here, and is not indicated in [204].) 894
18.5 Trellis decoder ([204, Figure 3]). 894
18.6 Distances from $(\hat{I}, \hat{Q})$ to subsets in the 64-QAM constellation. 894
18.7 Rate 4/5 convolutional encoder. 895
18.8 OFDM transmission over a frequency-selective channel (without and with coding) (see [87, Section 4.7]). (a) OFDM transmission over a frequency-selective channel; (b) channel coding and interleaving to provide frequency diversity. 896
18.9 Rate 1/3 convolutional encoder for LTE [1, Figure 2]. 897
18.10 LTE rate 1/3 turbo code encoder [1, Figure 3]. 898

19.1 Multiple reflections from transmitter to receiver. 902
19.2 Simulation of a fading channel. 903
19.3 Diversity performance of quasi-static, flat-fading channel with BPSK modulation. Solid is exact; dashed is approximation. 905
19.4 Multiple transmit and receive antennas across a fading channel. 906
19.5 Two receive antennas and a maximal ratio combiner receiver. 907
19.6 A two-transmit antenna diversity scheme: the Alamouti code. 909
19.7 A delay diversity scheme. 917
19.8 8-PSK constellation and the trellis for a delay-diversity encoder. 918
19.9 Space-time codes with diversity 2 for 4-PSK having 8, 16, and 32 states [435]. 919
19.10 Performance of codes with 4-PSK that achieve diversity 2 [435]. (a) Two receive and two transmit antennas; (b) one receive and two transmit antennas.. 920
19.11 Space-time codes with diversity 2 for 8-PSK having 16 and 32 states [435]. 921
19.12 Performance of codes with 8-PSK that achieve diversity 2 [435]. (a) Two receive and two transmit antennas; (b) one receive and two transmit antennas.. 921

List of Tables

1.1 Historical Milestones 40

3.1 The Standard Array for a Code 101

4.1 Codewords in the Code Generated by $g(x) = 1 + x^2 + x^3 + x^4$ 132
4.2 Systematic Codewords in the Code Generated by $g(x) = 1 + x^2 + x^3 + x^4$ 136
4.3 Computation Steps for Long Division Using a Shift Register Circuit 141
4.4 Computing the Syndrome and Its Cyclic Shifts 148
4.5 Operation of the Meggitt Decoder, Input from the Left 151
4.6 Operation of the Meggitt Decoder, Input from the Right 152
4.7 CRC Generators 158
4.8 Lookup Table for CRC-ANSI 160
4.9 LFSR Example with $g(x) = 1 + x + x^2 + x^4$ and Initial State 1 163
4.10 LFSR Example with $g(x) = 1 + x + x^2 + x^4]$ and Initial State $1 + x$ 163
4.11 LFSR Example with $g(x) = 1 + x + x^2 + x^4]$ and Initial State $1 + x^2 + x^3$ 163
4.12 LFSR example with $g(x) = 1 + x + x^4]$ and Initial State 1 164
4.13 Barker Codes 178

5.1 Power, Vector, and Polynomial Representations of $GF(2^4)$ as an Extension Using $g(\alpha) = 1 + \alpha + \alpha^4$ 206
5.2 Conjugacy Classes over $GF(2^3)$ with Respect to $GF(2)$ 220
5.3 Conjugacy Classes over $GF(2^4)$ with Respect to $GF(2)$ 221
5.4 Conjugacy Classes over $GF(2^5)$ with Respect to $GF(2)$ 221
5.5 Conjugacy Classes over $GF(4^2)$ with Respect to $GF(4)$ 221
5.6 Cyclotomic Cosets Modulo 15 with Respect to $GF(7)$ 223

6.1 Weight Distribution of the Dual of a Double-Error-Correcting Primitive Binary BCH Code of Length $n = 2^m - 1$, $m \geq 3$, m Odd 246
6.2 Weight Distribution of the Dual of a Double-Error-Correcting Primitive Binary Narrow-Sense BCH Code, $n = 2^m - 1$, $m \geq 4$, m Even 247
6.3 Weight Distribution of the Dual of a Triple-Error- Correcting Primitive Binary Narrow-Sense BCH Code, $n = 2^m - 1$, $m \geq 5$, m Odd 247
6.4 Weight Distribution of the Dual of a Triple-Error- Correcting Primitive Binary Narrow-Sense BCH Code, $n = 2^m - 1$, $m \geq 6$, m Even 248
6.5 Evolution of the Berlekamp–Massey Algorithm for the Input Sequence $\{1, 1, 1, 0, 1, 0, 0\}$ 266
6.6 Berlekamp–Massey Algorithm for a Double-Error-Correcting Code 266
6.7 Berlekamp-Massey Algorithm for a Double-Error-Correcting Code: Simplifications for the Binary Code 267
6.8 Berlekamp–Massey Algorithm for a Triple-Error-Correcting Code 272

7.1 Monomials Ordered Under $(1, 3)$-revlex Order 330
7.2 Monomials Ordered Under $(1, 3)$-lex Order 330

8.1 Extended Quadratic Residue Codes Q 399
8.2 Weight Distributions for the $\mathcal{G}_{23}$ and $\mathcal{G}_{24}$ Codes 400

10.1 Performance Specification of the Compact Disc Coding System 430

12.1 Quantized Branch Metrics Using Linear Quantization 489
12.2 Best Convolutional Codes of Rates $R = 1/2$ and $R = 1/3$ 506
12.3 Best Convolutional Codes of Rates $R = 1/4$, $R = 2/3$, and $R = 3/4$ 507
12.4 Comparison of Free Distance as a Function of Constraint Length for Systematic and Nonsystematic Codes 508

12.5 Best Known $R = 2/3$ and $R = 3/4$ Convolutional Codes Obtained by Puncturing a $R = 1/2$ Code 510
12.6 Performance of Fano Algorithm on a Particular Sequence as a Function of Δ 523

13.1 Average Energy Requirements for Some QAM Constellations 547
13.2 Maximum Free-Distance Trellis Codes for 8-PSK Constellation 561
13.3 Maximum Free-Distance Trellis Codes for 16-PSK Constellation 562
13.4 Maximum Free-Distance Trellis Codes for Amplitude Modulated (One-Dimensional) Constellations 562
13.5 Encoder Connections and Coding Gains for Maximum Free-Distance QAM Trellis Codes 563
13.6 Attributes of Some Lattices 573
13.7 Comparison of Average Signal Energy for Circular Boundary $\mathbb{Z}^2$ and A_2 Constellations with Regular QAM 577
13.8 Some Good Multidimensional TCM Codes 578
13.9 Bit Converter: Sublattice Partition of 4D Rectangular Lattice [?] 580
13.10 4D Block Encoder [?] 584

14.1 α_t' and β_t' Example Computations 604
14.2 Posterior Input Bit Example Computations 604

15.1 Threshold Values for Various LDPC Codes for the Binary AWGN Channel 699
15.2 Degree Distribution Pairs for $R = 1/2$ Codes for Binary Transmission over an AWGN [? , p. 623]. σ_τ is the decoding threshold, also expressed as (E_b/N_0) (dB) 707

16.1 IS Simulation Results 757
16.2 MC and IS Comparison for Digital Communication for Various IS Mean Translation Schemes 763
16.3 IS Simulation Results for Single Parity-Check Codes of Length N. (See [493]) 768
16.4 IS Simulation Results for a $(3, 6, 3)$ Decoding Tree. $\epsilon = 10\%$ [493] 770

18.1 Coding Parameters for a Normal Frame $N_{\text{LDPC}} = 64{,}800$ (See [105, Table 6(a)] 888
18.2 Coding Parameters for a Short Frame $N_{\text{LDPC}} = 16{,}200$ (See [105, Table 6(b)] 888
18.3 BCH Minimal Polynomials for Normal Codes ($N_{\text{LDPC}} = 64{,}800$) (See [105, Table 7(a)]) 889
18.4 BCH Minimal Polynomials for Normal Codes ($N_{\text{LDPC}} = 6200$) (See [105, Table 7(b)]) . 889
18.5 IRA Indices for DVB-T2 Rate 2/3 Code, $N_{\text{LDPC}} = 64{,}800$ (See [105, Table A.3]) 890
18.6 Q_{LDPC} Values for Frames of Normal and Short Length 891
18.7 Channel Coding Schemes for Transmission and Control Channels [2, Section 5.1.3] 897

List of Boxes

Box 1.1 The Union Bound 22
Box 2.1 One-to-One and Onto Functions. 79
Box 3.1 Error Correction and Least Squares 100
Box 3.2 The UDP Protocol [SD1] 113
Box 4.1 The Division Algorithm 124
Box 5.1 Èveriste Galois (1811–1832) 204
Box 9.1 O and o Notation 409
Box 9.2 The Cauchy–Schwartz Inequality 412
Box 12.1 Graphs: Basic Definitions 459

About the Companion Website

This book is accompanied by a companion website:

www.wiley.com/go/Moon/ErrorCorrectionCoding

The website includes:

- Solutions Manual
- Program Files

Part I

Introduction and Foundations

Chapter 1

A Context for Error Correction Coding

I will make weak things become strong unto them ...

—Ether 12:27

... he denies that any error in the machine is responsible for the so-called errors in the answers. He claims that the Machines are self correcting and that it would violate the fundamental laws of nature for an error to exist in the circuits of relays.

—Isaac Asimov
I, Robot

1.1 Purpose of This Book

Error control coding in the context of digital communication has a history dating back to the middle of the twentieth century. In recent years, the field has been revolutionized by codes which are capable of approaching the theoretical limits of performance, the *channel capacity*. This has been impelled by a trend away from purely combinatoric and discrete approaches to coding theory toward codes which are more closely tied to a physical channel and soft decoding techniques. The purpose of this book is to present error correction and detection coding covering both traditional concepts thoroughly as well as modern developments in soft-decision and iteratively decoded codes and recent decoding algorithms for algebraic codes. An attempt has been made to maintain some degree of balance between the mathematics and their engineering implications by presenting both the mathematical methods used in understanding the codes as well as the algorithms which are used to efficiently encode and decode.

1.2 Introduction: Where Are Codes?

Error correction coding is the means whereby errors which may be introduced into digital data as a result of transmission through a communication channel can be corrected based upon received data. Error detection coding is the means whereby errors can be detected based upon received information. Collectively, error correction and error detection coding are *error control coding*. Error control coding can provide the difference between an operational communications system and a dysfunctional system. It has been a significant enabler in the telecommunications revolution, portable computers, the Internet, digital media, and space exploration. Error control coding is nearly ubiquitous in modern, information-based society. Every compact disc, CD-ROM, or DVD employs codes to protect the data embedded in the plastic disk. Every flash drive (thumb drive) and hard disk drive employs correction coding. Every phone call made over a digital cellular phone employs it. Every frame of digital television is protected by error correction coding. Every packet transmitted over the Internet has a protective coding "wrapper" used to determine if the packet has been received correctly. Even everyday commerce takes advantage of error detection coding, as the following examples illustrate.

Error Correction Coding: Mathematical Methods and Algorithms, Second Edition. Todd K. Moon
© 2021 John Wiley & Sons, Inc. Published 2021 by John Wiley & Sons, Inc.
Companion website: www.wiley.com/go/Moon/ErrorCorrectionCoding

Example 1.1 The ISBN (international standard book number) is used to uniquely identify books. An ISBN such as 0-201-36186-8 can be parsed as

$$\underbrace{0}_{\text{country}} - \underbrace{20}_{\text{publisher}} - \underbrace{1 - 36186}_{\text{book no.}} - \underbrace{8}_{\text{check}}.$$

Hyphens do not matter. The first digit indicates a country/language, with 0 for the United States. The next two digits are a publisher code. The next six digits are a publisher-assigned book number. The last digit is a check digit, used to validate if the code is correct using what is known as a weighted code. An ISBN is checked as follows: the cumulative sum of the digits is computed, then the cumulative sum of the cumulative sum is computed. For a valid ISBN, the sum-of-the-sum must be equal to 0, modulo 11. The character X is used for the check digit 10. For this ISBN, we have

	Cumulative	Cumulative
Digit	Sum	Sum
0	0	0
2	2	2
0	2	4
1	3	7
3	6	13
6	12	25
1	13	38
8	21	59
6	27	86
8	35	121

The final sum-of-the-sum is 121, which is equal to 0 modulo 11 (i.e., the remainder after dividing by 11 is 0). □

Example 1.2 The Universal Product Codes (UPC) employed on the bar codes of most merchandise employ a simple error detection system to help ensure reliability in scanning. In this case, the error detection system consists of a simple parity check. A UPC consists of a 12-digit sequence, which can be parsed as

$$\underbrace{0\ 16000}_{\substack{\text{manufacturer}\\ \text{identification}\\ \text{number}}} \quad \underbrace{66610}_{\substack{\text{item}\\ \text{number}}} \quad \underbrace{8}_{\substack{\text{parity}\\ \text{check}}}.$$

Denoting the digits as $u_1, u_2, \ldots, u_{12}$, the parity digit u_{12} is determined such that

$$3(u_1 + u_3 + u_5 + u_7 + u_9 + u_{11}) + (u_2 + u_4 + u_6 + u_8 + u_{10} + u_{12})$$

is a multiple of 10. In this case,

$$3(0 + 6 + 0 + 6 + 6 + 0) + (1 + 0 + 0 + 6 + 1 + 8) = 70.$$

If, when a product is scanned, the parity condition does not work, the operator is flagged so that the object may be re-scanned. □

1.3 The Communications System

Appreciation of the contributions of coding and understanding its limitations require some awareness of information theory and how its major theorems delimit the performance of a digital communication system. Information theory is increasingly relevant to coding theory, because with recent advances in theory it is now possible to achieve the performance bounds of information theory. By contrast, in the past, the bounds were more of a backdrop to the action on the stage of coding research and practice. Part of this success has come by placing the coding problem more fully in its communications context, marrying the coding problem more closely to the signal detection problem instead of treating the coding problem mostly as one of discrete combinatorics.

Information theory treats *information* almost as a physical quantity which can be measured, transformed, stored, and moved from place to place. A fundamental concept of information theory is that information is conveyed by the resolution of uncertainty. Information can be measured by the amount of uncertainty resolved. For example, if a digital source always emits the same value, say 1, then no information is gained by observing that the source has produced, yet again, the output 1, since there was no uncertainty about the outcome to begin with. Probabilities are used to mathematically describe the uncertainty. For a discrete random variable X (i.e., one which produces discrete outputs, such as $X = 0$ or $X = 1$), the information conveyed by observing an outcome x is defined as $-\log_2 P(X = x)$ bits. (If the logarithm is base 2, the units of information are in **bits**. If the natural logarithm is employed, the units of information are in **nats**.) For example, if $P(X = 1) = 1$ (the outcome 1 is certain), then observing $X = 1$ yields $-\log_2(1) = 0$ bits of information. On the other hand, observing $X = 0$ in this case yields $-\log_2(0) = \infty$: total surprise at observing an impossible outcome. The *entropy* is the *average information*. For a binary source X having two outcomes occurring with probabilities p and $1 - p$, the binary entropy function, denoted as either $H_2(X)$ (indicating that it is the entropy of the source) or $H_2(p)$ (indicating that it is a function of the outcome probabilities) is

$$H_2(X) = H_2(p) = E[-\log_2 P(X)] = -p\log_2(p) - (1-p)\log_2(1-p) \text{ bits.}$$

A plot of the binary entropy function as a function of p is shown in Figure 1.1. The peak information of 1 bit occurs when $p = \frac{1}{2}$.

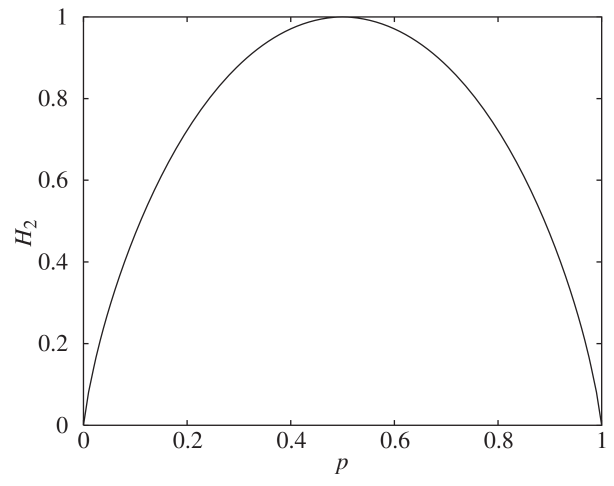


Figure 1.1: The binary entropy function $H_2(p)$.

Example 1.3 A fair coin is tossed once per second, with the outcomes being "head" and "tail" with equal probability. Each toss of the coin generates an event that can be described with $H_2(0.5) = 1$ bit of information. The sequence of tosses produces information at a rate of 1 bit/second. An unfair coin, with $P(\text{head}) = 0.01$ is tossed. The average information generated by each throw in this case is $H_2(0.01) = 0.0808$ bits. Another unfair coin, with $P(\text{head}) = 1$ is tossed. The information generated by each throw in this case is $H_2(1) = 0$ bits. □

For a source X having M outcomes $x_1, x_2, \ldots, x_M$, with probabilities $P(X = x_i) = p_i$, $i = 1, 2, \ldots, M$, the entropy is

$$H(X) = E[-\log_2 P(X)] = -\sum_{i=1}^{M} p_i \log_2 p_i \text{ bits.} \tag{1.1}$$

Note: The "bit" as a measure of entropy (or information content) is different from the "bit" as a measure of storage. For the unfair coin having $P(\text{head}) = 1$, the actual information content determined by a toss of the coin is 0: there is no information gained by observing that the outcome is again 1. For this process with this unfair coin, the entropy rate — that is, the amount of actual information it generates — is 0. However, if the coin outcomes were for some reason to be stored directly, without the benefit of some kind of coding, each outcome would require 1 bit of storage (even though they don't represent any new information).

With the prevalence of computers in our society, we are accustomed to thinking in terms of "bits" — e.g., a file is so many bits long, the register of a computer is so many bits wide. But these are "bits" as a measure of storage size, not "bits" as a measure of actual information content. Because of the confusion between "bit" as a unit of information content and "bit" as an amount of storage, the unit of information content is sometimes — albeit rarely — called a *Shannon*, in homage to the founder of information theory, Claude Shannon.[1]

A digital communication system embodies functionality to perform physical actions on information. Figure 1.2 illustrates a fairly general framework for a single digital communication link. In this link, digital data from a *source* are encoded and modulated (and possibly encrypted) for communication over a *channel*. At the other end of the channel, the data are demodulated, decoded (and possibly decrypted), and sent to a sink. The elements in this link all have mathematical descriptions and theorems from information theory which govern their performance. The diagram indicates the realm of applicability of three major theorems of information theory.

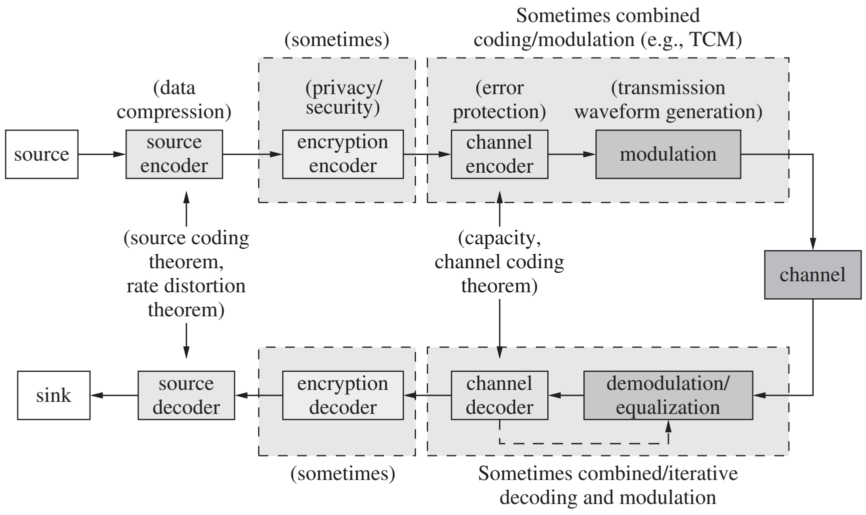


Figure 1.2: A general framework for digital communications.

There are actually many kinds of codes employed in a communication system. In the description below, we point out where some of these codes arise. Throughout the book we make some connections between these codes and our major focus of study, error correction codes.

The source is the data to be communicated, such as a computer file, a video sequence, or a telephone conversation. For our purposes, it is represented in digital form, perhaps as a result of an

[1] This mismatch of object and value is analogous to the physical horse, which may or may not be capable of producing one "horsepower" of power, 550 ft-lbs/second. Thermodynamicists can dodge the issue by using the SI unit of Watts for power, information theorists might sidestep confusion by using the Shannon. Both of these units honor founders of their respective disciplines.

analog-to-digital conversion step. Information-theoretically, sources are viewed as streams of random numbers governed by some probability distribution.

Every source of data has a measure of the information that it represents, which (in principle) can be exactly quantified in terms of entropy.

The source encoder performs data compression by removing redundancy.

As illustrated in Example 1.3, the number of bits used to store the information from a source may exceed the number of bits of actual information content. That is, the number of bits to represent the data may exceed the number of mathematical bits — Shannons — of actual information content.

The amount a particular source of data can be compressed without any loss of information (*lossless* compression) is governed theoretically by the *source coding theorem* of information theory, which states that a source of information can be represented without any loss of information in such a way that the amount of storage required (in bits) is equal to the amount of information content — the entropy — in bits or Shannons. To achieve this lower bound, it may be necessary for long blocks of the data to be jointly encoded.

Example 1.4 For the unfair coin with $P(\text{head}) = 0.01$, the entropy is $H(0.01) = 0.0808$. Therefore, 10,000 such (independent) tosses convey 808 bits (Shannons) of information, so theoretically the information of 10,000 tosses of the coin can be represented exactly using only 808 (physical) bits of information. □

Thus a bit (in a computer register) in principle *can* represents an actual (mathematical) bit of information content, if the source of information is represented correctly.

In compressing a data stream, a source encoder removes redundancy present in the data. For compressed binary data, 0 and 1 occur with equal probability in the compressed data (otherwise, there would be some redundancy which could be exploited to further compress the data). Thus, it is frequently assumed at the channel coder that 0 and 1 occur with equal probability.

The source encoder employs special types of codes to do the data compression, called collectively source codes or data compression codes. Such coding techniques include Huffman coding, run-length coding, arithmetic coding, Lempel–Ziv coding, and combinations of these, all of which fall beyond the scope of this book.

If the data need to be compressed below the entropy rate of the source, then some kind of distortion must occur. This is called lossy data compression. In this case, another theorem governs the representation of the data. It is possible to do lossy compression in a way that minimizes the amount of distortion for a given rate of transmission. The theoretical limits of lossy data compression are established by the *rate–distortion theorem* of information theory. One interesting result of rate–distortion theory says that for a binary source having equiprobable outcomes, the minimal rate to which the data can be compressed with the average distortion per bit equal to p is

$$r = 1 - H_2(p) \quad p \leq \frac{1}{2}. \tag{1.2}$$

Lossy data compression uses its own kinds of codes as well.

Example 1.5 In the previous example, suppose that a channel is available that only provides 800 bits of information to convey 10,000 tosses of the biased coin. Since this number of bits is less than allowed by the source coding theorem, there must be some distortion introduced. From above, the source rate is 0.0808 bits of information per toss. The rate we are sending over this restricted channel is $800/10\ 000 = 0.08$ bits/toss. The rate at which tosses are coded is

$$r = \frac{800}{808} = 0.9901.$$

The distortion is

$$p = H_2^{-1}(1 - r) = H_2^{-1}(1 - 0.9901) = H_2^{-1}(0.0099) = 0.00085.$$

That is, out of 10,000 coin tosses, transmission at this rate would introduce distortion in (0.00085) (10,000) = 8.5 bits.

□

The encrypter hides or scrambles information so that unintended listeners are unable to discern the information content. The codes used for encryption are generally different from the codes used for error correction.

Encryption is often what the layperson frequently thinks of when they think of "codes," as in secret codes, but as we are seeing, there are many other different kinds of codes.

As we will see, however, the mathematical tools used in error correction coding can be applied to some encryption codes. In particular, we will meet RSA public key encryption as an outgrowth of number theoretic tools to be developed, and McEliece public key encryption as a direct application of a particular family of error correction codes.

The channel coder is the first step in the error correction or error detection process.

The channel coder adds redundant information to the stream of input symbols in a way that allows errors which are introduced into the channel to be corrected. **This book is primarily dedicated to the study of the channel coder and its corresponding channel decoder.**

It may seem peculiar to remove redundancy with the source encoder, then turn right around and add redundancy back in with the channel encoder. However, the redundancy in the source typically depends on the source in an unstructured way and may not provide uniform protection to all the information in the stream, nor provide any indication of how errors occurred or how to correct them. The redundancy provided by the channel coder, on the other hand, is introduced in a structured way, precisely to provide error control capability.

Treating the problems of data compression and error correction separately, rather than seeking a jointly optimal source/channel coding solution, is asymptotically optimal (as the block sizes get large). This fact is called the *source–channel separation theorem* of information theory. (There has been work on combined source/channel coding for finite — practical — block lengths, in which the asymptotic theorems are not invoked. This work falls outside the scope of this book.)

Because of the redundancy introduced by the channel coder, there must be more symbols at the output of the coder than at the input. Frequently, a channel coder operates by accepting a block of k input symbols and producing at its output a block of n symbols, with $n > k$. The **rate** of such a channel coder is

$$R = k/n,$$

so that $R < 1$.

The input to the channel coder is referred to as the *message symbols* (or, in the case of binary codes, the *message bits*). The input may also be referred to as the *information symbols* (or bits).

The modulator converts symbol sequences from the channel encoders into signals appropriate for transmission over the channel. Many channels require that the signals be sent as a continuous-time voltage, or an electromagnetic waveform in a specified frequency band. The modulator provides the appropriate channel-conforming representation.

Included within the modulator block one may find codes as well. Some channels (such as magnetic recording channels) have constraints on the maximum permissible length of runs of 1s. Or they might have a restriction that the sequence must be DC-free. Enforcing such constraints employs special codes. Treatment of such runlength-limited codes appears in [275]; see also [209].

Some modulators employ mechanisms to ensure that the signal occupies a broad bandwidth. This *spread-spectrum* modulation can serve to provide multiple-user access, greater resilience to jamming, low probability of detection, and other advantages. (See, e.g., [509].) Spread-spectrum systems frequently make use of pseudorandom sequences, some of which are produced using linear feedback shift registers as discussed in Appendix 4.A.

The channel is the medium over which the information is conveyed. Examples of channels are telephone lines, internet cables, fiber-optic lines, microwave radio channels, high-frequency channels, cell phone channels, etc. These are channels in which information is conveyed between two distinct places. Information may also be conveyed between two distinct times, for example, by writing information onto a computer disk, then retrieving it at a later time. Hard drives, CD-ROMs, DVDs, and solid-state memory are other examples of channels.

As signals travel through a channel, they are corrupted. For example, a signal may have noise added to it; it may experience time delay or timing jitter, or suffer from attenuation due to propagation distance and/or carrier offset; it may be reflected by multiple objects in its path, resulting in constructive and/or destructive interference patterns; it may experience inadvertent interference from other channels, or be deliberately jammed. It may be filtered by the channel response, resulting in interference among symbols. These sources of corruption in many cases can all occur simultaneously.

For purposes of analysis, channels are frequently characterized by mathematical models, which (it is hoped) are sufficiently accurate to be representative of the attributes of the actual channel, yet are also sufficiently abstracted to yield tractable mathematics. Most of our work in this book will assume one of two idealized channel models, the binary symmetric channel (BSC) and the additive white Gaussian noise channel (AWGNC), which are described in Section 1.5. While these idealized models do not represent all of the possible problems a signal may experience, they form a starting point for many, if not most, of the more comprehensive channel models. The experience gained by studying these simpler channel models forms a foundation for more accurate and complicated channel models. (As exceptions to the AWGN or BSC rule, in Section 14.7, we comment briefly on convolutive channels and turbo equalization, while in Chapter 19, coding for quasi-static Rayleigh flat fading channels are discussed.)

As suggested by Figure 1.2, the channel encoding and modulation may be combined in what is known as *coded modulation*.

Channels have different information-carrying capabilities. For example, a dedicated fiber-optic line is capable of carrying more information than a plain-old-telephone-service (POTS) pair of copper wires. Associated with each channel is a quantity known as the **capacity**, C, which indicates how much information it can carry **reliably**.

The information a channel can reliably carry is intimately related to the use of error correction coding. The governing theorem from information theory is Shannon's **channel coding theorem**, which states essentially this: Provided that the rate R of transmission is less than the capacity C, *there exists* a code such that the probability of error can be made arbitrarily small.

The demodulator/equalizer receives the signal from the channel and converts it into a sequence of symbols. This typically involves many functions, such as filtering, demodulation, carrier synchronization, symbol timing estimation, frame synchronization, and matched filtering, followed