TLFeBOOK

**AN INTRODUCTION TO NUMERICAL ANALYSIS FOR ELECTRICAL AND COMPUTER ENGINEERS**

**Christopher J. Zarowski** *University of Alberta, Canada*

**A JOHN WILEY & SONS, INC. PUBLICATION**

TLFeBOOK

TLFeBOOK

**AN INTRODUCTION TO NUMERICAL ANALYSIS FOR ELECTRICAL AND COMPUTER ENGINEERS**

TLFeBOOK

TLFeBOOK

**AN INTRODUCTION TO NUMERICAL ANALYSIS FOR ELECTRICAL AND COMPUTER ENGINEERS**

**Christopher J. Zarowski** *University of Alberta, Canada*

**A JOHN WILEY & SONS, INC. PUBLICATION**

TLFeBOOK

Copyright c 2004 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey. Published simultaneously in Canada.

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, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400, fax 978-646-8600, or on the Web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services, please contact our Customer Care Department within the United States at 877-762-2974, outside the United States at 317-572-3993 or fax 317-572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print, however, may not be available in electronic format.

***Library of Congress Cataloging-in-Publication Data:***

Zarowski, Christopher J.

An introduction to numerical analysis for electrical and computer engineers / Christopher J. Zarowski.

p. cm. Includes bibliographical references and index. ISBN 0-471-46737-5 (coth)

1. Electric engineering—Mathematics. 2. Computer science—Mathematics. 3. Numerical analysis I. Title.

TK153.Z37 2004 621.3 01 518—dc22

2003063761

Printed in the United States of America.

10987654321

TLFeBOOK

*In memory of my mother Lilian and of my father Walter*

TLFeBOOK

TLFeBOOK

**CONTENTS**

**Preface xiii**

**1 Functional Analysis Ideas 1**

1.1 Introduction 1 1.2 Some Sets 2 1.3 Some Special Mappings: Metrics, Norms, and Inner Products 4 1.3.1 Metrics and Metric Spaces 6 1.3.2 Norms and Normed Spaces 8 1.3.3 Inner Products and Inner Product Spaces 14 1.4 The Discrete Fourier Series (DFS) 25 Appendix 1.A Complex Arithmetic 28 Appendix 1.B Elementary Logic 31 References 32 Problems 33

**2 Number Representations 38**

2.1 Introduction 38 2.2 Fixed-Point Representations 38 2.3 Floating-Point Representations 42 2.4 Rounding Effects in Dot Product Computation 48 2.5 Machine Epsilon 53 Appendix 2.A Review of Binary Number Codes 54 References 59 Problems 59

**3 Sequences and Series 63**

3.1 Introduction 63 3.2 Cauchy Sequences and Complete Spaces 63 3.3 Pointwise Convergence and Uniform Convergence 70 3.4 Fourier Series 73 3.5 Taylor Series 78**vii**TLFeBOOK

**viii** CONTENTS

3.6 Asymptotic Series 97 3.7 More on the Dirichlet Kernel 103 3.8 Final Remarks 107 Appendix 3.A *CO*ordinate *R*otation *DI*gital *C*omputing

(CORDIC) 107 3.A.1 Introduction 107 3.A.2 The Concept of a Discrete Basis 108 3.A.3 Rotating Vectors in the Plane 112 3.A.4 Computing Arctangents 114 3.A.5 Final Remarks 115 Appendix 3.B Mathematical Induction 116 Appendix 3.C Catastrophic Cancellation 117 References 119 Problems 120

**4 Linear Systems of Equations 127**

4.1 Introduction 127 4.2 Least-Squares Approximation and Linear Systems 127 4.3 Least-Squares Approximation and Ill-Conditioned Linear

Systems 132 4.4 Condition Numbers 135 4.5 *LU* Decomposition 148 4.6 Least-Squares Problems and *QR* Decomposition 161 4.7 Iterative Methods for Linear Systems 176 4.8 Final Remarks 186 Appendix 4.A Hilbert Matrix Inverses 186 Appendix 4.B SVD and Least Squares 191 References 193 Problems 194

**5 Orthogonal Polynomials 207**

5.1 Introduction 207 5.2 General Properties of Orthogonal Polynomials 207 5.3 Chebyshev Polynomials 218 5.4 Hermite Polynomials 225 5.5 Legendre Polynomials 229 5.6 An Example of Orthogonal Polynomial Least-Squares

Approximation 235 5.7 Uniform Approximation 238TLFeBOOK

CONTENTS **ix**

References 241 Problems 241

**6 Interpolation 251**

6.1 Introduction 251 6.2 Lagrange Interpolation 252 6.3 Newton Interpolation 257 6.4 Hermite Interpolation 266 6.5 Spline Interpolation 269 References 284 Problems 285

**7 Nonlinear Systems of Equations 290**

7.1 Introduction 290 7.2 Bisection Method 292 7.3 Fixed-Point Method 296 7.4 Newton–Raphson Method 305 7.4.1 The Method 305 7.4.2 Rate of Convergence Analysis 309 7.4.3 Breakdown Phenomena 311 7.5 Systems of Nonlinear Equations 312 7.5.1 Fixed-Point Method 312 7.5.2 Newton–Raphson Method 318 7.6 Chaotic Phenomena and a Cryptography Application 323 References 332 Problems 333

**8 Unconstrained Optimization 341**

8.1 Introduction 341 8.2 Problem Statement and Preliminaries 341 8.3 Line Searches 345 8.4 Newton’s Method 353 8.5 Equality Constraints and Lagrange Multipliers 357 Appendix 8.A MATLAB Code for Golden Section Search 362 References 364 Problems 364

**9 Numerical Integration and Differentiation 369**

9.1 Introduction 369TLFeBOOK

**x** CONTENTS

9.2 Trapezoidal Rule 371 9.3 Simpson’s Rule 378 9.4 Gaussian Quadrature 385 9.5 Romberg Integration 393 9.6 Numerical Differentiation 401 References 406 Problems 406

**10 Numerical Solution of Ordinary Differential Equations 415**

10.1 Introduction 415 10.2 First-Order ODEs 421 10.3 Systems of First-Order ODEs 442 10.4 Multistep Methods for ODEs 455 10.4.1 Adams–Bashforth Methods 459 10.4.2 Adams–Moulton Methods 461 10.4.3 Comments on the Adams Families 462 10.5 Variable-Step-Size (Adaptive) Methods for ODEs 464 10.6 Stiff Systems 467 10.7 Final Remarks 469 Appendix 10.A MATLAB Code for Example 10.8 469 Appendix 10.B MATLAB Code for Example 10.13 470 References 472 Problems 473

**11 Numerical Methods for Eigenproblems 480**

11.1 Introduction 480 11.2 Review of Eigenvalues and Eigenvectors 480 11.3 The Matrix Exponential 488 11.4 The Power Methods 498 11.5 *QR* Iterations 508 References 518 Problems 519

**12 Numerical Solution of Partial Differential Equations 525**

12.1 Introduction 525 12.2 A Brief Overview of Partial Differential Equations 525 12.3 Applications of Hyperbolic PDEs 528 12.3.1 The Vibrating String 528 12.3.2 Plane Electromagnetic Waves 534TLFeBOOK

CONTENTS **xi**

12.4 The Finite-Difference (FD) Method 545 12.5 The Finite-Difference Time-Domain (FDTD) Method 550 Appendix 12.A MATLAB Code for Example 12.5 557 References 560 Problems 561

**13 An Introduction to MATLAB 565**

13.1 Introduction 565 13.2 Startup 565 13.3 Some Basic Operators, Operations, and Functions 566 13.4 Working with Polynomials 571 13.5 Loops 572 13.6 Plotting and M-Files 573 References 577

**Index 579**TLFeBOOK

TLFeBOOK

**PREFACE**

The subject of numerical analysis has a long history. In fact, it predates by cen- turies the existence of the modern computer. Of course, the advent of the modern computer in the middle of the twentieth century gave greatly added impetus to the subject, and so it now plays a central role in a large part of engineering analysis, simulation, and design. This is so true that no engineer can be deemed competent without some knowledge and understanding of the subject. Because of the back- ground of the author, this book tends to emphasize issues of particular interest to electrical and computer engineers, but the subject (and the present book) is certainly relevant to engineers from all other branches of engineering.

Given the importance level of the subject, a great number of books have already been written about it, and are now being written. These books span a colossal range of approaches, levels of technical difficulty, degree of specialization, breadth versus depth, and so on. So, why should this book be added to the already huge, and growing list of available books?

To begin, the present book is intended to be a part of the students’ first exposure to numerical analysis. As such, it is intended for use mainly in the second year of a typical 4-year undergraduate engineering program. However, the book may find use in later years of such a program. Generally, the present book arises out of the author’s objections to educational practice regarding numerical analysis. To be more specific

1. Some books adopt a “grocery list” or “recipes” approach (i.e., “methods” at the expense of “analysis”) wherein several methods are presented, but with little serious discussion of issues such as how they are obtained and their relative advantages and disadvantages. In this genre often little consideration is given to error analysis, convergence properties, or stability issues. When these issues are considered, it is sometimes in a manner that is too superficial for contemporary and future needs. 2. Some books fail to build on what the student is supposed to have learned prior to taking a numerical analysis course. For example, it is common for engineering students to take a first-year course in matrix/linear algebra. Yet, a number of books miss the opportunity to build on this material in a manner that would provide a good bridge from first year to more sophisticated uses of matrix/linear algebra in later years (e.g., such as would be found in digital signal processing or state variable control systems courses).

**xiii**TLFeBOOK

**xiv** PREFACE

3. Some books miss the opportunity to introduce students to the now quite vital area of functional analysis ideas as applied to engineering problem solving. Modern numerical analysis relies heavily on concepts such as function spaces, orthogonality, norms, metrics, and inner products. Yet these concepts are often considered in a very ad hoc way, if indeed they are considered at all. 4. Some books tie the subject matter of numerical analysis far too closely to particular software tools and/or programming languages. But the highly tran- sient nature of software tools and programming languages often blinds the user to the timeless nature of the underlying principles of analysis. Further- more, it is an erroneous belief that one can successfully employ numerical methods solely through the use of “canned” software without any knowledge or understanding of the technical details of the contents of the can. While this does not imply the need to understand a software tool or program down to the last line of code, it does rule out the “black box” methodology. 5. Some books avoid detailed analysis and derivations in the misguided belief that this will make the subject more accessible to the student. But this denies the student the opportunity to learn an important mode of thinking that is a huge aid to practical problem solving. Furthermore, by cutting the student off from the language associated with analysis the student is prevented from learning those skills needed to read modern engineering literature, and to extract from this literature those things that are useful for solving the problem at hand.

The prospective user of the present book will likely notice that it contains material that, in the past, was associated mainly with more advanced courses. However, the history of numerical computing since the early 1980s or so has made its inclusion in an introductory course unavoidable. There is nothing remarkable about this. For example, the material of typical undergraduate signals and systems courses was, not so long ago, considered to be suitable only for graduate-level courses. Indeed, most (if not all) of the contents of any undergraduate program consists of material that was once considered far too advanced for undergraduates, provided one goes back far enough in time.

Therefore, with respect to the observations mentioned above, the following is a summary of some of the features of the present book:

1. An axiomatic approach to function spaces is adopted within the first chapter. So the book immediately exposes the student to function space ideas, espe- cially with respect to metrics, norms, inner products, and the concept of orthogonality in a general setting. All of this is illustrated by several examples, and the basic ideas from the first chapter are reinforced by routine use throughout the remaining chapters. 2. The present book is not closely tied to any particular software tool or pro- gramming language, although a few MATLAB-oriented examples are pre- sented. These may be understood without any understanding of MATLABTLFeBOOK

PREFACE **xv**

(derived from the term *matrix laboratory*) on the part of the student, how- ever. Additionally, a quick introduction to MATLAB is provided in Chapter 13. These examples are simply intended to illustrate that modern software tools implement many of the theories presented in the book, and that the numerical characteristics of algorithms implemented with such tools are not materially different from algorithm implementations using older software technologies (e.g., catastrophic convergence, and ill conditioning, continue to be major implementation issues). Algorithms are often presented in a Pascal-like pseudocode that is sufficiently transparent and general to allow the user to implement the algorithm in the language of their choice. 3. Detailed proofs and/or derivations are often provided for many key results. However, not all theorems or algorithms are proved or derived in detail on those occasions where to do so would consume too much space, or not provide much insight. Of course, the reader may dispute the present author’s choices in this matter. But when a proof or derivation is omitted, a reference is often cited where the details may be found. 4. Some modern applications examples are provided to illustrate the conse- quences of various mathematical ideas. For example, chaotic cryptography, the CORDIC (*co*ordinate *r*otational *d*igital *c*omputing) method, and least squares for system identification (in a biomedical application) are considered. 5. The sense in which series and iterative processes converge is given fairly detailed treatment in this book as an understanding of these matters is now so crucial in making good choices about which algorithm to use in an appli- cation. Thus, for example, the difference between pointwise and uniform convergence is considered. Kernel functions are introduced because of their importance in error analysis for approximations based on orthogonal series. Convergence rate analysis is also presented in the context of root-finding algorithms. 6. Matrix analysis is considered in sufficient depth and breadth to provide an adequate introduction to those aspects of the subject particularly relevant to modern areas in which it is applied. This would include (but not be limited to) numerical methods for electromagnetics, stability of dynamic systems, state variable control systems, digital signal processing, and digital commu- nications. 7. The most important general properties of orthogonal polynomials are pre- sented. The special cases of Chebyshev, Legendre, and Hermite polynomials are considered in detail (i.e., detailed derivations of many basic properties are given). 8. In treating the subject of the numerical solution of ordinary differential equations, a few books fail to give adequate examples based on nonlin- ear dynamic systems. But many examples in the present book are based on nonlinear problems (e.g., the Duffing equation). Furthermore, matrix methods are introduced in the stability analysis of both explicit and implicit methods for *n*th-order systems. This is illustrated with second-order examples.

TLFeBOOK

**xvi** PREFACE

Analysis is often embedded in the main body of the text rather than being rele- gated to appendixes, or to formalized statements of proof immediately following a theorem statement. This is done to discourage attempts by the reader to “skip over the math.” After all, skipping over the math defeats the purpose of the book.

Notwithstanding the remarks above, the present book lacks the rigor of a math- ematically formal treatment of numerical analysis. For example, Lebesgue measure theory is entirely avoided (although it is mentioned in passing). With respect to functional analysis, previous authors (e.g., E. Kreyszig, *Introductory Functional Analysis with Applications*) have demonstrated that it is very possible to do this while maintaining adequate rigor for engineering purposes, and this approach is followed here.

It is largely left to the judgment of the course instructor about what particular portions of the book to cover in a course. Certainly there is more material here than can be covered in a single term (or semester). However, it is recommended that the first four chapters be covered largely in their entirety (perhaps excepting Sections 1.4, 3.6, 3.7, and the part of Section 4.6 regarding SVD). The material of these chapters is simply too fundamental to be omitted, and is often drawn on in later chapters.

Finally, some will say that topics such as function spaces, norms and inner products, and uniform versus pointwise convergence, are too abstract for engineers. Such individuals would do well to ask themselves in what way these ideas are more abstract than Boolean algebra, convolution integrals, and Fourier or Laplace transforms, all of which are standard fare in present-day electrical and computer engineering curricula.

Engineering past Engineering present Engineering future

*Christopher Zarowski*TLFeBOOK

**1 Functional Analysis Ideas**

**1.1 INTRODUCTION**

Many engineering analysis and design problems are far too complex to be solved without the aid of computers. However, the use of computers in problem solving has made it increasingly necessary for users to be highly skilled in (practical) mathematical analysis. There are a number of reasons for this. A few are as follows. such For as one *π* or thing, √

2 do computers not have an represent exact representation data to finite on precision. a digital computer Irrational (with numbers the possible exception of methods based on symbolic computing). Additionally, when arithmetic is performed, errors occur as a result of rounding (e.g., the truncation of the product of two *n*-bit numbers, which might be 2*n* bits long, back down to *n* bits). Numbers have a limited dynamic range; we might get overflow or underflow in a computation. These are examples of *finite-precision arithmetic effects.* Beyond this, computational methods frequently have sources of error independent of these. For example, an infinite series must be truncated if it is to be evaluated on a com- puter. The truncation error is something “additional” to errors from finite-precision arithmetic effects. In all cases, the sources (and sizes) of error in a computation must be known and understood in order to make sensible claims about the accuracy of a computer-generated solution to a problem.

Many methods are “iterative.” Accuracy of the result depends on how many iterations are performed. It is possible that a given method might be very slow, requiring many iterations before achieving acceptable accuracy. This could involve much computer runtime. The obvious solution of using a faster computer is usually unacceptable. A better approach is to use mathematical analysis to understand why a method is slow, and so to devise methods of speeding it up. Thus, an important feature of analysis applied to computational methods is that of assessing how much in the way of computing resources is needed by a given method. A given computational method will make demands on computer memory, operations count (the number of arithmetic operations, function evaluations, data transfers, etc.), number of bits in a computer word, and so on.

A given problem almost always has many possible alternative solutions. Other than accuracy and computer resource issues, ease of implementation is also rel- evant. This is a human labor issue. Some methods may be easier to implement on a given set of computing resources than others. This would have an impact

*An Introduction to Numerical Analysis for Electrical and Computer Engineers*, by C.J. Zarowski ISBN 0-471-46737-5 c 2004 John Wiley & Sons, Inc.

**1**TLFeBOOK

**2** FUNCTIONAL ANALYSIS IDEAS

on software/hardware development time, and hence on system cost. Again, math- ematical analysis is useful in deciding on the relative ease of implementation of competing solution methods.

The subject of numerical computing is truly vast. Methods are required to handle an immense range of problems, such as solution of differential equations (ordi- nary or partial), integration, solution of equations and systems of equations (linear or nonlinear), approximation of functions, and optimization. These problem types appear to be radically different from each other. In some sense the differences between them are true, but there are means to achieve some unity of approach in understanding them.

The branch of mathematics that (perhaps) gives the greatest amount of unity is sometimes called *functional analysis*. We shall employ ideas from this subject throughout. However, our usage of these ideas is not truly rigorous; for example, we completely avoid topology, and measure theory. Therefore, we tend to follow simplified treatments of the subject such as Kreyszig [1], and then only those ideas that are immediately relevant to us. The reader is assumed to be very comfortable with elementary linear algebra, and calculus. The reader must also be comfortable with complex number arithmetic (see Appendix 1.A *now* for a review if necessary). Some knowledge of electric circuit analysis is presumed since this will provide a source of applications examples later. (But application examples will also be drawn from other sources.) Some knowledge of ordinary differential equations is also assumed.

It is worth noting that an understanding of functional analysis is a tremendous aid to understanding other subjects such as quantum physics, probability theory and random processes, digital communications system analysis and design, digital control systems analysis and design, digital signal processing, fuzzy systems, neural networks, computer hardware design, and optimal design of systems. Many of the ideas presented in this book are also intended to support these subjects.

**1.2 SOME SETS**

Variables in an engineering problem often take on values from sets of numbers. In the present setting, the sets of greatest interest to us are (1) the *set of integers* **Z** *complex* = {*...*−3*,*−2*,*−1*,*0*,*1*,*2*,*3*...*}, *numbers* **C** = {*x* + *jy*|*j* = (2) √

−1*,x,y* the set of ∈ *real* **R**}. *numbers* The set of **R**, nonnegative and (3) the *set* inte- *of*

gers is **Z**+ = {0*,*1*,*2*,*3*,...,*} (so **Z**+ ⊂ **Z**). Similarly, the set of nonnegative real numbers is **R**+ = {*x* ∈ **R**|*x* ≥ 0}. Other kinds of sets of numbers will be introduced if and when they are needed.

If *A* and *B* are two sets, their *Cartesian product* is denoted by *A* × *B* = {*(a, b)*|*a* ∈ *A, b* ∈ *B*}. The Cartesian product of *n* sets denoted *A*0*,A*1*,...,An*−1 is *A*0 × *A*1 ×···× *An*−1 = {*(a*0*,a*1*,...,an*−1*)*|*ak* ∈ *Ak*}. Ideas from matrix/linear algebra are of great importance. We are therefore also interested in sets of vectors. Thus, **R***n* shall denote the set of *n*-element vectors with real-valued components, and similarly, **C***n* shall denote the set of *n*-elementTLFeBOOK

SOME SETS **3**

vectors with complex-valued components. By default, we assume any vector *x* to be a column vector:

*x* =



*x*0*x...* 1*xn*−2 *. (*1*.*1*) xn*−1

Naturally, row vectors are obtained by transposition. We will generally avoid using bars over or under symbols to denote vectors. Whether a quantity is a vector will be clear from the context of the discussion. However, bars will be used to denote vectors when this cannot be easily avoided. The indexing of vector elements *xk* will often begin with 0 as indicated in (1.1). Naturally, matrices are also important. Set **R***n*×*m* denotes the set of matrices with *n* rows and *m* columns, and the elements are real-valued. The notation **C***n*×*m* should now possess an obvious meaning. Matri- ces will be denoted by uppercase symbols, again without bars. If *A* is an *n* × *m* matrix, then

*A* = [*ap,q*]*p*=0*,...,n*−1*, q*=0*,...,m*−1*. (*1*.*2*)*

Thus, the element in row *p* and column *q* of *A* is denoted *ap,q*. Indexing of rows and columns again will typically begin at 0. The subscripts on the right bracket “]” in (1.2) will often be omitted in the future. We may also write *apq* instead of *ap,q* where no danger of confusion arises. The elements of any vector may be regarded as the elements of a sequence of finite length. However, we are also very interested in sequences of infinite length. An *infinite sequence* may be denoted by *x* = *(xk)* = *(x*0*,x*1*,x*2*,...)*, for which *xk* could be either real-valued or complex-valued. It is possible for sequences to be *doubly infinite*, for instance, *x* = *(xk)* = *(...,x*−2*,x*−1*,x*0*,x*1*,x*2*,...)*.

Relationships between variables are expressed as mathematical functions, that is, *mappings* between sets. The notation *f* |*A* → *B* signifies that function *f* associates an element of set *A* with an element from set *B*. For example, *f*|**R** → **R** represents a function defined on the real-number line, and this function is also real-valued; that is, it maps “points” in **R** to “points” in **R**. We are familiar with the idea of “plotting” such a function on the *xy* plane if *y* = *f (x)* (i.e., *x,y* ∈ **R**). It is important to note that we may regard sequences as functions that are defined on either the set **Z** (the case of doubly infinite sequences), or the set **Z**+ (the case of singly infinite sequences). To be more specific, if, for example, *k* ∈ **Z**+, then this number maps to some number *xk* that is either real-valued or complex-valued. Since vectors are associated with sequences of finite length, they, too, may be regarded as functions, but defined on a finite subset of the integers. From (1.1) this subset might be denoted by **Z***n* = {0*,*1*,*2*,...,n* − 2*,n* − 1}.

Sets of functions are important. This is because in engineering we are often interested in mappings between sets of functions. For example, in electric circuits voltage and current waveforms (i.e., functions of time) are input to a circuit via volt- age and current sources. Voltage drops across circuit elements, or currents throughTLFeBOOK

**4** FUNCTIONAL ANALYSIS IDEAS

circuit elements are output functions of time. Thus, any circuit maps functions from an input set to functions from some output set. Digital signal processing systems do the same thing, except that here the functions are sequences. For example, a simple digital signal processing system might accept as input the sequence *(xn)*, and produce as output the sequence *(yn)* according to

*yn* = *xn* + 2 *xn*+1

*(*1*.*3*)*

for which *n* ∈ **Z**+.

Some specific examples of sets of functions are as follows, and more will be seen later. The set of real-valued functions defined on the interval [*a,b*] ⊂ **R** that are *n* times *continuously* differentiable may be denoted by *Cn*[*a,b*]. This means that all derivatives up to and including order *n* exist and are continuous. If *n* = 0 we often just write *C*[*a,b*], which is the set of continuous functions on the interval [*a,b*]. We remark that the notation [*a,b*] implies inclusion of the endpoints of the interval. Thus, *(a, b)* implies that the endpoints *a* and *b* are not to be included [i.e., if *x* ∈ *(a, b)*, then *a<x<b*].

A polynomial in the *indeterminate x* of degree *n* is

*pn(x)* =

∑*nk*=0

*pn,kxk. (*1*.*4*)*

*pn,kxk. (*1*.*4*)*

Unless otherwise stated, we will always assume *pn,k* ∈ **R**. The indeterminate *x* is often considered to be either a real number or a complex number. But in some circumstances the indeterminate *x* is merely regarded as a “placeholder,” which means that *x* is not supposed to take on a value. In a situation like this the polynomial coefficients may also be regarded as elements of a vector (e.g., *pn* = [*pn,*0 wish to convolve*pn,*1 1 ··· sequences *pn,n*]*T* ). of This finite happens length, in digital signal processing when we because the multiplication of polyno- mials is mathematically equivalent to the operation of sequence convolution. We will denote the set of all polynomials of degree *n* as **P***n*. If *x* is to be from the interval [*a,b*] ⊂ **R**, then the set of polynomials of degree *n* on [*a,b*] is denoted by **P***n*[*a,b*]. If *m<n* we shall usually assume **P***m*[*a,b*] ⊂ **P***n*[*a,b*].

**1.3 SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS**

Sets of objects (vectors, sequences, polynomials, functions, etc.) often have cer- tain special mappings defined on them that turn these sets into what are commonly called *function spaces.* Loosely speaking, functional analysis is about the properties

1These days it seems that the operation of convolution is first given serious study in introductory signals and systems courses. The operation of convolution is fundamental to all forms of signal processing, either analog or digital.

TLFeBOOK

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **5**

of function spaces. Generally speaking, numerical computation problems are best handled by treating them in association with suitable mappings on well-chosen function spaces. For our purposes, the three most important special types of map- pings are (1) metrics, (2) norms, and (3) inner products. You are likely to be already familiar with special cases of these really very general ideas.

The vector dot product is an example of an inner product on a vector space, while the Euclidean norm (i.e., the square root of the sum of the squares of the elements in a real-valued vector) is a norm on a vector space. The Euclidean distance between two vectors (given by the Euclidean norm of the difference between the two vectors) is a metric on a vector space. Again, loosely speaking, metrics give meaning to the concept of “distance” between points in a function space, norms give a meaning to the concept of the “size” of a vector, and inner products give meaning to the concept of “direction” in a vector space.2

In Section 1.1 we expressed interest in the sizes of errors, and so naturally the concept of a norm will be of interest. Later we shall see that inner products will prove to be useful in devising means of overcoming problems due to certain sources of error in a computation. In this section we shall consider various examples of function spaces, some of which we will work with later on in the analysis of certain computational problems. We shall see that there are many different kinds of metric, norm, and inner product. Each kind has its own particular advantages and disadvantages as will be discovered as we progress through the book.

Sometimes a quantity cannot be computed exactly. In this case we may try to estimate *bounds* on the size of the quantity. For example, finding the exact error in the truncation of a series may be impossible, but putting a bound on the error might be relatively easy. In this respect the concepts of supremum and infimum can be important. These are defined as follows.

Suppose we have *E* ⊂ **R**. We say that *E* is *bounded above* if *E* has an *upper bound*, that is, if there exists a *B* ∈ **R** such that *x* ≤ *B* for all *x* ∈ *E*. If *E* = ∅ (*empty set*; set containing no elements) there is a *supremum* of *E* [also called a *least upper bound* (lub)], denoted

sup *E.*

For example, suppose *E* = [0*,*1*)*, then any *B* ≥ 1 is an upper bound for *E*, but sup *E* = 1. More generally, sup *E* ≤ *B* for every upper bound *B* of *E*. Thus, the supremum is a “tight” upper bound. Similarly, *E* may be *bounded below*. If *E* has a *lower bound* there is a *b* ∈ **R** such that *x* ≥ *b* for all *x* ∈ *E*. If *E* = ∅, then there exists an *infimum* [also called a *greatest lower bound* (glb)], denoted by

inf *E.*

For example, suppose now *E* = *(*0*,*1]; then any *b* ≤ 0 is a lower bound for *E*, but inf *E* = 0. More generally, inf *E* ≥ *b* for every lower bound *b* of *E*. Thus, the infimum is a “tight” lower bound.

2The idea of “direction” is (often) considered with respect to the concept of an orthogonal basis in a vector space. To define “orthogonality” requires the concept of an inner product. We shall consider this in various ways later on.

TLFeBOOK

**6** FUNCTIONAL ANALYSIS IDEAS

**1.3.1 Metrics and Metric Spaces**

In mathematics an axiomatic approach is often taken in the development of analysis methods. This means that we define a set of objects, a set of operations to be performed on the set of objects, and rules obeyed by the operations. This is typically how mathematical systems are constructed. The reader (hopefully) has already seen this approach in the application of Boolean algebra to the analysis and design of digital electronic systems (i.e., digital logic). We adopt the same approach here. We will begin with the following definition.

**Definition 1.1: Metric Space, Metric** A *metric space* is a set *X* and a function *d*|*X* × *X* → **R**+, which is called a *metric* or *distance function* on *X*. If *x,y,z* ∈ *X* then *d* satisfies the following axioms:

(M1) *d(x,y)* = 0 if and only if (iff) *x* = *y*. (M2) *d(x,y)* = *d(y,x)* (symmetry property). (M3) *d(x,y)* ≤ *d(x,z)* + *d(z,y)* (triangle inequality).

We emphasize that *X* by itself cannot be a metric space until we define *d*. Thus, the metric space is often denoted by the pair *(X,d)*. The phrase “if and only if” probably needs some explanation. In (M1), if you were told that *d(x,y)* = 0, then you must immediately conclude that *x* = *y*. Conversely, if you were told that *x* = *y*, then you must immediately conclude that *d(x,y)* = 0. Instead of the words “if and only if” it is also common to write

*d(x,y)* = 0 ⇔ *x* = *y.*

The phrase “if and only if” is associated with elementary logic. This subject is reviewed in Appendix 1.B. It is recommended that the reader study that appendix before continuing with later chapters.

Some examples of metric spaces now follow.

**Example 1.1** Set *X* = **R**, with*d(x,y)* = |*x* − *y*| *(*1*.*5*)*

forms a metric space. The metric (1.5) is what is commonly meant by the “distance between two points on the real number line.” The metric (1.5) is quite useful in discussing the sizes of errors due to rounding in digital computation. This is because there is a norm on **R** that gives rise to the metric in (1.5) (see Section 1.3.2).

**Example 1.2** The set of vectors **R***n* with

*d(x,y)* =

[∑*n*−1*k*=0

[*xk* − *yk*]2]1*/*2

[*xk* − *yk*]2]1*/*2

*(*1*.*6a*)*TLFeBOOK

*(*1*.*6a*)*TLFeBOOK

*(*1*.*6a*)*TLFeBOOK

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **7**

forms a (Euclidean) metric space. However, another valid metric on **R***n* is given by

*d*1*(x,y)* =

∑*n*−1*k*=0

|*xk* − *yk*|*. (*1*.*6b*)*

In other words, we can have the metric space *(X,d)*, or *(X, d*1*)*. These spaces are different because their metrics differ.

Euclidean metrics, and their related norms and inner products, are useful in pos- ing and solving least-squares approximation problems. Least-squares approximation is a topic we shall consider in detail later.

**Example 1.3** Consider the set of (singly) infinite, complex-valued, and bounded sequences

*X* = {*x* = *(x*0*,x*1*,x*2*,...)*|*xk* ∈ **C***,*|*xk*| ≤ *c(x)(*all *k)*}*. (*1*.*7a*)*

Here *c(x)* ≥ 0 is a bound that may depend on *x*, but not on *k*. This set forms a metric space that may be denoted by *l*∞[0*,*∞] if we employ the metric

*d(x,y)* = sup

*k*∈**Z**+ |*xk* − *yk*|*. (*1*.*7b*)*

The notation [0*,*∞] emphasizes that the sequences we are talking about are only singly infinite. We would use [−∞*,*∞] to specify that we are talking about doubly infinite sequences.

**Example 1.4** Define *J* = [*a,b*] ⊂ **R**. The set *C*[*a,b*] will be a metric space if

*d(x,y)* = sup

*t*∈*J* |*x(t)* − *y(t)*|*. (*1*.*8*)*

In Example 1.1 the metric (1.5) gives the “distance” between points on the real- number line. In Example 1.4 the “points” are real-valued, continuous functions of *t* ∈ [*a,b*]. In functional analysis it is essential to get used to the idea that functions can be considered as points in a space.

**Example 1.5** The set *X* in (1.7a), where we now allow *c(x)* → ∞ (in other words, the sequence need not be bounded here), but with the metric

*d(x,y)* =

∑∞*k*=0

1

1 + |*xk* − *yk*| *(*1*.*9*)*

is a metric space. (Sometimes this space is denoted *s*.)

TLFeBOOK

2*k*+1

|*xk* − *yk*|

**8** FUNCTIONAL ANALYSIS IDEAS

**Example 1.6** Let *p* be a real-valued constant such that *p* ≥ 1. Consider the set of complex-valued sequences

*X* =

{*x* = *(x*0*,x*1*,x*2*,...)*|*xk* ∈ **C***,*

|*x*}*k*|*p <* ∞*. (*1*.*10a*)*

|*x*}*k*|*p <* ∞*. (*1*.*10a*)*

∑∞*k*=0

∑∞*k*=0

∑∞*k*=0

This set together with the metric

*d(x,y)* =

[ ∑∞*k*=0

|*xk* − *yk*|*p*]1*/p*

|*xk* − *yk*|*p*]1*/p*

*(*1*.*10b*)*

*(*1*.*10b*)*

*(*1*.*10b*)*

forms a metric space that we denote by *lp*[0*,*∞].

**Example 1.7** Consider the set of complex-valued functions on [*a,b*] ⊂ **R**

*X* =

{*x(t)*∣∣∣∣∫ *ba* |*x(t)*|2 *dt <* ∞}

*(*1*.*11a*)*

*(*1*.*11a*)*

for which

*d(x,y)* =

[∫ *ba* |*x(t)* − *y(t)*|2 *dt*]1*/*2

*(*1*.*11b*)*

*(*1*.*11b*)*

is a metric. Pair *(X,d)* forms a metric space that is usually denoted by *L*2[*a,b*].

The metric space of Example 1.7 (along with certain variations) is very impor- tant in the theory of orthogonal polynomials, and in least-squares approximation problems. This is because it turns out to be an inner product space too (see Section 1.3.3). Orthogonal polynomials have a major role to play in the solution of least squares, and other types of approximation problem.

All of the metrics defined in the examples above may be shown to satisfy the axioms of Definition 1.1. Of course, at least in some cases, much effort might be required to do this. In this book we largely avoid making this kind of effort.

**1.3.2 Norms and Normed Spaces**

So far our examples of function spaces have been metric spaces (Section 1.3.1). Such spaces are not necessarily associated with the concept of a vector space. However, normed spaces (i.e., spaces with norms defined on them) are always associated with vector spaces. So, before we can define a norm, we need to recall the general definition of a vector space.

The following definition invokes the concept of a *field* of numbers. This concept arises in abstract algebra and number theory [e.g., 2, 3], a subject we wish to avoid considering here.3 It is enough for the reader to know that **R** and **C** are fields under

3This avoidance is not to disparage abstract algebra. This subject is a necessary prerequisite to under- standing concepts such as fast algorithms for digital signal processing (i.e., fast Fourier transforms, and fast convolution algorithms; e.g., see Ref. 4), cryptography and data security, and error control codes for digital communications.

TLFeBOOK

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **9**

the usual real and complex arithmetic operations. These are really the only fields that we shall work with. We remark, largely in passing, that rational numbers (set denoted **Q**) are also a field under the usual arithmetic operations.

**Definition 1.2: Vector Space** A *vector space* (*linear space*) over a field *K* is a nonempty set *X* of elements *x,y,z,...* called *vectors* together with two algebraic operations. These operations are vector addition, and the multiplication of vectors by scalars that are elements of *K*. The following axioms must be satisfied:

(V1) If *x,y* ∈ *X*, then *x* + *y* ∈ *X* (additive closure). (V2) If *x,y,z* ∈ *X*, then *(x* + *y)* + *z* = *x* + *(y* + *z)* (associativity). (V3) There exists a vector in *X* denoted 0 (*zero vector*) such that for all *x* ∈ *X*,

we have *x* + 0 = 0 + *x* = *x*. (V4) For all *x* ∈ *X*, there is a vector −*x* ∈ *X* such that −*x* + *x* = *x* +

*(*−*x)* = 0. We call −*x* the *negative of a vector*. (V5) For all *x,y* ∈ *X* we have *x* + *y* = *y* + *x* (commutativity). (V6) If *x* ∈ *X* and *a* ∈ *K*, then the product of *a* and *x* is *ax*, and *ax* ∈ *X*. (V7) If *x,y* ∈ *X*, and *a* ∈ *K*, then *a(x* + *y)* = *ax* + *ay*. (V8) If *a,b* ∈ *K*, and *x* ∈ *X*, then *(a* + *b)x* = *ax* + *bx*. (V9) If *a,b* ∈ *K*, and *x* ∈ *X*, then *ab(x)* = *a(bx)*. (V10) If *x* ∈ *X*, and 1 ∈ *K*, then 1*x* = *x* multiplication of a vector by a unit

scalar; all fields contain a unit scalar (i.e., a number called “one”).

In this definition, as already noted, we generally work only with *K* = **R**, or *K* = **C**. We represent the zero vector by 0 just as we also represent the scalar zero by 0. Rarely is there danger of confusion.

The reader is already familiar with the special instances of this that relate to the sets **R***n* and **C***n*. These sets are vector spaces under Definition 1.2, where vector addition is defined to be

*x* + *y* =



*x*0*x...* 1 +



*y*0*y...* 1 =



*, (*1*.*12a*) xn*−1

*yn*−1

and multiplication by a field element is defined to be

*ax* =

*x*0 + *y*0 *x*1 + *... y*1 *xn*−1 + *yn*−1



*ax*0 *ax...*

1*. (*1*.*12b*) axn*−1

The zero vector is 0 = [00···00]then the elements of *x* and *y T* , and −*x* = [−*x*0 − *x*1 ···− are real-valued, and *a* ∈ **R**, but *xn*−1]*T* . if *X* = **C**If *n X* = **R***n*

then theTLFeBOOK

**10** FUNCTIONAL ANALYSIS IDEAS

elements of *x* and *y* are complex-valued, and *a* ∈ **C**. The metric spaces in Exam- ple 1.2 are therefore also vector spaces under the operations defined in (1.12a,b).

Some further examples of vector spaces now follow.

**Example 1.8** Metric space *C*[*a,b*] (Example 1.4) is a vector space under the operations

*(x* + *y)(t)* = *x(t)* + *y(t), (αx)(t)* = *αx(t), (*1*.*13*)*

where *α* ∈ **R**. The zero vector is the function that is identically zero on the interval [*a,b*].

**Example 1.9** Metric space *l*2[0*,*∞] (Example 1.6) is a vector space under the operations*x* + *y* = *(x*0*,x*1*,...)* + *(y*0*,y*1*,...)* = *(x*0 + *y*0*,x*1 + *y*1*, . . .),*

*αx* = *(αx*0*,αx*1*, . . .).* (1.14)

Here *α* ∈ **C**.

If *x,y* ∈ *l*2[0*,*∞], then some effort is required to verify axiom (V1). This requires the *Minkowski inequality*, which is

[ ∑∞*k*=0

|*xk* + *yk*|*p*]1*/p*

≤

[ ∑∞*k*=0

|*xk*|*p*]1*/p*

+

[ ∑∞*k*=0

|*yk*|*p*]1*/p*

*. (*1*.*15*)*

Refer back to Example 1.6; here we employ *p* = 2, but (1.15) is valid for *p* ≥ 1. Proof of (1.15) is somewhat involved, and so is omitted here. The interested reader can see Kreyszig [1, pp. 11–15].

We remark that the Minkowski inequality can be proved with the aid of the *H ̈older inequality*

∑∞*k*=0

[ ∑∞*k*=0

|*xk*|*p*]1*/p* [ ∑∞*k*=0

|*yk*|*q*]1*/q* |*xkyk*| ≤

*(*1*.*16*)*

for which here *p >* 1 and 1

We are now ready to define a *q* normed = 1.

space.

**Definition 1.3: Normed Space, Norm** A *normed space X* is a vector space with a norm defined on it. If *x* ∈ *X* then the norm of *x* is denoted by

||*x*|| *(*read this as “norm of *x*”*).*

The norm must satisfy the following axioms:

(N1) ||*x*|| ≥ 0 (i.e., the norm is nonnegative).

TLFeBOOK

*p* + 1

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **11**

(N2) ||*x*|| = 0 ⇔ *x* = 0. (N3) ||*αx*|| = |*α*| ||*x*||. Here *α* is a scalar in the field of *X* (i.e., *α* ∈ *K*; see

Definition 3.2). (N4) ||*x* + *y*|| ≤ ||*x*|| + ||*y*|| (triangle inequality).

The normed space is vector space *X* together with a norm, and so may be properly denoted by the pair *(X,*|| · ||*)*. However, we may simply write *X*, and say “normed space *X*,” so the norm that goes along with *X* is understood from the context of the discussion.

It is important to note that all normed spaces are also metric spaces, where the metric is given by

*d(x,y)* = ||*x* − *y*|| *(x, y* ∈ *X). (*1*.*17*)*

The metric in (1.17) is called the *metric induced by the norm*.

Various other properties of norms may be deduced. One of these is:

**Example 1.10** Prove | ||*y*|| − ||*x*|| | ≤ ||*y* − *x*||.

***Proof*** From (N3) and (N4)

||*y*|| = ||*y* − *x* + *x*|| ≤ ||*y* − *x*|| + ||*x*||*,*||*x*|| = ||*x* − *y* + *y*|| ≤ ||*y* − *x*|| + ||*y*||*.*

Combining these, we obtain

||*y*|| − ||*x*|| ≤ ||*y* − *x*||*,*||*y*|| − ||*x*|| ≥ −||*y* − *x*||*.*

The claim follows immediately.

We may regard the norm as a mapping from *X* to set **R**: || · |||*X* → **R**. This mapping can be shown to be continuous. However, this requires generalizing the concept of continuity that you may know from elementary calculus. Here we define continuity as follows.

**Definition 1.4: Continuous Mapping** Suppose *X* = *(X,d)* and *Y* = *(Y,*

*d)* are two metric spaces. The mapping *T* |*X* → *Y* is said to be *continuous at a point x*0 ∈ *X* if for all *ε >* 0 there is a *δ >* 0 such that

*d(Tx,Tx*0*)<ε* for all *x* satisfying *d(x,x*0*) < δ. (*1*.*18*)*

*T* is said to be *continuous* if it is continuous at every point of *X*.

Note that *Tx* is just another way of writing *T (x)*. *(***R***,*|·|*)* is a normed space; that is, the set of real numbers with the usual arithmetic operations defined on it is aTLFeBOOK

*d(*||*x*||*,*||*x*0||*)* = | ||*x*|| − ||*x*0|| | *< ε, d(x, x*0*)* = ||*x* − *x*0|| *< δ.*

To make these claims, we are using (1.17). In other words, *X* and *Y* are normed spaces, and we employ the metrics induced by their respective norms. In addition, we identify *T* with || · ||. Using Example 1.10, we obtain

| ||*x*|| − ||*x*0|| | ≤ ||*x* − *x*0|| *< δ.*

Thus, the requirements of Definition 1.4 are met, and so we conclude that norms are continuous mappings.

We now list some other normed spaces.

**Example 1.11** The *Euclidean space* **R***n* and the *unitary space* **C***n* are both normed spaces, where the norm is defined to be

||*x*|| =

[*n*−1∑*k*=0

|*xk*|2]1*/*2

*. (*1*.*19*)*

For **R***n* the absolute value bars may be dropped.4 It is easy to see that *d(x,y)* = ||*x* − *y*|| gives the same metric as in (1.6a) for space **R***n*. We further remark that for *n* = 1 we have ||*x*|| = |*x*|.

**Example 1.12** The space *lp*[0*,*∞] is a normed space if we define the norm to be

||*x*|| =

[ ∑∞*k*=0

|*xk*|*p*]1*/p*

*(*1*.*20*)*

for which *d(x,y)* = ||*x* − *y*|| coincides with the metric in (1.10b).

**Example 1.13** The sequence space*l*∞[0*,*∞] from Example 1.3 of Section 1.3.1 is a normed space, where the norm is defined to be

||*x*|| = sup

*k*∈**Z**+ |*xk*|*, (*1*.*21*)*

and this norm induces the metric of (1.7b).

4Suppose *z* = *x* + *jy* (*j* = √

**12** FUNCTIONAL ANALYSIS IDEAS

vector space, and the absolute value of an element of **R** is the norm of that element. If we identify *Y* in Definition 1.4 with metric space *(***R***,*|·|*)*, then (1.18) becomes

−1, *x,y* ∈ **R**) is some arbitrary complex number. Recall that *z*2 = |*z*|2 in general.

TLFeBOOK

*d(Tx,Tx*0*)* =

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **13**

**Example 1.14** The space *C*[*a,b*] first seen in Example 1.4 is a normed space, where the norm is defined by

||*x*|| = sup

*t*∈*J* |*x(t)*|*. (*1*.*22*)*

Naturally, this norm induces the metric of (1.8).

**Example 1.15** The space *L*2[*a,b*] of Example 1.7 is a normed space for the norm

||*x*|| =

[∫ *ba* |*x(t)*|2 *dt*]1*/*2

*. (*1*.*23*)*

This norm induces the metric in (1.11b).

The normed space of Example 1.15 is important in the following respect. Observe that

||*x*||2 =

∫ *ba* |*x(t)*|2 *dt. (*1*.*24*)*

Suppose we now consider a resistor with resistance *R*. If the voltage drop across its terminals is *v(t)* and the current through it is *i(t)*, we know that the instantaneous power dissipated in the device is *p(t)* = *v(t)i(t)*. If we assume that the resistor is a linear device, then *v(t)* = *Ri(t)* via Ohm’s law. Thus

*p(t)* = *v(t)i(t)* = *Ri*2*(t). (*1*.*25*)*

Consequently, the amount of energy delivered to the resistor over time interval *t* ∈ [*a,b*] is given by

*E* = *R*

∫ *ba i*2*(t)dt. (*1*.*26*)*

If the voltage/current waveforms in our circuit containing *R* belong to the space *L*2[*a,b*], then clearly *E* = *R*||*i*||2. We may therefore regard the square of the *L*2 norm [given by (1.24)] of a signal to be the *energy of the signal*, provided the norm exists. This notion can be helpful in the optimal design of electric circuits (e.g., electric filters), and also of optimal electronic circuits. In analogous fashion, an element *x* of space *l*2[0*,*∞] satisfies

||*x*||2 =

∑∞*k*=0

|*xk*|2 *<* ∞ *(*1*.*27*)*

[see (1.10a) and Example 1.12]. We may consider ||*x*||2 to be the *energy of the single-sided sequence x*. This notion is useful in the optimal design of digital filters.TLFeBOOK

**14** FUNCTIONAL ANALYSIS IDEAS

**1.3.3 Inner Products and Inner Product Spaces**

The concept of an inner product is necessary before one can talk about orthogonal bases for vector spaces. Recall from elementary linear algebra that orthogonal bases were important in representing vectors. From a computational standpoint, as mentioned earlier, orthogonal bases can have a simplifying effect on certain types of approximation problem (e.g., least-squares approximations), and represent a means of controlling numerical errors due to so-called ill-conditioned problems.

Following our axiomatic approach, consider the following definition.

**Definition 1.5: Inner Product Space, Inner Product** An *inner product space* is a vector space *X* with an *inner product* defined on it. The inner product is a mapping 〈·*,*·〉|*X* × *X* → *K* that satisfies the following axioms:

(I1) 〈*x* + *y,z*〉=〈*x,z*〉+〈*y,z*〉. (I2) 〈*αx,y*〉 = *α*〈*x,y*〉. (I3) 〈*x,y*〉=〈*y,x*〉∗. (I4) 〈*x,x*〉 ≥ 0, and 〈*x,x*〉 = 0 ⇔ *x* = 0.

Naturally, *x,y,z* ∈ *X*, and *α* is a scalar from the field *K* of vector space *X*. The asterisk superscript on 〈*y,x*〉 in (I3) denotes complex conjugation.5

If the field of *X* is not **C**, then the operation of complex conjugation in (I3) is redundant.

All inner product spaces are also normed spaces, and hence are also metric spaces. This is because the inner product induces a norm on *X*

||*x*|| = [〈*x,x*〉]1*/*2 *(*1*.*28*)*

for all *x* ∈ *X*. Following (1.17), the induced metric is

*d(x,y)* = ||*x* − *y*|| = [〈*x* − *y,x* − *y*〉]1*/*2*. (*1*.*29*)*

Directly from the axioms of Definition 1.5, it is possible to deduce that (for *x,y,z* ∈ *X* and *a,b* ∈ *K*)〈*ax* + *by, z*〉 = *a*〈*x,z*〉 + *b*〈*y,z*〉*,* (1.30a)

〈*x, ay*〉 = *a*∗〈*x,y*〉*,* (1.30b)

and

〈*x, ay* + *bz*〉 = *a*∗〈*x,y*〉 + *b*∗〈*x,z*〉*. (*1*.*30c*)*

The reader should prove these as an exercise.

5If *z* = *x* + *yj* is a complex number, then its conjugate is *z*∗ = *x* − *yj*.

TLFeBOOK

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **15**

We caution the reader that not all normed spaces are inner product spaces. We may construct an example with the aid of the following example.

**Example 1.16** Let *x,y* be from an inner product space. If || · || is the norm induced by the inner product, then ||*x* + *y*||2 + ||*x* − *y*||2 = 2*(*||*x*||2 + ||*y*||2*)*. This is the *parallelogram equality*.

***Proof*** Via (1.30a,c) we have

||*x* + *y*||2 = 〈*x* + *y,x* + *y*〉=〈*x,x* + *y*〉+〈*y,x* + *y*〉

= 〈*x,x*〉+〈*x,y*〉+〈*y,x*〉+〈*y,y*〉*,*

and

||*x* − *y*||2 = 〈*x* − *y,x* − *y*〉=〈*x,x* − *y*〉−〈*y,x* − *y*〉

= 〈*x,x*〉−〈*x,y*〉−〈*y,x*〉+〈*y,y*〉*.*

Adding these gives the stated result.

It turns out that the space *lp*[0*,*∞] with *p* = 2 is not an inner product space. The parallelogram equality can be used to show this. Consider *x* = *(*1*,*1*,*0*,*0*, . . .), y* = *(*1*,*−1*,*0*,*0*,...)*, which are certainly elements of *lp*[0*,*∞] [see (1.10a)]. We see that

||*x*|| = ||*y*|| = 21*/p,*||*x* + *y*|| = ||*x* − *y*|| = 2*.*

The parallelogram equality is not satisfied, which implies that our norm does not come from an inner product. Thus, *lp*[0*,*∞] with *p* = 2 cannot be an inner product space.

On the other hand, *l*2[0*,*∞] **is** an inner product space, where the inner product is defined to be

〈*x,y*〉 =

∑∞*k*=0

*xkyk*∗*. (*1*.*31*)*

Does this infinite series converge? Yes, it does. To see this, we need the *Cauchy– Schwarz inequality*.6 Recall the H ̈older inequality of (1.16). Let *p* = 2, so that *q* = 2. Then the Cauchy–Schwarz inequality is

∑∞*k*=0

[ ∑∞*k*=0

|*xk*|2]1*/*2 [ ∑∞*k*=0

|*yk*|2]1*/*2 |*xkyk*| ≤

*. (*1*.*32*)*

6The inequality we consider here is related to the Schwarz inequality. We will consider the Schwarz inequality later on. This inequality is of immense practical value to electrical and computer engineers. It is used to derive the matched-filter receiver, which is employed in digital communications systems, to derive the uncertainty principle in quantum mechanics and in signal processing, and to derive the Cramér–Rao lower bound on the variance of parameter estimators, to name only three applications.

TLFeBOOK

**16** FUNCTIONAL ANALYSIS IDEAS

Now

|〈*x,y*〉| =

∣∣∣∣∣

|*xkyk*|*. (*1*.*33*)*

The inequality in (1.33) follows from the triangle inequality for |·|. (Recall that the absolute then |*z*| = value √

*x*2 operation + *y*2.) The is a right-hand norm on **R**. side It is of also (1.32) a norm is finite on **C**; because if *z* = *x x* + and *jy y* ∈ are **C**,

in *l*2[0*,*∞]. Thus, from (1.33), 〈*x,y*〉 is finite. Thus, the series (1.31) converges.

It turns out that *C*[*a,b*] is not an inner product space, either. But we will not demonstrate the truth of this claim here.

Some further examples of inner product spaces are as follows.

**Example 1.17** The *Euclidean space* **R***n* is an inner product space, where the inner product is defined to be

〈*x,y*〉 =

∑*n*−1*k*=0

*xkyk. (*1*.*34*)*

The reader will recognize this as the *vector dot product* from elementary linear algebra; that is, *x* · *y* = 〈*x,y*〉. It is well worth noting that

〈*x,y*〉 = *xT y. (*1*.*35*)*

Here the superscript *T* denotes *transposition.* So, *xT* is a row vector. The inner product in (1.34) certainly induces the norm in (1.19).

**Example 1.18** The *unitary space* **C***n* is an inner product space for the inner product

〈*x,y*〉 =

*n*−1∑*k*=0

*xkyk*∗*. (*1*.*36*)*

Again, the norm of (1.19) is induced by inner product (1.36). If *H* denotes the operation of complex conjugation and transposition (this is called *Hermitian trans- position*), then

*yH* = [*y*0∗*y*1 ∗···*yn*−1∗]

(row vector), and

〈*x,y*〉 = *yHx. (*1*.*37*)*

**Example 1.19** The space *L*2[*a,b*] from Example 1.7 is an inner product space if the inner product is defined to be

〈*x,y*〉 =

∫ *ba x(t)y*∗*(t)dt. (*1*.*38*)*TLFeBOOK

∞∑*k*=0

*xky*∗*k*∣∣∣∣∣ ≤

∞∑*k*=0

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **17**

The norm induced by (1.38) is

||*x*|| =

[∫ *ba* |*x(t)*|2 *dt*]1*/*2

*. (*1*.*39*)*

This in turn induces the metric in (1.11b).

Now we consider the concept of orthogonality in a completely general manner.

**Definition 1.6: Orthogonality** Let *x,y* be vectors from some inner product space *X*. These vectors are *orthogonal* iff

〈*x,y*〉 = 0*.*

The orthogonality of *x* and *y* is symbolized by writing *x* ⊥ *y*. Similarly, for subsets *A, B* ⊂ *X* we write *x* ⊥ *A* if *x* ⊥ *a* for all *a* ∈ *A*, and *A* ⊥ *B* if *a* ⊥ *b* for all *a* ∈ *A*, and *b* ∈ *B*.

If we consider the inner product space **R**2, then it is easy to see that 〈[1 0]*T ,*[0 1]*T* 〉 = 0, so [01]*T* , and [10]*T* are orthogonal vectors. In fact, these vectors form an orthogonal basis for **R**2, a concept we will consider more gen- erally recall this below. that reasoning any If to *x* we ∈ **R***n* **R**define 2 for can *n* the *>* be 2 *unit* expressed should *vectors* be as *e*clear.) 0 *x* = = [1 *x*Another 0]0*eT* 0 *,* and *e*1 = [0 1]*T* , then we + *x*1*e*1. (The extension of example of a pair of orthogonal vectors would be *x* = 1 an orthogonal basis for the space **R**2.

2[1 − 1]*T* . These too form

Define the functions

*φ(x)* =

{ 0*, x <* 0 and *x* ≥ 1

1*,* 0 ≤ *x <* 1 *(*1*.*40*)*

and

*ψ(x)* =



0*, x <* 0 and *x* ≥ 1 1*,* 0 ≤ *x <* 1

*. (*1*.*41*)*

Function *φ(x)* is called the *Haar scaling function*, and function *ψ(x)* is called the *Haar wavelet* [5]. The function *φ(x)* is also called an *non-return-to-zero* (NRZ) *pulse*, and function *ψ(x)* is also called a *Manchester pulse* [6]. It is easy to con- firm that these pulses are elements of *L*2*(***R***)* = *L*2*(*−∞*,*∞*)*, and that they are orthogonal, that is, 〈*φ,ψ*〉 = 0 under the inner product defined in (1.38). This is so because

〈*φ,ψ*〉 =

2 −1*,* 1

∫ ∞*φ(x)ψ*∗*(x)dx* =

∫ 1*ψ(x)dx* = 0*.* −∞ 0 TLFeBOOK

√

2[1 1]*T* , and *y* = 1

2 ≤ *x <* 1

√

**18** FUNCTIONAL ANALYSIS IDEAS

Thus, we consider *φ* and *ψ* to be elements in the inner product space *L*2*(***R***)*, for which the inner product is

〈*x,y*〉 =

∫ ∞−∞ *x(t)y*∗*(t)dt.*

It turns out that the Haar wavelet is the simplest example of the more general class of *Daubechies wavelets*. The general theory of these wavelets first appeared in Daubechies [7]. Their development has revolutionized signal processing and many other areas.7 The main reason for this is the fact that for any *f(t)* ∈ *L*2*(***R***)*

*f(t)* =

∑∞*n*=−∞

∑∞*k*=−∞〈*f, ψn,k*〉*ψn,k(t), (*1*.*42*)*

where *ψn,k(t)* = 2*n/*2*ψ(*2*nt* − *k)*. This doubly infinite series is called a *wavelet series expansion* for *f* . The coefficients *fn,k* = 〈*f, ψn,k*〉 have finite energy. In effect, if we treat either *k* or *n* as a constant, then the resulting doubly infinite sequence is in the space *l*2[−∞*,*∞]. In fact, it is also the case that

∑∞*n*=−∞

∑∞*k*=−∞|*fn,k*|2 *<* ∞*. (*1*.*43*)*

It is to be emphasized that the *ψ* used in (1.42) could be (1.41), or it could be chosen from the more general class in Ref. 7. We shall not prove these things in this book, as the technical arguments are quite hard.

The wavelet series is presently not as familiar to the broader electrical and computer engineering community as is the *Fourier series*. A brief summary of the Fourier series is as follows. Again, rigorous proofs of many of the following claims will be avoided, though good introductory references to Fourier series are Tolstov [8] or Kreyszig [9]. If *f* ∈ *L*2*(*0*,*2*π)*, then

*f(t)* =

∑∞*n*=−∞*fnejnt, j* = √

−1*, (*1*.*44*)*

where the *Fourier* (*series*) *coefficients* are given by

*fn* = 1

∫ 0 2*π*

*f(t)e*−*jnt dt. (*1*.*45*)*

We may define

*en(t)* = exp*(jnt) (t* ∈ *(*0*,*2*π), n* ∈ **Z***) (*1*.*46*)*

7For example, in digital communications the problem of designing good signaling pulses for data transmission is best treated with respect to wavelet theory.

TLFeBOOK

2*π*

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **19**

so that we see

〈*f, en*〉 = 1

[*ejnt*]∗ *dt* = *fn. (*1*.*47*)*

The series (1.44) is the *complex Fourier series expansion* for *f*. Note that for *n, k* ∈ **Z**

exp[*jn(t* + 2*πk)*] = exp[*jnt*] exp[2*πjnk*] = exp[*jnt*]*. (*1*.*48*)*

Here we have used *Euler’s identity*

*ejx* = cos*x* + *j* sin*x (*1*.*49*)*

and cos*(*2*πk)* = 1*,*sin*(*2*πk)* = 0. The function *ejnt* is therefore 2*π*-periodic; that is, its period is 2*π*. It therefore follows that the series on the right-hand side of (3.40) is a 2*π*-periodic function, too. The result (1.48) implies that, although *f* in (1.44) is initially defined only on *(*0*,*2*π)*, we are at liberty to “periodically extend” *f* over the entire real-number line; that is, we can treat *f* as one period of the periodic function

*f(t)*  ̃= ∑*k*∈**Z**

*f (t* + 2*πk) (*1*.*50*)*

for which *f(t)* = *f(t)*  ̃for *t* ∈ *(*0*,*2*π)*. Thus, series (1.44) is a way to represent periodic functions. Because *f* ∈ *L*2*(*0*,*2*π)*, it turns out that

∑∞*n*=−∞|*fn*|2 *<* ∞ *(*1*.*51*)*

so that Observe *(fn)* that ∈ *l*2in [−∞*,*∞].

(1.47) we have “redefined” the inner product on *L*2*(*0*,*2*π)* to be

〈*x,y*〉 = 1

∫ 2*π*

0 *x(t)y*∗*(t)dt (*1*.*52*)*

which differs from (1.38) happens to be a valid inner in product that it on has the the vector factor space 2*π* 1

in front. This variation also defined by the set in (1.11a). Actually, it is a simple example of a weighted inner product.

Now consider, for *n* = *m*

〈*en,em*〉 = 1

[*ej (n*−*m)t*]2*π*0

= *e*2*πj (n*−*m)* − 1

2*πj (n* − *m)* = 0*.* (1.53)

Similarly

〈*en,en*〉 = 1

∫ 2*π*

0 *dt* = 1*. (*1*.*54*)* So, *en* and *em* (if *n* = *m*) are orthogonal with respect to the inner product in (1.52).TLFeBOOK

2*π*

2*πj (n* − *m)* = 1 − 1

∫ 2*π*

0 *ejnte*−*jmt dt* = 1

2*π*

∫ 2*π*

0 *ejnte*−*jnt dt* = 1

2*π*

∫ 2*π*

0 *f(t)*

2*π*

2*πj (n* − *m)*

2*π*

**20** FUNCTIONAL ANALYSIS IDEAS

From basic electric circuit analysis, periodic signals have finite power. Therefore, series (1.44) is a way to represent finite power signals.8 We might therefore consider the space *L*2*(*0*,*2*π)* to be the “space of finite power signals.” From considerations involving the wavelet series representation of (1.42), we may consider *L*2*(***R***)* to be the “space of finite energy signals.” Recall also the discussion at the end of Section 1.3.2 (last paragraph).

An example of a Fourier series expansion is the following.

**Example 1.20** Suppose that

*f(t)* =

{ 1*,* −1*,* 0 *<t<π*

*π* ≤ *t <* 2*π . (*1*.*55*)*

A sketch of this function is one period of a 2*π*-periodic square wave. The Fourier coefficients are given by (for *n* = 0)

*fn* = 1

∫ *π* 2*π*

*e*−*jnt dt*]

= 1

]

= 1

)*,*

(1.56) where we have made use of

sin*x* = 2*j* 1

[*ejx* − *e*−*jx*]*. (*1*.*57*)*

This is easily derived using the Euler identity in (1.49). For *n* = 0, it should be clear that *f*0 = 0.

The coefficients *fn* in (1.56) involve expressions containing *j*. Since *f(t)* is real-valued, it therefore follows that we can rewrite the series expansion in such a manner as to avoid complex arithmetic. It is almost a standard practice to do this. We now demonstrate this process:

∑∞*n*=−∞*fnejnt* = 2

2 *n*)*e*−*jnt*]

8In fact, using phasor analysis and superposition, you can apply (1.44) to determine the steady-state output of a circuit for any periodic input (including, and especially, nonsinusoidal periodic functions). This makes the Fourier series very important in electrical/electronic circuit analysis.

TLFeBOOK

2*π*

2*π*

*π*

1 − *e*−*jnπ*

∫ 2*π*

0 *f(t)e*−*jnt dt* = 1

[− 1

2 *n*)*ejnt*]

= 2

*jn* = 2

*jn*

*π*

*π*

[ ∑∞[ *n*=1 ∑∞*n*=1

[*e*−*jnt*]*π*0 + 1

*ne*−*jnπ/*2 sin1

(*π*

*ne*−*jnπ/*2 sin1

(*π*

*πne*−*jnπ/*2*ejnπ/*2 − *e*−*jnπ/*2

*jn*

2*π*

[*e*−*jnt*]2*ππ*

[∫ *π*0 *e*−*jnt dt* −

2 *n*)*ejnt* +

2 *n*)*ejnt* +

2*j* = 2

]

= 1

2*π*

−1∑*n*=−∞

∞∑*n*=1

[1 − *e*−*jnπ* − *e*−*jnπ* + 1

*nejnπ/*2 sin1

(*π*

*πne*−*jnπ/*2 sin(*πn*

*ne*−*jnπ/*2 sin1

(*π*

*jn*

2

2 ) *n*Here we have used the fact that (see Appendix 1.A)

*ejnte*−*jπn/*2 + *e*−*jntejπn/*2 = 2 Re [*ejnte*−*jπn/*2] = 2 cos[*n*(*t* − *π*

)]

*.*

This is so because if *z* = *x* + *jy*, then *z* + *z*∗ = 2*x* = 2 Re [*z*]. Since

cos*(α* + *β)* = cos*α* cos*β* − sin*α* sin*β,*

we have

cos[*n*(*t* − *π*

2 *.*

However, if *n* is an even number, then sin*(πn/*2*)* = 0, and if *n* is an odd number, then cos*(πn/*2*)* = 0. Therefore

4

]

*,*

but sin2[*(*2*n* + 1*)π*

2*n* + 1 sin[*(*2*n* + 1*)t*]*.*

It is important to note that the wavelet series and Fourier series expansions have something in common, in spite of the fact that they look quite different and indeed are associated with quite different function spaces. The common feature is that both representations involve the use of orthogonal basis functions. We are now ready to consider this in a general manner.

Begin by recalling from elementary linear algebra that a basis for a vector space such as *X* = **R***n* or *X* = **C***n* is a set of *n* vectors, say

*B* = {*e*0*,e*1*,...,en*−1} *(*1*.*58*)*

such that the elements *ek* (basis vectors) are *linearly independent*. This means that no vector in the set can be expressed as a linear combination of any of the others.TLFeBOOK

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **21**

= 2

2 *n*)[*ejnte*−*jπn/*2 + *e*−*jntejπn/*2]

= 4

2 ] = 1, so finally we have

*f(t)* =

∑∞*n*=−∞

*fnejnt* = 4

*π*

*π*

*π*

2 ∑∞) *nn*=1

= 4

∞∑*n*=1

∞∑*n*=1

*π*

*n* cos[*n*1

(*t* − *π*

*n* sin(*π*

*n* cos[*n*(*t* − *π*

1

1

∞∑*n*=0

2

2*n* + 1 sin [*(*2*n* + 1*)t*] sin2 [*(*2*n* + 1*)π*

)] = cos*(nt)* cos *πn*

1

2

2

)]sin(*π*

)]

sin(*π*

*π*

∞∑*n*=0

2 + sin*(nt)*sin *πn*

1

2

2

**22** FUNCTIONAL ANALYSIS IDEAS

In general, it is not necessary that 〈*ek,en*〉 = 0 for *n* = *k*. In other words, indepen- dence does not require orthogonality. However, if set *B* is a basis (orthogonal or otherwise) then for any *x* ∈ *X* (vector space) there exists a set of coefficients from the field of the vector space, say, *b* = {*b*0*,b*1 *...,bn*−1}, such that

*x* =

∑*n*−1*k*=0

*bkek. (*1*.*59*)*

We say that spaces **R***n* and **C***n* are of *dimension n*. This is a direct reference to the number of basis vectors in *B*. This notion generalizes.

Now let us consider a sequence space (e.g., *l*2[0*,*∞]). Suppose *x* = *(x*0*,x*1*,x*2*,...)* ∈ *l*2[0*,*∞]. Define the following unit vector sequences:

*e*0 = *(*1*,*0*,*0*,*0*, . . .), e*1 = *(*0*,*1*,*0*,*0*, . . .), e*2 = *(*0*,*0*,*1*,*0*, . . .),* etc. *(*1*.*60*)*

Clearly

*x* =

∑∞*k*=0

*xkek. (*1*.*61*)*

It is equally clear that no vector *ek* any of the others. Thus, the *countably* basis for *l*2[0*,*∞]. The sequence space can be expressed as a linear combination of *infinite* set9 *B* = {*e*0*,e*1*,e*2*,...*} forms a is therefore of *infinite dimension* because *B* has a countable infinity of members. It is apparent as well that, under the inner product defined in (1.31), we have 〈*en,em*〉 = *δn*−*m*. Sequence *δ* = *(δn)* is called the *Kr ̈onecker delta sequence*. It is defined by

*δn* =

{ 1*,* 0*, n* = 0

*n* = 0 *. (*1*.*62*)*

Therefore, the vectors in (1.60) are mutually orthogonal as well. So they happen to form an orthogonal basis for *l*2[0*,*∞]. Of course, this is not the only possible basis. In general, given a countably sarily those in (1.60)] that for any *x* ∈ *l*2[0*,*∞] infinite set of vectors {*ek*|*k* ∈ **Z**+} [no longer are linearly independent, and such that *ek* neces- ∈ *l*2[0*,*∞], there will exist coefficients *ak* ∈ **C** such that

*x* =

∑∞*k*=0

*akek. (*1*.*63*)*

In view of the above, consider the following linearly independent set of vectors from some inner product space *X*:

*B* = {*ek*|*ek* ∈ *X, k* ∈ **Z**}*. (*1*.*64*)*

9A set *A* is countably infinite if its members can be put into one-to-one (1–1) correspondence with the members of the set **Z**+. This is also equivalent to being able to place the elements of *A* into 1–1 correspondence with the elements of **Z**.

TLFeBOOK

SOME SPECIAL MAPPINGS: METRICS, NORMS, AND INNER PRODUCTS **23**

Assume that this is a basis for *X*. In this case for any *x* ∈ *X*, there are coefficients *ak* such that

*x* = ∑*k*∈**Z**

*akek. (*1*.*65*)*

We define the set *B* to be orthogonal iff for all *n, k* ∈ **Z**

〈*en,ek*〉 = *δn*−*k. (*1*.*66*)*

Assume that the elements of *B* in (1.64) satisfy (1.66). It is then easy to see that

〈*x,en*〉 =

〈∑*k*

*akek,en*〉

= ∑*k*

〈*akek,en*〉 *(*using (I1)*)*

= ∑*k*

*ak*〈*ek,en*〉 *(*using (I2)*)*

= ∑*k*

*δk*−*nak (*using (1.66)*)*

so finally we may say that

〈*x,en*〉 = *an. (*1*.*67*)*

In other words, if the basis *B* is orthogonal, then

*x* = ∑*k*∈**Z**〈*x,ek*〉*ek. (*1*.*68*)*

Previous examples (e.g., Fourier series expansion) are merely special cases of this general idea. We see that one of the main features of an orthogonal basis is the ease with which we can obtain the coefficients *ak*. Nonorthogonal bases are harder to work with in this respect. This is one of the reasons why orthogonal bases are so universally popular.

A few comments on terminology are in order here. Some would say that the condition (1.66) on *B* in (1.64) means that *B* is an *orthonormal set*, and we would say that condition

〈*en,ek*〉 = *αnδn*−*k*

is the condition for *B* to be an orthogonal set, where *αn* is not necessarily unity (i.e., equal to one) for all *n*. However, in this book we often insist that orthogonal basis vectors be “normalized” so condition (1.66) holds.

We conclude the present section by considering the following theorem. It was mentioned in a footnote that the following *Schwarz inequality* (or variations of it) is of very great value in electrical and computer engineering.

TLFeBOOK

**24** FUNCTIONAL ANALYSIS IDEAS

**Theorem 1.1: Schwarz Inequality** Let *X* be an inner product space, where *x,y* ∈ *X*. Then

|〈*x,y*〉| ≤ ||*x*|| ||*y*||*. (*1*.*69*)*

Equality holds iff {*x,y*} is a linearly dependent set.

***Proof*** If *y* = 0 then 〈*x,*0〉 = 0, and (1.69) clearly holds in this special case. Let *y* = 0. For all scalars *α* in the field of *X* we must have [via inner product axioms and (1.30)]

0 ≤ ||*x* − *αy*||2 = 〈*x* − *αy, x* − *αy*〉

= 〈*x,x*〉 − *α*∗〈*x,y*〉 − *α*[〈*y,x*〉 − *α*∗〈*y,y*〉]*.*

If we select *α*∗ = 〈*y,x*〉*/*〈*y,y*〉, then the quantity in the brackets [·] vanishes. Thus

0 ≤ 〈*x,x*〉 − 〈*y,x*〉

||*y*||2

[using 〈*x,y*〉=〈*y,x*〉∗, i.e., axiom (I3)]. Rearranging, this yields

|〈*x,y*〉|2 ≤ ||*x*||2||*y*||2*,*

and the result (1.69) follows (we must take positive square roots as ||*x*|| ≥ 0, and |*x*| ≥ 0).

Equality holds iff *y* = 0, or else ||*x* − *αy*||2 = 0, hence *x* − *αy* = 0 [recall (N2)], so *x* = *αy*, demonstrating linear dependence of *x* and *y*.

We may now see what Theorem 1.1 has to say when applied to the special case of a vector dot product.

**Example 1.21** Suppose that *X* is the inner product space of Example 1.17. Since

|〈*x,y*〉| =

*xkyk*∣∣∣∣∣

and ||*x*|| =

∣∣∣∣∣*n*−1∑*k*=0 [∑*n*−1 *k*=0 *xk*2]1*/*2, we have from Theorem 1.1 that

∣∣∣∣∣*n*−1∑*k*=0*xkyk*∣∣∣∣∣ ≤

*yk*2]1*/*2

*. (*1*.*70*)*

If *yk* = *αxk* (*α* ∈ **R**) for all *k* ∈ **Z***n*, then

∣∣∣∣∣∑*n*−1*k*=0

[*n*−1∑*xk*2]1*/*2 [∑*n*−1*k*=0 *k*=0

*xkyk*∣∣∣∣∣ = |*α*|

∑*n*−1*k*=0

*xk*2*,*

TLFeBOOK

〈*y,y*〉〈*x,y*〉 = ||*x*||2 − |〈*x,y*〉|2

THE DISCRETE FOURIER SERIES (DFS) **25**

and

[∑*n*−1 *k*=0 *yk*2]1*/*2 = |*α*|[∑*n*−1 *k*=0 *xk*2]1*/*2, hence

[∑*n*−1*k*=0

*xk*2]1*/*2 [∑*n*−1*k*=0

*yk*2]1*/*2

= |*α*|

∑*n*−1*k*=0

*xk*2*.*

Thus, (1.70) does indeed hold with equality when *y* = *αx*.

**1.4 THE DISCRETE FOURIER SERIES (DFS)**

The subject of discrete Fourier series (DFS) and its relationship to the complex Fourier series expansion of Section 1.3.3 is often deferred to later courses (e.g., signals and systems), but will be briefly considered here as an additional example of an orthogonal series expansion.

The complex Fourier series expansion of Section 1.3.3 was for 2*π*-periodic functions defined on the real-number line. A similar series expansion exists for *N*-periodic sequences such as ̃*x* = *(* ̃*xn)*; that is, for *N* ∈ {2*,*3*,*4*,...*} ⊂ **Z**, consider

̃*xn* = ∑*k*∈**Z**

*xn*+*kN (*1*.*71*)*

where *x* = *(xn)* is such that *xn* = 0 for *n <* 0, and for *n* ≥ *N* as well. Thus, *x* is just one period of ̃*x*. We observe that

̃*xn*+*mN* =

∑∞*k*=−∞

*xn*+*mN*+*kN* =

∑∞*xn*+*(m*+*k)N* =

∑ ∞*xn*+*rN* = ̃*xn k*=−∞*r*=−∞(*r* = *m* + *k*). This confirms that ̃*x* is indeed *N*-periodic (i.e., periodic with period *N*). We normally assume in a context such as a vector: *x* = [*x*0 *x*1 ··· *xN*−1]*T* ∈ **C***N*. as this that *xn* ∈ **C**. We also regard *x* An inner product may be defined on the space of *N*-periodic sequences according to

〈 ̃*x,*  ̃*y*〉=〈*x,y*〉 = *yHx (*1*.*72*)*

(recall Example 1.18), where *y* ∈ **C***N* is one period of ̃*y*. We assume, of course, that ̃*x* and ̃*y* are bounded sequences so that (1.72) is well defined. Now define *ek* = [*ek,*0 *ek,*1 ··· *ek,N*−1]*T* ∈ **C***N* according to

*ek,n* = exp[*j* 2*π*

*N* ]*kn, (*1*.*73*)*

where *n* ∈ **Z***N*. The periodization of *ek* = *(ek,n)* is

̃*ek,n* = ∑*m*∈**Z***ek,n*+*mN (*1*.*74*)*TLFeBOOK

**26** FUNCTIONAL ANALYSIS IDEAS

yielding ̃*ek* = *(* ̃*ek,n)*. That (1.73) is periodic with period *N* with respect to index *n* is easily seen:

*ek,n*+*mN* = exp[*j* 2*π*

*N kn*]exp[*j*2*πkm*] = *ek,n.*

It can be shown (by exercise) that [using definition (1.72)]

〈 ̃*ek,*  ̃*er*〉=〈*ek,er*〉 =

exp*N*−1∑[−*j* 2*π n*=0

{ *N,* 0*, k* − *r* = 0

otherwise *.* (1.75)

Thus, if orthogonal, we consider and so *(ek,n)*, form an and orthogonal *(er,n)* with *k* = *r* we find that these sequences basis for the vector space **C***N*. are From (1.75) we may write

〈*ek,er*〉 = *Nδk*−*r. (*1*.*76*)*

Thus, there must exist another vector *X* = [*X*0 *X*1 ··· *XN*−1]*T* ∈ **C***N* such that

*xn* = 1

*N* ]

*kn(*1*.*77*)*

for *n* ∈ **Z***N*. In fact

〈*x,er*〉 =

*N*−1∑*n*=0

*xner,n* ∗= 1

*Xk(Nδk*−*r)* = *Xr.* (1.78)

That is

*Xk* =

*xn* exp*N N*−1∑[−*j* 2*π*

*kn*]

*(*1*.*79*) n*=0

for *k* ∈ **Z***N*.

TLFeBOOK

*N* ] *kn*=

exp*N*−1∑[*j* 2*π n*=0

*N* ] *rn*= 1

*N (k* − *r)n*]}

= 1

*N*

*N*

*N*

*N k(n* + *mN)*]

= exp[*j* 2*π*

*N*−1∑*n*=0

*N*−1∑*k*=0

*N*−1∑*k*=0

{*N*−1∑*k*=0

*Xk*

*N (k* − *r)n*]

=

*N*

{*N*−1∑*n*=0

*N*−1∑*k*=0

*Xk* exp[*j* 2*π*

exp[*j* 2*π*

*Xk* exp[*j* 2*π*

*N rn*]exp[*j* 2*π*

*N kn*]}exp[−*j* 2*π*

THE DISCRETE FOURIER SERIES (DFS) **27**

In (1.77) we see *xn*+*mN* = *xn* for all *m* ∈ **Z**. Thus, *(xn)* in (1.77) is *N*-periodic, and so we have ̃*xn* = 1

*N* ]

*kn*with *Xk* given by (1.79). Equation (1.77) is the *discrete Fourier series* (DFS) expansion for an *N*-periodic complex- valued sequence ̃*x* such as in (1.71). The *DFS coefficients* are given by (1.79). However, to only it is common practice considering the vector to consider *x* ∈ **C***N*. In only this ̃*x*case *n* for the *n* ∈ vector **Z***N*, which *X* ∈ **C**is *N* equivalent given by (1.79) is now called the *discrete Fourier transform* (DFT) of the vector *x*, and the expression in (1.77) is the *inverse DFT* (IDFT) of the vector *X*. We observe that the DFT, and the IDFT can be concisely expressed in matrix form, where we define the *DFT matrix*

*F* =

[exp(−*j* 2*π*

∈ **C***N*×*N, (*1*.*80*)*

and we see from (1.77) that the symmetry of that *F* −1 *F* (i.e., *F* = = *F N* 1

*T F* ) ∗ means (*IDFT* that *matrix*). either Thus, *k X* = *Fx*. We remark or *n* in (1.80) may be interpreted as row or column indices.

The DFT has a long history, and its invention is now attributed to Gauss [10]. The DFT is of central importance to numerical computing generally, but has particularly great significance in digital signal processing as it represents a numerical approximation to the Fourier transform, and it can also be used to efficiently implement digital filtering operations via so-called *fast Fourier trans- form* (FFT) algorithms. The construction of FFT algorithms to efficiently compute *X* = *Fx* (and *x* = *F* −1*X*) is rather involved, and not within the scope of the present book. Simply note that the *direct* computation of the matrix-vector product *X* = *Fx* needs *N*2 complex multiplications and *N(N* − 1*)* complex additions. For *N* = 2*p* (*p* ∈ {1*,*2*,*3*,...*}), which is called the *radix-2 case*, the algorithm of Coo- ley and Tukey [11] reduces the number of operations to something proportional to *N* approach log2 *N*, when which *N* is is a large substantial enough. savings Essentially, compared the method to *N*2 operations in Ref. 11 with the direct implicitly fac- tors *F* according to *F* = *FpFp*−1 ···*F*1, where the matrix factors *Fk* ∈ **C***N*×*N* are *sparse* (i.e., contain many zero-valued entries). Note that multiplication by zero is not implemented in either hardware or software and so does not represent a compu- tational cost in the practical implementation of the FFT algorithm. It is noteworthy that the algorithm of Ref. 11 also has a long history dating back to the work of Gauss, as noted by Heideman et al. [10]. It is also important to mention that fast algorithms exist for all possible *N* = 2*p* [4]. The following example suggests one of the important applications of the DFT/DFS.

**Example 1.22** *N*

Suppose that *xn* = *Aejθn* with *θ* = 2*π*

*N m* for *m* = 1*,*2*,...,*

2 − 1 (*N* is assumed to be even here). From (1.79) using (1.75)

*Xk* = *ANδm*−*k. (*1*.*81*)*TLFeBOOK

*N*

∑*N*−1 *k*=0 *Xk* exp[*j* 2*π*

*N kn*)]*k,n*∈**Z***N*

*N m*. The DFT/DFS is therefore quite use- ful in detecting “sinusoids” (also sometimes called “tone detection”). This makes the DFT/DFS useful in such applications as narrowband radar and sonar signal detection.

Can you explain the necessity (or, at least, the desirability) of the second equality in Eq. (1.82)?

**APPENDIX 1.A COMPLEX ARITHMETIC**

Here we summarize the most important facts about arithmetic with complex num- bers *z* ∈ **C** (set of complex numbers). You shall find this material very useful in electric circuits, as well as in the present book.

Complex numbers may be represented in two ways: (1) *Cartesian (rectangular) form* or (2) *polar form*. First we consider the Cartesian form.

In this case *z* ∈ **C** has the form *z* = *x* + *jy*, where *x,y* ∈ **R** (set of real num- bers), and *j* = √

**28** FUNCTIONAL ANALYSIS IDEAS

Now suppose instead that *xn* = *Ae*−*jθn*, so similarly

*Xk* = *A*

exp*N*−1∑[−*j* 2*π n*=0

*N* ]

*n(N* − *m* − *k)*= *ANδN*−*m*−*k.* (1.82)

Thus, if now *xn* = *A*cos*(θn)* = must have

1 2*A*[*ejθn* + *e*−*jθn*], then from (1.81) and (1.82), we

*Xk* = 1

2*AN*[*δm*−*k* + *δN*−*m*−*k*]*. (*1*.*83*)*

We observe that *Xk* = 0 for all *k* = *m, N* − *m*, but that

*Xm* = 1

2*AN.*

Thus, *Xk* the frequency is nonzero only for indices *k* = *m* and *k* = *N* − *m* corresponding to

of *(xn)*, which is *θ* = 2*π*

−1. The complex conjugate of *z* is defined to be *z*∗ = *x* − *jy* (so *j*∗ = −*j*).

Suppose that *z*1 = *x*1 + *jy*1 and *z*2 = *x*2 + *jy*2 are two complex numbers. Addi- tion and subtraction are defined as

*z*1 ± *z*2 = *(x*1 ± *x*2*)* + *j (y*1 ± *y*2*)*

TLFeBOOK

*N* ] *n(m* + *k)*= *A*

exp*N*−1∑[*j* 2*π n*=0

2*AN,* and *XN*−*m* = 1

COMPLEX ARITHMETIC **29**

[e.g., *(*1 + 2*j)* + *(*3 − 5*j)* = 4 − 3*j*, and *(*1 + 2*j)* − *(*3 − 5*j)* = −2 + 7*j*]. Using *j*2 = −1, the product of *z*1 and *z*2 is

*z*1*z*2 = *(x*1 + *jy*1*)(x*2 + *jy*2*)*

= *x*1*x*2 + *j*2*y*1*y*2 + *jy*1*x*2 + *jx*1*y*2 = *(x*1*x*2 − *y*1*y*2*)* + *j (x*1*y*2 + *x*2*y*1*).*

We note that

so |*z*| = √

*x*2 + *y*2 *zz*∗ = *(x* + *jy)(x* − *jy)* = *x*2 + *y*2 = |*z*|2*,*

defines the *magnitude of z*. For example, *(*1 + 2*j)(*3 − 5*j)* = 13 + *j*. The quotient of *z*1 and *z*2 is defined to be

*z*1

*,*

where the last equality is *z*1*/z*2 in Cartesian form.

Now we may consider polar form representations. For *z* = *x* + *jy*, we may regard *x* and *y* as the *x* and *y* coordinates (respectively) of a point in the Cartesian plane (sometimes denoted **R**2).10 We may therefore express these coordinates in polar form; thus, for any *x* and *y* we can write

*x* = *r* cos*θ, y* = *r* sin*θ,*

where *r* ≥ 0, and *θ* ∈ [0*,*2*π)*, or *θ* ∈ *(*−*π,π*]. We observe that

*x*2 + *y*2 = *r*2*(*cos2 *θ* + sin2 *θ)* = *r*2*,*

so |*z*| = *r*.

Now recall the following Maclaurin series expansions (considered in greater depth in Chapter 3):

sin*x* =

∑∞*n*=1*(*−1*)n*−1 *x*2*n*−1

*(n* − 1*)*!

10This suggests that *z* may be equivalently represented by the column vector [*xy*]*T* . The vector inter- pretation of complex numbers can be quite useful.

TLFeBOOK

*z*2 = *z*1*z*∗2

*x*2 2+ *y*2 2= *(x*1*x*2 + *y*1*y*2*)* + *j (x*2*y*1 − *x*1*y*2*)*

*x*2 2+ *y*2 2= *x*1*x*2 + *y*1*y*2

*(*2*n* − 1*)*!

cos*x* =

∑∞*n*=1*(*−1*)n*−1 *x*2*n*−2

*z*2*z*∗2 = *(x*1 + *jy*1*)(x*2 − *jy*2*)*

*(*2*n* − 2*)*!

*ex* =

∑∞*n*=1

*xn*−1

*x*22 + *y*22

+ *j x*2*y*1 − *x*1*y*2

*x*22 + *y*22

]*,*

where we have split the summation into terms involving even *n* and odd *n*. Thus, continuing*ejx* =

∑∞[*j*2*n*−2*x*2*n*−2

*n*=1

*(*2*n* − 1*)*!

*(j*2*n*−2 = *(j*2*)n*−1 = *(*−1*)n*−1*)*

= cos*x* + *j* sin*x.*

Thus, *ejx* = cos*x* + *j* sin*x*. This is justification for *Euler’s identity* in (1.49). Addi- tionally, since *e*−*jx* = cos*x* − *j* sin*x*, we have

*ejx* + *e*−*jx* = 2 cos*x, ejx* − *e*−*jx* = 2*j* sin*x.*

These immediately imply that

sin*x* = *ejx* − *e*−*jx*

2 *.*

These identities allow for the conversion of expressions involving trig(onometric) functions into expressions involving exponentials, and vice versa. The necessity to do this is frequent. For this reason, they should be memorized, or else you should remember how to derive them “on the spot” when necessary.

Now observe that

*rejθ* = *r* cos*θ* + *jr* sin*θ,*

so that if *z* = *x* + *jy*, then, because there exist *r* and *θ* such that *x* = *r* cos*θ* and *y* = *r* sin*θ*, we may immediately write*z* = *rejθ.*

TLFeBOOK

**30** FUNCTIONAL ANALYSIS IDEAS

These series converge for −∞ *<x<* ∞. Observe the following:

*ejx* =

∑∞*n*=1

*(jx)n*−1

]

=

*(*2*n* − 1*)*!]

*(jj*2*n*−2 = *j*2*n*−1*)*

=

∑∞[ *j*2*n*−2 *x*2*n*−2 *n*=1 ∑∞*n*=1*(*−1*)n*−1 *x*2*n*−2

*(n* − 1*)*! =

*(*2*n* − 2*)*! + *j*2*n*−1*x*2*n*−1

*(*2*n* − 2*)*! + *j x*2*n*−1

*(*2*n* − 2*)*! + *j*

2*j ,* cos*x* = *ejx* + *e*−*jx*

∞∑*n*=1

[ *(jx)(*2*n*−1*)*−1

*(*2*n* − 1*)*!

[*(*2*n* − 1*)* − 1]! + *(jx)(*2*n*−1*)*

∞∑*n*=1*(*−1*)n*−1 *x*2*n*−1

[2*n* − 1]!

ELEMENTARY LOGIC **31**

This is *z* in polar form. For example (assuming that *θ* is in radians)

1 + *j* = √

2*e*3*πj/*4*,* 1 − *j* = √

2*e*−3*πj/*4*.*

It can sometimes be useful to observe that

*j* = *ejπ/*2*,* −*j* = *e*−*jπ/*2*,* and − 1 = *e*±*jπ.*

If *z*1 = *r*1*ejθ*1, and *z*2 = *r*2*ejθ*2, then

*z*1*z*2 = *r*1*r*2*ej (θ*1+*θ*2*), z*1

*r*2*ej (θ*1−*θ*2*).*

In other words, multiplication and division of complex numbers is very easy when they are expressed in polar form.

Finally, some terminology. For *z* = *x* + *jy*, we call *x* the *real part* of *z*, and we call *y* the *imaginary part* of *z*. The notation is

*x* = Re [*z*]*, y* = Im [*z*]*.*

That is, *z* = Re [*z*] + *j* Im [*z*].

**APPENDIX 1.B ELEMENTARY LOGIC**

Here we summarize the basic language and ideas associated with elementary logic as some of what is found here appears in later sections and chapters of this book. The concepts found here appear often in mathematics and engineering literature.

Consider two mathematical statements represented as *P* and *Q*. Each statement may be either true or false. Suppose that we know that if *P* is true, then *Q* is certainly true (allowing the possibility that *Q* is true even if *P* is false). Then we say that *P* implies *Q*, or *Q* is implied by *P*, or *P* is a *sufficient condition* for *Q*, or symbolically

*P* ⇒ *Q* or *Q* ⇐ *P.*

Suppose that if *P* is false, then *Q* is certainly false (allowing the possibility that *Q* may be false even if *P* is true). Then we say that *P* is implied by *Q*, or *Q* implies *P*, or *P* is a *necessary condition* for *Q*, or

*P* ⇐ *Q* or *Q* ⇒ *P.*

Now suppose that if *P* is true, then *Q* is certainly true, and if *P* is false, then *Q* is certainly false. In other words, *P* and *Q* are either both true or both false. Then we say that *P* implies and is implied by *Q*, or *P* is a *necessary and sufficient*TLFeBOOK

2*ejπ/*4*,* −1 + *j* = √

2*e*−*jπ/*4*,* −1 − *j* = √

*z*2 = *r*1

**32** FUNCTIONAL ANALYSIS IDEAS

*condition* for *Q*, or *P* and *Q* are *logically equivalent*, or *P if and only if Q*, or symbolically

*P* ⇔ *Q.*

A common abbreviation for “if and only if” is *iff*.

The *logical contrary* of the statement *P* is called “not *P*.” It is often denoted by either

*P* or ∼ *P*. This is the statement that is true if *P* is false, or false if *P* is true. For example, if *P* is the statement “*x >* 1,” then ∼ *P* is the statement “*x* ≤ 1.” If *P* is the statement “*f (x)* = 0 for all *x* ∈ **R**,” then ∼ *P* is the statement “there is at least one *x* ∈ **R** for which *f (x)* = 0.” We may write

*x*4 − 5*x*2 + 4 = 0 ⇐ *x* = 1 or *x* = 2*,*

but the converse is not true because *x*4 − 5*x*2 + 4 = 0 is a quartic equation pos- sessing four possible solutions. We may write

*x* = 3 ⇒ *x*2 = 3*x,*

but we cannot say *x*2 = 3*x* ⇒ *x* = 3 because *x* = 0 is also possible.

Finally, we observe that

*P* ⇒ *Q* is equivalent to ∼ *P* ⇐∼ *Q,*

*P* ⇐ *Q* is equivalent to ∼ *P* ⇒∼ *Q,*

*P* ⇔ *Q* is equivalent to ∼ *P* ⇔∼ *Q*;

that is, taking logical contraries reverses the directions of implication arrows.

**REFERENCES**

1. E. Kreyszig, *Introductory Functional Analysis with Applications*, Wiley, New

York, 1978. 2. A. P. Hillman and G. L. Alexanderson, *A First Undergraduate Course in Abstract Alge-*

*bra*, 3rd ed., Wadsworth, Belmont, CA, 1983. 3. R. B. J. T. Allenby, *Rings, Fields and Groups: An Introduction to Abstract Algebra*,

Edward Arnold, London, UK, 1983. 4. R. E. Blahut, *Fast Algorithms for Digital Signal Processing*, Addison-Wesley, Reading,

MA, 1985. 5. C. K. Chui, *Wavelets: A Mathematical Tool for Signal Analysis*. SIAM, Philadelphia,

PA, 1997. 6. R. E. Ziemer and W. H. Tranter, *Principles of Communications: Systems, Modulation,*

*and Noise*, 3rd ed., Houghton Mifflin, Boston, MA, 1990. 7. I. Daubechies, “Orthonormal Bases of Compactly Supported Wavelets,” *Commun. Pure*

*Appl. Math.* **41**, 909–996 (1988).

TLFeBOOK

PROBLEMS **33**

8. G. P. Tolstov, *Fourier Series* (transl. from Russian by R. A. Silverman), Dover Publi-

cations, New York, 1962. 9. E. Kreyszig, *Advanced Engineering Mathematics*, 4th ed., Wiley, New York, 1979. 10. M. T. Heideman, D. H. Johnson and C. S. Burrus, “Gauss and the History of the Fast

Fourier Transform,” *IEEE ASSP Mag.* **1**, 14–21 (Oct. 1984). 11. J. W. Cooley and J. W. Tukey, “An Algorithm for the Machine Calculation of Complex

Fourier Series,” *Math. Comput.*, **19**, 297–301 (April 1965).

**PROBLEMS**

**1.1. (a)** Find *a,b* ∈ **R** in

1 + 2*j* −3 − *j* = *a* + *bj.*

**(b)** Find *r, θ* ∈ **R** in

−3 + *j* = *rejθ*

(Of course, choose *r >* 0, and *θ* ∈ *(*−*π,π*].)

**1.2.** Solve for *x* ∈ **C** in the quadratic equation

*x*2 − 2*r* cos*θx* + *r*2 = 0*.*

Here *r* ≥ 0, and *θ* ∈ *(*−*π,π*]. Express your solution in polar form.

**1.3.** Let *θ*, and *φ* be arbitrary angles (so *θ,φ* ∈ **R**). Show that

*(*cos*θ* + *j* sin*θ)(*cos*φ* + *j* sin*φ)* = cos*(θ* + *φ)* + *j* sin*(θ* + *φ).*

**1.4.** Prove the following theorem. Suppose *z* ∈ **C** such that

*z* = *r* cos*θ* + *jr* sin*θ*

for which *r* = |*z*| *>* 0, and *θ* ∈ *(*−*π,π*]. Let *n* ∈ {1*,*2*,*3*,...*} (i.e., *n* is a positive integer). The *n* different *n*th roots of *z* are given by

*r*1*/n* [cos(*θ* + 2*πk*

)]*,*

for *k* = 0*,*1*,*2*,...,n* − 1.

**1.5.** State whether the following are true or false:

**(a)** |*x*| *<* 2 ⇒ *x <* 2 **(b)** |*x*| *<* 3 ⇐ 0 *<x<* 3

TLFeBOOK

*n*

)

+ *j* sin(*θ* + 2*πk*

*n*

**34** FUNCTIONAL ANALYSIS IDEAS

**(c)** *x* − *y >* 0 ⇒ *x>y>* 0 **(d)** *xy* = 0 ⇒ *x* = 0 and *y* = 0 **(e)** *x* = 10 ⇐ *x*2 = 10*x*

Explain your answer in all cases.

**1.6.** Consider the function

*f (x)* =

{ −*x*2 + 2*x* + 1*, x*2 − 2*x* + 3

2*,* 0 ≤ *x <* 1

1 *< x* ≤ 2 *.*

Find

sup *x*∈[0*,*2]*f (x), x*∈[0*,*2]inf *f (x).*

**1.7.** Suppose that we have the following polynomials in the indeterminate *x*:

*a(x)* =

∑*nakxk, b(x)* =

∑*mbjxj. k*=0*j*=0Prove that

*c(x)* = *a(x)b(x)* =

*n*+*m*∑*l*=0

*clxl,*

where

*cl* =

∑*nk*=0

*akbl*−*k.*

[*Comment:* This is really asking us to prove that discrete convolution is mathematically equivalent to polynomial multiplication. It explains why the MATLAB routine for multiplying polynomials is called *conv.* Discrete con- volution is a fundamental operation in digital signal processing, and is an instance of something called *finite impulse response* (FIR) filtering. You will find it useful to note that *ak* = 0 for *k <* 0, and *k>n*, and that *bj* = 0 for *j <* 0, and *j>m*. Knowing this allows you to manipulate the summation limits to achieve the desired result.]

**1.8.** Recall Example 1.5. Suppose that *xk* = 2*k*+1, and that *yk* = 1 for *k* ∈ **Z**+.

Find the sum of series. For example, the ∑series *Nk*=0 *αd(x,y)*. *k* = (*Hint:* Recall the 1−*α*1−*α N*+1

if *α* = 1.)

theory of geometric

**1.9.** Prove that if *x* = 1, then

*Sn* =

∑*nk*=1

*kxk*−1

TLFeBOOK

PROBLEMS **35**

is given by

*Sn* = 1 − *(n* + 1*)xn* + *(*1 − *x)*2 *nxn*+1

*.*

What *Sn* − *xS*is *n* the = 1 formula + *x* + *x*for 2 +···+ *Sn* when *x* = 1 ? (*Hint:* Begin by showing that

*xn*−1 − *nxn*.)

**1.10.** Recall Example 1.1. Prove that *d(x,y)* in (1.5) satisfies all the axioms for a

metric.

**1.11.** Recall Example 1.18. Prove that 〈*x,y*〉 in (1.36) satisfies all the axioms for

an inner product.

**1.12.** By direct calculation, show that if *x,y,z* are elements from an inner product

space, then||*z* − *x*||2 + ||*z* − *y*||2 = 1

2*(x* + *y)*||2

(Appolonius’ identity).

**1.13.** Suppose *x,y* ∈ **R**3 (three-dimensional Euclidean space) such that

*x* = [1 1 1]*T , y* = [1 − 1 1]*T .*

Find all vectors *z* ∈ **R**3 such that 〈*x,z*〉=〈*y,z*〉 = 0.

**1.14.** The complex Fourier series expansion method as described is for *f* ∈ *L*2*(*0*,*2*π)*. Find the complex Fourier series expansion for *f* ∈ *L*2*(*0*,T)*, where 0 *<T <* ∞ (i.e., the interval on which *f* is defined is now of arbitrary length).

**1.15.** Consider again the complex Fourier series expansion for *f* ∈ *L*2*(*0*,*2*π)*. Specifically, consider Eq. (1.44). If *f(t)* ∈ **R** for all *t* ∈ *(*0*,*2*π)*, then show that that *f*for *n* = suitable *f* −*n*∗. [The sequence *(fn)* is *conjugate an,bn* ∈ **R** (all *n*) we have

*symmetric*.] Use this to show

∑∞*n*=−∞*fnejnt* = *a*0 +

∑∞*n*=1[*an* cos*(nt)* + *bn* sin*(nt)*]*.*

How are the coefficients *an* and *bn* related to *fn* ? (Be very specific. There is a simple formula.)

**1.16. (a)** Suppose that *f* ∈ *L*2*(*0*,*2*π)*, and that specifically

*f(t)* =

{ 1*, j,* 0 *<t<π*

*π* ≤ *t <* 2*π .*

Find *fn* in Eq. (1.44) using (1.45); that is, find the complex Fourier series expansion for *f(t)*. Make sure that you appropriately simplify your series expansion.

TLFeBOOK

2||*x* − *y*||2 + 2||*z* − 1

**36** FUNCTIONAL ANALYSIS IDEAS

**(b)** Show how to use the result in Example 1.20 to find the complex Fourier

series expansion for *f(t)* in (a).

**1.17.** This problem is about finding the Fourier series expansion for the wave- form at the output of a full-wave rectifier circuit. This circuit is used in AC/DC (alternating/direct-current) converters. Knowledge of the Fourier series expansion gives information to aid in the design of such converters.

**(a)** Find the complex Fourier series expansion of

*f(t)* =

∣∣∣∣sin(2*π*

)*.*

**(b)** Find the sequences *(an)*, and *(bn)* in

*f(t)* = *a*0 +

∑∞*n*=1[*an* cos(2*πn*

*T t*)]

for *f(t)* in (a). You need to consider how *T* is related to *T*1. **1.18.** Recall the definitions of the Haar scaling function and Haar wavelet in

Eqs. (1.40) and (1.41), respectively. *ψk,n(t)* = 2*k/*2*ψ(*2*kt* − *n)*. Recall that inner product for *L*2*(***R***)*.

Define 〈*f(t),g(t)*〉 *φk,n(t)* = ∫ = −∞ ∞2*k/*2*f(t)gφ(*2*k*∗*t (t)dt* − *n)*, is and the

**(a)** Sketch *φk,n(t)*, and *ψk,n(t)*. **(b)** Evaluate the integrals ∫ ∞−∞ *φk,n*2*(t)dt,* and

∫ ∞−∞ *ψk,n*2*(t)dt.* **(c)** Prove that

〈*φk,n(t),φk,m(t)*〉 = *δn*−*m.*

**1.19.** Prove the following version of the Schwarz inequality. For all *x,y* ∈ *X* (inner

product space)

| Re [〈*x,y*〉]| ≤ ||*x*|| ||*y*||

with equality iff *y* = *βx*, and *β* ∈ **R** is a constant.

[*Hint:* The proof of this one is not quite like that of Theorem 1.1. Consider 〈*αx* + *y,αx* + *y*〉 ≥ 0 with *α* ∈ **R**. The inner product is to be viewed as a quadratic in *α*.]

**1.20.** The following result is associated with the proof of the uncertainty principle

for analog signals.

Prove that for *f(t)* ∈ *L*2*(***R***)* such that |*t*|*f(t)* ∈ *L*2*(***R***)* and *f (*1*)(t)* = *df(t)/dt* ∈ *L*2*(***R***)*, we have the inequality

∣∣∣∣Re[∫ ∞−∞ *tf(t)*[*f (*1*)(t)*]∗ *dt*]∣∣∣∣2

≤

[∫ ∞−∞ |*tf(t)*|2 *dt*] [∫ ∞−∞ |*f (*1*)(t)*|2 *dt*]*.*

TLFeBOOK

*T*1 *t*)∣∣∣∣ ∈ *L*2 (0*, T*1

*T t*)

+ *bn* sin(2*πn*

2

PROBLEMS **37**

**1.21.** Suppose *ek* = [*ek,*0 *ek,*1 ··· *ek,N*−2 *ek,N*−1]*T* ∈ **C***N*, where

*ek,n* = exp[*j* 2*π*

*N kn*] and *k* ∈ **Z***N*. If 〈*ek,er*〉 = *Nδk*−*r*. *x,y* ∈ **C***N* recall that 〈*x,y*〉 Thus, *B* = {*ek*|*k* ∈ **Z***N*} is an = orthogonal ∑*N*−1 *k*=0 *xkyk*∗basis . Prove that for **C***N*. Set *B* is important in digital signal processing because it is used to define the *discrete Fourier transform*.

TLFeBOOK

**2 Number Representations**

**2.1 INTRODUCTION**

In this chapter we consider how numbers are represented on a computer largely with respect to the errors that occur when basic arithmetical operations are performed on them. We are most interested here in so-called *rounding errors* (also called *roundoff errors*). Floating-point computation is emphasized. This is due to the fact that most numerical computation is performed with floating-point numbers, especially when numerical methods are implemented in high-level programming languages such as C, Pascal, FORTRAN, and C++. However, an understanding of floating-point requires some understanding of fixed-point schemes first, and so this case will be considered initially. In addition, fixed-point schemes are used to represent integer data (i.e., subsets of **Z**), and so the fixed-point representation is important in its own right. For example, the exponent in a floating-point number is an integer.

The reader is assumed to be familiar with how integers are represented, and how they are manipulated with digital hardware from a typical introductory dig- ital electronics book or course. However, if this is not so, then some review of this topic appears in Appendix 2.A. The reader should study this material *now* if necessary.

Our main (historical) reference text for the material of this chapter is Wilkin- son [1]. However, Golub and Van Loan [4, Section 2.4] is also a good refer- ence. Golub and Van Loan [4] base their conventions and results in turn on Forsythe et al. [5].

**2.2 FIXED-POINT REPRESENTATIONS**

We now consider fixed-point fractions. We must do so because the mantissa in a floating-point number is a fixed-point fraction.

We assume that fractions are *t* + 1 digits long. If the number is in binary, then we usually say “*t* + 1 bits” long instead. Suppose, then, that *x* is a *(t* + 1*)*-bit fraction. We shall write it in the form

*(x)*2 = *x*0*.x*1*x*2 ···*xt*−1*xt (xk* ∈ {0*,*1}*). (*2*.*1*)*

*An Introduction to Numerical Analysis for Electrical and Computer Engineers*, by C.J. Zarowski ISBN 0-471-46737-5 c 2004 John Wiley & Sons, Inc.

**38**

TLFeBOOK

FIXED-POINT REPRESENTATIONS **39**

The notation *(x)*2 means that *x* is in base-2 (binary) form. More generally, *(x)r* means that *x* is expressed as a base-*r* number (e.g., if *r* = 10 this would be the decimal representation). We use this notation to emphasize which base we are working with when necessary (e.g., to avoid ambiguity). We shall assume that (2.1) is a two’s complement fraction. Thus, bit *x*0 is the *sign bit*. If this bit is 1, we interpret the fraction to be negative; otherwise, it is nonnegative. For example, *(*1*.*1011*)*2 = *(*−0*.*3125*)*10. [To take the two’s complement of *(*1*.*1011*)*2, first com- plement every bit, and then add *(*0*.*0001*)*2. This gives *(*0*.*0101*)*2 = *(*0*.*3125*)*10.] In general, for the case of a *(t* + 1*)*-bit two’s complement fraction, we obtain

−1 ≤ *x* ≤ 1 − 2−*t. (*2*.*2*)*

In fact

*(*−1*)*10 = *(*1*.* 00*...*00 }

}

*)*2*. (*2*.*3*) t bits*

We may regard (2.2) as specifying the *dynamic range* of the *(t* + 1*)*-bit two’s complement fraction representation scheme. Numbers beyond this range are not represented. Justification of (2.2) [and (2.3)] would follow the argument for the conversion of two’s complement integers into decimal integers that is considered in Appendix 2.A.

Consider the set {*x* ∈ **R**| − 1 ≤ *x* ≤ 1 − 2−*t*}. In other words, *x* is a real number within fraction. the For limits example, imposed *x* = by √

(2.2), 2 − 1 but is in it is the not range necessarily (2.2), equal to a *(t* + 1*)*-bit but it is an irrational number, and so does not possess an exact *(t* + 1*)*-bit representation. We may choose to approximate such a number with *t* + 1 bits. Denote the *(t* + 1*)*-bit approximation of *x* as *Q*[*x*]. For example, *Q*[*x*] might be the approximation to *x* obtained by selecting an element from set

*B* = {*bn* = −1 + 2−*tn*|*n* = 0*,*1*,...,*2*t*+1 − 1} ⊂ **R** *(*2*.*4*)*

that is the closest to *x*, where distance is measured by the metric in Example 1.1. Note that each number in *B* is representable as a *(t* + 1*)*-bit fraction. In fact, *B* is the entire set of *(t* + 1*)*-bit two’s complement fractions. Formally, our approximation is given by

*Q*[*x*] = argmin

*n*∈{0*,*1*,...,*2*t*+1−1}|*x* − *bn*|*. (*2*.*5*)*

The notation “argmin” {0*,*1*,...,*2*t*+1 − 1} means “let *Q*[*x*] be the *bn* that minimizes |*x* − *bn*|.” In other for the *n* in the set words, we choose the *arg*ument *bn* that *min*imizes the distance to *x*. Some reflection (and perhaps con- sidering some simple examples for small *t*) will lead the reader to conclude that the error in this approximation satisfies

|*x* − *Q*[*x*]| ≤ 2−*(t*+1*). (*2*.*6*)*TLFeBOOK

*)*{{

}

2*, (*1 − 2−*t)*10 = *(*0*.* 11*...*11 } *t bits*

{{

**40** NUMBER REPRESENTATIONS

The error *ε* = *x* − *Q*[*x*] is called *quantization error*. Equation (2.6) is an upper bound on the size (norm) of this error. In fact, in the notation of Chapter 1, if ||*x*|| = |*x*|, then ||*ε*|| = ||*x* − *Q*[*x*]|| ≤ 2−*(t*+1*)*. We remark that our quantization method is not unique. There are many other methods, and these will generally lead to different bounds.

When we represent the numbers in a computational problem on a computer, we see that errors due to quantization can arise even before we perform any oper- ations on the numbers at all. However, errors will also arise in the course of performing basic arithmetic operations on the numbers. We consider the sources of these now.

If *x,y* are coded as in (2.1), then their sum might not be in the range specified by (2.2). This can happen only if *x* and *y* are either both positive or both negative. Such a condition is *fixed-point overflow*. (A test for overflow in two’s complement integer addition appears in Appendix 2.A, and it is easy to modify it for the problem of overflow testing in the addition of fractions.) Similarly, overflow can occur when a negative number is subtracted from a positive number, or if a positive number is subtracted from a negative number. A test for this case is possible, too, but we omit the details. Other than the problem of overflow, no errors can occur in the addition or subtraction of fractions.

With respect to fractions, rounding error arises only when we perform multipli- cation and division. We now consider errors in these operations.

We will deal with multiplication first. Suppose that *x* and *y* are represented according to (2.1). Suppose also that *x*0 = *y*0 = 0. It is easy to see that the product of *x* and *y* is given by

*p* = *xy* =

( ∑*tk*=0

*xk*2−*k*)( ∑*tn*=0*yn*2−*n*) = *(x*0 + *x*12−1 +···+ *xt*2−*t)(y*0 + *y*12−1 +···+ *yt*2−*t)* = *x*0*y*0 + *(x*0*y*1 + *x*1*y*0*)*2−1 +···+ *xtyt*2−2*t.* (2.7)

This implies that the product is a *(*2*t* + 1*)*-bit number. If we allow *x* and *y* to be either positive or negative, then the product will also be 2*t* + 1 bits long. Of course, one of these bits is the sign bit. If we had to multiply several numbers together, we see that the product wordsize would grow in some proportion to the number of factors in the product. The growth is clearly very rapid, and no practical computer could sustain this for very long. We are therefore forced in general to *round off* the product *p* back down to a number that is only *t* + 1 bits long. Obviously, this will introduce an error.

How should the rounding be done? There is more than one possibility (just as there is more than one way to quantize). Wilkinson [1, p. 4] suggests the following. Since the product *p* has the form

*(p)*2 = *p*0*.p*1*p*2 ···*pt*−1*ptpt*+1 ···*p*2*t (pk* ∈ {0*,*1}*) (*2*.*8*)*TLFeBOOK

FIXED-POINT REPRESENTATIONS **41**

we may add 2−*(t*+1*)* to this product, and then simply discard the last *t* bits of the resulting sum (i.e., the bits indexed *t* + 1 to 2*t*). For example, suppose *t* = 4, and consider

0*.*00111111 = *p*

+0*.*00001000

= 2−5

0*.*01000111

Thus, the rounded product is manner is not higher in magnitude rounding operation to be *fx*[*p*] *(*0*.*0100*)*= *fx*[*xy*], than 2. 1

2The 2so −*t* error involved = 2−*(t*+1*)*. then

in Define rounding in this the result of the

|*p* − *fx*[*p*]| ≤ 1

22−*t. (*2*.*9*)*

[For the previous example, *p* = *(*0*.*00111111*)*2, and so *fx*[*p*] = *(*0*.*0100*)*2.] It is natural to measure the sizes of errors in the same way as we measured the size of quantization errors earlier. Thus, (2.9) is an upper bound on the size of the error due to rounding a product. As with quantization, other rounding methods would generally give other bounds. We remark that Wilkinson’s suggestion amounts to “ordinary rounding.”

Finally, we consider fixed-point division. Again, suppose that *x* and *y* are rep- resented as in (2.1), and consider the quotient *q* = *x/y*. Obviously, we must avoid *y* = 0. Also, the quotient will not be in the permitted range given by (2.2) unless |*y*|≥|*x*|. This implies that when fixed-point division is implemented either the *dividend x* or the *divisor y* need to be scaled to meet this restriction. Scaling is multiplication by a power of 2, and so should be implemented to reduce rounding error. We do not consider the specifics of how to achieve this. Another problem is that *x/y* may require an infinite number of bits to represent it. For example, suppose

*q* = *(*0*.*0010*)*2

01*)*2*.*

The bar over 01 denotes the fact that this pattern repeats indefinitely. Fortunately, the same recipe for the rounding of products considered above may also be used to round quotients. If *fx*[*q*] again denotes the result of applying this procedure to *q*, then

|*q* − *fx*[*q*]| ≤ 1

22−*t. (*2*.*10*)*

We see that the difficulties associated with division in fixed-point representations means that fixed-point arithmetic should, if possible, not be used to implement algorithms that require division. This forces us to either (1) employ floating-point representations or (2) develop algorithms that solve the problem without the need for division operations.

Both strategies are employed in practice. Usually choice 1 is easier.

TLFeBOOK

*(*0*.*0110*)*2 = *(*0*.*125*)*10

*(*0*.*375*)*10 =

(1

3)10 = *(*0*.*

**42** NUMBER REPRESENTATIONS

**2.3 FLOATING-POINT REPRESENTATIONS**

In the previous section we have seen that fixed-point numbers are of very limited dynamic range. This poses a major problem in employing them in engineering computations since obviously we desire to work with numbers far beyond the range in (2.2). Floating-point representations provide the definitive solution to this problem. We remark (in passing) that the basic organization of a floating-point arithmetic unit [i.e., digital hardware for floating-point addition and subtraction appears in Ref. 2 (see pp. 295–306)]. There is a standard IEEE format for floating- point numbers. We do not consider this standard here, but it is summarized in Ref. 2 (see pp. 304–306). Some of the technical subtleties associated with the IEEE standard are considered by Higham [6].

Following Golub and Van Loan [4, p. 61], the set **F** (subset of **R**) of floating- point numbers consists of numbers of the form

*x* = *x*0*.x*1*x*2 ···*xt*−1*xt* × *re, (*2*.*11*)*

where *x*0 is a sign bit (which means that we can replace *x*0 by ±; this is done in Ref. 4), and *r* is the base of the representation [typically *r* = 2 (binary), or *r* = 10 (decimal); we will emphasize *r* = 2]. Therefore, *xk* ∈ {0*,*1*,...,r* − 2*,r* − 1} for 1 ≤ *k* ≤ *t*. These are the digits (bits if *r* = 2) of the *mantissa*. We therefore see that the mantissa is a fraction. 1 It is important to note that *x*1 = 0, and this has implications with regard to how operations are performed and the resulting rounding errors. We call *e* the *exponent*. This is an integer quantity such that *L* ≤ *e* ≤ *U*. For example, we might represent *e* as an *n*-bit two’s complement integer. We will assume this unless otherwise specified in what follows. This would imply that *(e)*2 = *en*−1*en*−2 ···*e*1*e*0, and so−2*n*−1 ≤ *e* ≤ 2*n*−1 − 1 *(*2*.*12*)*

(see Appendix A for justification). For nonzero *x* ∈ **F**, then

*m* ≤ |*x*| ≤ *M, (*2*.*13a*)*

where

*m* = *rL*−1*, M* = *rU(*1 − *r*−*t). (*2*.*13b*)*

Equation (2.13) gives the *dynamic range* for the floating-point representation. With *r* = 2 we see that the total wordsize for the floating-point number is *t* + *n* + 1 bits. In the absence of rounding errors in a computation, our numbers may initially be from the set

*G* = {*x* ∈ **R**|*m* ≤ |*x*| ≤ *M*}∪{0}*. (*2*.*14*)*

1Including the sign bit the mantissa is (for *r* = 2) *t* + 1 bits long. Frequently in what follows we shall refer to it as being only *t* bits long. This is because we are ignoring the sign bit, which is always understood to be present.

TLFeBOOK

FLOATING-POINT REPRESENTATIONS **43**

This set is analogous to the set {*x* ∈ **R**| − 1 ≤ *x* ≤ 1 − 2−*t*} that we saw in the previous section in our study of fixed-point quantization effects. Again following Golub and Van Loan [4], we may define a mapping (operator) *fl*|*G* → **F**. Here *c* = *fl*[*x*] (*x* ∈ *G*) is obtained by choosing the closest *c* ∈ **F** to *x*. As you might expect, distance is measured using || · || = | · |, as we did in the previous section. Golub and Van Loan call this *rounded arithmetic* [4], and it coincides with the rounding procedure described by Wilkinson [1, pp. 7–11].

Suppose that *x* and *y* are two floating-point numbers (i.e., elements of **F**) and that “*op*” denotes any of the four basic arithmetic operations (addition, subtrac- tion, multiplication, or division). Suppose |*x op y*| */*∈ *G*. This implies that either |*x op y*| *> M* (floating-point *overflow*), or 0 *<* |*x op y*| *< m* (floating-point *under- flow*) has occurred. Under normal circumstances an *arithmetic fault* such as over- flow will not happen unless an unstable procedure is being performed. The issue of “numerical stability” will be considered later. Overflows typically cause runtime error messages to appear. The underflow arithmetic fault occurs when a number arises that is not zero, but is too small to represent in the set **F**. This usually poses less of a problem than overflow. 2 However, as noted before, we are concerned mainly with rounding errors here. If |*x op y*| ∈ *G*, then we assume that the com- puter implementation of *x op y* will be given by *fl*[*x op y*]. In other words, the operator *fl* models rounding effects in floating-point arithmetic operations. We remark that where floating-point arithmetic is concerned, rounding error arises in all four arithmetic operations. This contrasts with fixed-point arithmetic wherein rounding errors arise only in multiplication and division.

It turns out that for the floating-point rounding procedure suggested above

*fl*[*x op y*] = *(x op y)(*1 + *ε), (*2*.*15*)*

where

|*ε*| ≤ 1

2*r*1−*t(*= 2−*t* if *r* = 2*). (*2*.*16*)*

We shall justify this only for the case *r* = 2. Our arguments will follow those of Wilkinson [1, pp. 7–11].

Let us now consider the addition of the base-2 floating-point numbers

*x* = *x*}

0*.x*1 ···*xt* ×2*ex (*2*.*17a*)*

and

*y* = *y*}

0*.y*1 ···*yt* ×2*ey, (*2*.*17b*)*

and we assume that |*x*| *>* |*y*|. (If instead |*y*| *>* |*x*|, then reverse the roles of *x* and *y*.) If *ex* − *ey > t*, then

*fl*[*x* + *y*] = *x. (*2*.*18*)*

2Underflows are simply set to zero on some machines.

TLFeBOOK

} =*my*

} =*mx*

{{

{{

**44** NUMBER REPRESENTATIONS

For example, if *t* = 4, and *x* = 0*.*1001 × 24, and *y* = 0*.*1110 × 2−1, then to add these numbers, we must shift the bits in the mantissa of one of them so that both have the same exponent. If we choose *y* (usually shifting is performed on the smallest number), then *y* = 0*.*00000111 × 24. Therefore, *x* + *y* = 0*.*10010111 × 24, but then *fl*[*x* + *y*] = 0*.*1001 × 24 = *x*. tissa Now *ex* − if *e*instead *y* positions we have to the *ex* right. − *ey* The ≤ *t*, sum we divide *x* + 2*ey*−*exy* by *y* 2is *ex*−*ey* then by calculated shifting its exactly, man-

and requires ≤ 2*t* bits for its representation. The sum is multiplied by a power of 2, using left or right shifts to ensure that the mantissa is properly normalized [recall that for *x* in (2.11) we must have *x*1 = 0]. Of course, the exponent must be modi- fied to account for the shift of the bits in the mantissa. The 2*t*-bit mantissa is then rounded off to *t* bits using *fl*. Because we have |*mx*| + 2*ey*−*ex*|*my*| ≤ 1 + 1 = 2, the largest possible right shift is by one bit position. However, a left shift of up to *t* bit positions might be needed because of the cancellation of bits in the summation process. Let us consider a few examples. We will assume that *t* = 4.

**Example 2.1** Let *x* = 0*.*1001 × 24, and *y* = 0*.*1010 × 21. Thus

0*.*10010000 × 24

+0*.*00010100

× 24

0*.*10100100 × 24

and the sum is rounded to 0*.*1010 × 24 (computed sum).

**Example 2.2** Let *x* = 0*.*1111 × 24, and *y* = 0*.*1010 × 22. Thus

0*.*11110000 × 24

+0*.*00101000

× 24

1*.*00011000 × 24

but 1*.*00011000 × 24 = 0*.*100011000 × 25, and this exact sum is rounded to 0*.*1001 × 25 (computed sum).

**Example 2.3** Let *x* = 0*.*1111 × 2−4, and *y* = −*.*1110 × 2−4. Thus

0*.*11110000 × 2−4

−0*.*11100000

× 2−4

0*.*00010000 × 2−4

but 0*.*00010000 × 2−4 = 0*.*1000 × 2−7, and this exact sum is rounded to 0*.*1000 × 2−7 (computed sum). Here there is much *cancellation* of the bits leading in turn to a large shift of the mantissa of the exact sum to the left. Yet, the computed sum is exact.

TLFeBOOK

FLOATING-POINT REPRESENTATIONS **45**

We observe that the *computed sum* is obtained by computing the *exact sum*, normalizing it so that the mantissa *s*0*.s*1 ···*st*−1*stst*+1 ···*s*2*t* satisfies *s*1 = 1 (i.e., *s*1 = 0), and *exact sum* is number) 1 22−*t*2*es*. Essentially, *then* we round it to *s* = *ms* × 2*es(*= *x* + the error *ε t* places (i.e., we apply *fl*). *y)*, then the rounding error *ε* is due to rounding the mantissa If the *normalized* is such that |*ε* | ≤ (a fixed-point

1 22*es* according ≤ |*s*| *<* 2*es*, to the method used in Section 2.2. Because of the form of *ms*, and so

*fl*[*x* + *y*] = *(x* + *y)(*1 + *ε) (*2*.*19*)*

which is just a special case of (2.15). This expression requires further explanation, however. Observe that

|*s* − *fl*[*s*]|

|*s*|

which is the *relative error*3 due to rounding. Because we have this error is biggest when |*s*| = 1

1 22*es* ≤ |*s*| *<* 2*es*,

|*s*| ≤ 2−*t. (*2*.*20*)*

From (2.19) *fl*[*s*] = *s* + *sε*, so that |*s* − *fl*[*s*]|=|*s*||*ε*|, or |*ε*|=|*s* − *fl*[*s*]|*/*|*s*|. Thus, |*ε*| ≤ 2−*t*, which is (2.16). In other words, |*ε* | is the absolute error, and |*ε*| is the relative error.

Finally, if *x* = 0 or *y* = 0 then no rounding error occurs: *ε* = 0. Subtraction results do not differ from addition. Now consider computing the product of *x* and *y* and *y* = *my* × 2*ey* with *x*1 = 0, and *y*1 = 0 we must in (2.17). Since *x* = *mx* × 2*ex*,

have

1

2 ≤ |*mxmy*| *<* 1*. (*2*.*21*)*

This implies that it may be necessary to normalize the mantissa of the product with a shift to the left, and an appropriate adjustment of the exponent as well. The 2*t*-bit mantissa of the product is rounded to give a *t*-bit mantissa. If *x* = 0, or *y* = 0 (or both *x* and *y* are zero), then the product is zero.

3In general, if *a* is the exact value of some quantity and ˆ*a* is some approximation to *a*, the *absolute error* is ||*a* − ˆ*a*||, while the *relative error* is||*a* ||*a*|| − ˆ*a*||

*(a* = 0*).*

The relative error is usually more meaningful in practice. This is because an error is really “big” or “small” only in relation to the size of the quantity being approximated.

TLFeBOOK

|*s*| = |*s* − *(s* + *ε )*|

2 1

22*es*, so therefore we conclude that

|*s* − *fl*[*s*]|

|*s*| = |*ε* |

|*s*| ≤

1 22−*t*2*es*

**46** NUMBER REPRESENTATIONS

We may consider a few examples. We will suppose *t* = 4. Begin with *x* = 0*.*1010 × 22, and *y* = 0*.*1111 × 21, so then

*xy* = 0*.*10010110 × 23*,*

and so *fl*[*xy*] = 0*.*1001 × 23 (computed product). If now *x* = 0*.*1000 × 24, *y* = 0*.*1000 × 2−1, then, before normalizing the mantissa, we have

*xy* = 0*.*01000000 × 23*,*

and after normalization we have*xy* = 0*.*10000000 × 22

so that *fl*[*xy*] = 0*.*1000 × 22 (computed product). Finally, suppose that *x* = 0*.*1010 × 20, and *y* = 0*.*1010 × 20, so then the unnormalized product is

*xy* = 0*.*01100100 × 20

for which the normalized product is

*xy* = 0*.*11001000 × 2−1*,*

so finally *fl*[*xy*] = 0*.*1101 × 2−1 (computed product).

The application of *fl* to the normalized product will have exactly the same effect as it did in the case of addition (or of subtraction). This may be under- stood by recognizing that a 2*t*-bit mantissa will “look the same” to operator *fl* regardless of how that mantissa was obtained. It therefore immediately follows that

*fl*[*xy*] = *(xy)(*1 + *ε), (*2*.*22*)*

which is another special case of (2.15), and |*ε*| ≤ 2−*t*, which is (2.16) again.

Now consider the quotient *x/y*, for *x* and *y* = 0 in (2.17),

*q* = *x*

*my* × 2*ex*−*ey* = *mq* × 2*eq (*2*.*23*)*

(so *mq* = *mx/my*, and *eq* = *ex* − *ey*). The arithmetic unit in the machine has an accumulator that we assume contains *mx* and which is “double length” in that it is 2*t* bits long. Specifically, this accumulator initially stores *x*0*.x*1 ···*xt* 0···0 } }

. If |*mx*| *>* |*my*| the number in the accumulator is shifted one place to the right, *t bits*

and so *eq* is increased by one (i.e., incremented). The number in the accumulator is then divided by *my* in such a manner as to give a correctly rounded *t*-bit result. This implies that the computed mantissa of the quotient, say, *mq* = *q*0*.q*1 ···*qt*,TLFeBOOK

*y* = *mx* × 2*ex*

*my* × 2*ey* = *mx*

{{

FLOATING-POINT REPRESENTATIONS **47**

satisfies the normalization condition *q*1 = 1, so that must have

1 2 ≤ |*mq*| *<* 1. Once again we

*fl*

[*x*

*y(*1 + *ε) (*2*.*24*)*

such that |*ε*| ≤ 2−*t*. Therefore, (2.15) and (2.16) are now justified for all instances of *op*.We complete this section with a few examples. Suppose *x* = 0*.*1010 × 22, and *y* = 0*.*1100 × 2−2, then

*q* = *x*

0*.*1100 × 24 = 0*.*11010101 × 24

so that *fl*[*q*] = 0*.*1101 × 24 (computed quotient). Now suppose that *x* = 0*.*1110 × 23, and *y* = 0*.*1001 × 2−2, and so

*q* = *x*

0*.*1001 × 26 = 0*.*11000111 × 26

so that *fl*[*q*] = 0*.*1100 × 26 (computed quotient). Thus far we have emphasized *ordinary* tation operator of *fl*, *fl* is we to have use *chopping*. *fl*[*x*] = ±If (∑*x* including the sign bit). Thus, the absolute = *tk*=1 ±*x*(∑error *rounding*, *k*2−*k*∞*k*=1 ) is

× *xk*22but *e* −*k*(chopping ) an × 2alternative *e*, then, *x* implemen- for chopping to *t* + 1 bits

|*ε* |=|*x* − *fl*[*x*]| =



∑∞*k*=*t*+1*xk*2−*k*2*e* ≤ 2*e*

∑∞*k*=*t*+12−*k*

(as *xk* = 1 for all *k>t*), but since ∑∞*k*=*t*+1 2−*k* = 2−*t*, we must have

|*ε* |=|*x* − *fl*[*x*]| ≤ 2−*t*2*e,*

and so the relative error for chopping is

|*ε*| = |*x* − *fl*[*x*]|

22*e* = 2−*t*+1

(because we recall that |*x*| ≥ bigger than the error in rounding, 1

22*e*). but We chopping see that the is somewhat error in chopping easier to is implement.somewhat

TLFeBOOK

0*.*1100 × 2−2 = 0*.*10100000

0*.*1001 × 2−2 = 0*.*01110000

*y* = 0*.*1010 × 22

*y* = 0*.*1110 × 23

0*.*1100 × 2−2 = 0*.*10100000 × 22

0*.*1001 × 2−2 = 0*.*01110000 × 24

|*x*| ≤ 2−*tee*

*y*]

= *x*

1

**48** NUMBER REPRESENTATIONS

**2.4 ROUNDING EFFECTS IN DOT PRODUCT COMPUTATION**

Suppose *x,y* ∈ **R***n*. We recall from Chapter 1 (and from elementary linear algebra) that the vector dot product is given by

〈*x,y*〉 = *xT y* = *yT x* =

∑*n*−1*k*=0

*xkyk. (*2*.*25*)*

*xkyk. (*2*.*25*)*

This operation occurs in matrix–vector product computation (e.g., *y* = *Ax*, where *A* ∈ **R***n*×*n*), digital filter implementation (i.e., computing discrete-time convolu- tion), numerical integration, and other applications. In other words, it is so common that it is important to understand how rounding errors can affect the accuracy of a computed dot product.

We may regard dot product computation as a recursive process. Thus

*sn*−1 =

*n*−1∑*k*=0

*xkyk* =

*xkyk* =

∑*n*−2*xkyk* + *xn*−1*yn*−1 = *sn*−2 + *xn*−1*yn*−1*. k*=0

∑*n*−2*xkyk* + *xn*−1*yn*−1 = *sn*−2 + *xn*−1*yn*−1*. k*=0

∑*n*−2*xkyk* + *xn*−1*yn*−1 = *sn*−2 + *xn*−1*yn*−1*. k*=0

So

*sk* = *sk*−1 + *xkyk (*2*.*26*)*

for *k* = 0*,*1*,...,n* − 1, and *s*−1 = 0. Each arithmetic operation in (2.26) is a separate floating-point operation and so introduces its own error into the over- all calculation. We would like to obtain a general expression for this error. To begin, we may model the computation process according to

ˆ*s*0 = *fl*[*x*0*y*0]

ˆ*s*1 = *fl*[ˆ*s*0 + *fl*[*x*1*y*1]]

ˆ*s*2 = *fl*[ˆ*s*1 + *fl*[*x*2*y*2]]

*...*

ˆ*sn*−2 = *fl*[ˆ*sn*−3 + *fl*[*xn*−2*yn*−2]]

ˆ*sn*−1 = *fl*[ˆ*sn*−2 + *fl*[*xn*−1*yn*−1]]*.* (2.27)

From (2.15) we may write

ˆ*s*0 = *(x*0*y*0*)(*1 + *δ*0*)*

ˆ*s*1 = [ˆ*s*0 + *(x*1*y*1*)(*1 + *δ*1*)*]*(*1 + *ε*1*)*

ˆ*s*2 = [ˆ*s*1 + *(x*2*y*2*)(*1 + *δ*2*)*]*(*1 + *ε*2*)*

*...*

TLFeBOOK

ROUNDING EFFECTS IN DOT PRODUCT COMPUTATION **49**

ˆ*sn*−2 = [ˆ*sn*−3 + *(xn*−2*yn*−2*)(*1 + *δn*−2*)*]*(*1 + *εn*−2*)*

ˆ*sn*−1 = [ˆ*sn*−2 + *(xn*−1*yn*−1*)(*1 + *δn*−1*)*]*(*1 + *εn*−1*),* (2.28)

where |*δk*| ≤ 2−*t n* − 1), via (2.16). (for *k* = 0*,*1*,...,n* − 1), It is possible to write4

and |*εk*| ≤ 2−*t* (for *k* = 1*,*2*,...,*

ˆ*sn*−1 =

∑*n*−1*k*=0

*xkyk(*1 + *γk)* = *sn*−1 +

*xkyk(*1 + *γk)* = *sn*−1 +

*n*−1∑*xkykγk, (*2*.*29*) k*=0

*n*−1∑*xkykγk, (*2*.*29*) k*=0

*n*−1∑*xkykγk, (*2*.*29*) k*=0

where

1 + *γk* = *(*1 + *δk)*

*n*−1∏*j*=*k(*1 + *εj)(ε*0 = 0*). (*2*.*30*)*

Note that the notation means, for example

∏*nk*=0*xk* = *x*0*x*1*x*2 ···*xn*−1*xn, (*2*.*31*)*

where is the symbol to compute the product of all *xk* for *k* = 0*,*1*,...,n*. The similarity to how we interpret notation should therefore be clear.

The absolute value operator is a norm on **R**, so from the axioms for a norm (recall Definition 1.3), we must have

|*sn*−1 − ˆ*sn*−1|=|*xT y* − *fl*[*xT y*]| ≤

∑*n*−1*k*=0

|*xkyk*||*γk*|*. (*2*.*32*)*

|*xkyk*||*γk*|*. (*2*.*32*)*

In particular, obtaining this involves the repeated use of the triangle inequality. Equation (2.32) thus represents an upper bound on the absolute error involved in computing a vector dot product. Of course, the notation *fl*[*xT y*] symbolizes the floating-point approximation to the exact quantity *xT y*. However, the bound in (2.32) is incomplete because we need to appropriately bound the numbers *γk*.

To obtain the bound we wish involves using the following lemma.

**Lemma 2.1:** We have

1 + *x* ≤ *ex, x* ≥ 0 (2.33a)

*ex* ≤ 1 + 1*.*01*x,* 0 ≤ *x* ≤ *.*01*.* (2.33b)

4Equation (2.29) is most easily arrived at by considering examples for small *n*, for instance

ˆ*s*3 = *x*0*y*0*(*1 + *δ*0*)(*1 + *ε*0*)(*1 + *ε*1*)(*1 + *ε*2*)(*1 + *ε*3*)* + *x*1*y*1*(*1 + *δ*1*)(*1 + *ε*1*)(*1 + *ε*2*)(*1 + *ε*3*)*

+ *x*2*y*2*(*1 + *δ*2*)(*1 + *ε*2*)(*1 + *ε*3*)* + *x*3*y*3*(*1 + *δ*3*)(*1 + *ε*3*),*

and using such examples to “spot the pattern.”

TLFeBOOK

**50** NUMBER REPRESENTATIONS

***Proof*** Begin with consideration of (2.33a). Recall that for −∞ *<x<* ∞

*ex* =

∑∞*n*=0

*xn*! *n*

*. (*2*.*34*)*

Therefore

*ex* = 1 + *x* +

∑∞*n*=2

*xn*

*n*! *,*

but the terms in the summation are all nonnegative, so (2.33a) follows immediately. Now consider (2.33b), which is certainly valid for *x* = 0. The result will follow if we prove *ex* − 1

*(m* + 1*)*! ≤ 0*.*01

for 0 *< x* ≤ 0*.*01. Observe that

∑∞*m*=1

*xm*

1 − *x .*

It is not hard to verify that

1

1 − *x* ≤ 0*.*01

for 0 *< x* ≤ 0*.*01. Thus, (2.33b) follows.

TLFeBOOK

*n*!

so that

1 + *x* = *ex* −

∑∞*n*=2

*xn*

*x* ≤ 1*.*01 *(x* = 0*).*

From (2.34)

*ex* − 1

*(m* + 1*)*!

so we may also equivalently prove instead that

∑∞*m*=1

*xm*

*(m* + 1*)*! = 1

2*x* + *x*2 + *x*3 + *x*4 +···

= 1

*xk* − 1 − *x*

= 1

*x* =

2*x* + 1

2*x* +

1 − *x* − 1

∞∑*k*=2

∞∑*m*=0

6*x*2 + 1

2*x* − 1 = 1

*xk* = 1

2*x*1 + *x*

*(m* + 1*)*! = 1 +

*xm*

24*x*3 +···≤ 1

2*x* +

2*x* 1 + *x*

∞∑*k*=0

∞∑*m*=1

*xm*

ROUNDING EFFECTS IN DOT PRODUCT COMPUTATION **51**

If *n* = 1*,*2*,*3*,...*, and if 0 ≤ *nu* ≤ 0*.*01, then

*(*1 + *u)n* ≤ *(eu)n* [via *(*2*.*33a*)*]

≤ 1 + 1*.*01*nu* [via *(*2*.*33b*)*]*.* (2.35)

Now if |*δi*| ≤ *u* for *i* = 0*,*1*,...,n* − 1 then

*n*−1∏*i*=0*(*1 + *δi)* ≤

*n*−1∏*i*=0*(*1 + |*δi*|*)* ≤ *(*1 + *u)n*

so via (2.35)

*n*−1∏*i*=0*(*1 + *δi)* ≤ 1 + 1*.*01*nu, (*2*.*36*)*

where we must emphasize that 0 ≤ *nu* ≤ 0*.*01. Certainly there is a *δ* such that

1 + *δ* =

*n*−1∏*i*=0*(*1 + *δi), (*2*.*37*)*

and so from (2.36), |*δ*| ≤ 1*.*01*nu*. If we identify *γk* with *δ* in (2.33) for all *k*, then

|*γk*| ≤ 1*.*01*nu (*2*.*38*)*

for which we consider *u* = 2−*t* [because in (2.30) both |*εi*| and |*δi*| ≤ 2−*t*]. Using (2.38) in (2.32), we obtain

|*xT y* − *fl*[*xT y*]| ≤ 1*.*01*nu*

*n*−1∑*k*=0

|*xkyk*|*, (*2*.*39*)*

but |*x*| = ∑[|*xn*−1 *k*=0 0||*x*|*xky*1|···|*xk*| = ∑*n*−1|]*n*−1 *k*=0 *T* |*x*). *k*||*yk*|, and this may be symbolized Thus, we may rewrite (2.39) as

as |*x*|*T* |*y*| (so that

|*xT y* − *fl*[*xT y*]| ≤ 1*.*01*nu*|*x*|*T* |*y*|*. (*2*.*40*)*

Observe that the relative error satisfies

|*xT y* − *fl*[*xT y*]|

|*xT y*| *. (*2*.*41*)*

The bound in (2.41) may be quite large if |*x*|*T* |*y*|≫|*xT y*|. This suggests the possibility of a large relative error. We remark that since *u* = 2−*t*, *nu* ≤ 0*.*01 will hold in all practical cases unless *n* is very large (a typical value for *t* is *t* = 56).

The potentially large relative errors indicated by the analysis we have just made are a consequence of the details of how the dot product was calculated. As noted on p. 65 of Ref. 4, the use of a double-precision accumulator to compute the dotTLFeBOOK

|*xT y*| ≤ 1*.*01*nu*|*x*|*T* |*y*|

||*x*||2 = 1*.*01*nu* (|*x*|*T* |*x*| = ∑*n*−1 *k*=0 |*xk*|2 = ∑*n*−1 *k*=0 *x*2*k* = ||*x*||2, and |||*x*||2| = ||*x*||2). So in “short- hand” notation, *fl*[√

**52** NUMBER REPRESENTATIONS

product can reduce the error dramatically. Essentially, if *x* and *y* are floating- point vectors with *t*-bit mantissas, the “running sum” *sk* [of (2.26)] is built up in an accumulator with a 2*t*-bit mantissa. Multiplication of two *t*-bit numbers can be stored exactly in a double-precision variable. The large dynamic floating-point range limits the likelihood of overflow/underflow. Only when final sum *sn*−1 is written to a single-precision memory location will there be a rounding error. It therefore follows that when this alternative procedure is employed, we get

*fl*[*xT y*] = *xT y(*1 + *δ) (*2*.*42*)*

for which |*δ*| ≈ 2−*t* (= *u*). Clearly, this is a big improvement.

The material of this section shows

1. The analysis required to obtain insightful bounds on errors can be quite

arduous. 2. Proper numerical technique can have a dramatic effect in reducing errors. 3. Proper technique can be revealed by analysis.

The following example illustrates how the bound on rounding error in dot prod- uct computation may be employed.

*fl*[**Example** √ **2.4** Assume the existence of a square root function such that *x(*1 bound of Eq. (2.40) rithm for ||*x*|| = + √ *xε) T* to *x*. and compute This |*ε*| can ≤ *xu*. *T* be *x* We expressed (*x* use ∈ **R**the *n*), and algorithm then use that this corresponds to give an to algo- the

in the form of pseudocode:

*s*−1 := 0; for *k* := 0 to *n* − 1 do begin ||*x*|| *s*end; := *k* := √

*ssk*−1 *n*−1;

+ *x*2*k*;

We of ||*x*||. will now We will obtain use a the bound fact on that the √

relative 1 + *x* ≤ error 1 + due *x* (for to rounding *x* ≥ 0).

in the computation

Now

*ε*1 = *fl*[*xT xx*] *T x* − *xT x*

⇒ *fl*[*xT x*] = *xT x(*1 + *ε*1*),*

and via (2.41)

|*ε*1| ≤ 1*.*01*nu*|*x*|*T* |*x*|

*x*] = √

*fl*[*xT x*]] ≡ *fl*[||*x*||], and *fl*[||*x*||] = √

*xT x*√

|*xT x*| = 1*.*01*nu*||*x*||2

1 + *ε*1*(*1 + *ε)* = ||*x*||√

1 + *ε*1*(*1 + *ε),*

TLFeBOOK

MACHINE EPSILON **53**

and √

1 + *ε*1 ≤ 1 + *ε*1, so*fl*[||*x*||] ≤ ||*x*||*(*1 + *ε*1*)(*1 + *ε).*

Now *(*1 + *ε*1*)(*1 + *ε)* = 1 + *ε*1 + *ε* + *ε*1*ε*, implying that

||*x*||*(*1 + *ε*1*)(*1 + *ε)* = ||*x*|| + ||*x*||*(ε*1 + *ε* + *ε*1*ε)*

so therefore

*fl*[||*x*||] ≤ ||*x*|| + ||*x*||*(ε*1 + *ε* + *ε*1*ε),*

and thus ∣∣∣∣*fl*[||*x*||] − ||*x*||

∣∣∣∣ ≤ |*ε*1 + *ε* + *ε*1*ε*| ≤ *u* + 1*.*01*nu* + 1*.*01*nu*2

= *u*[1 + 1*.*01*n* + 1*.*01*nu*]*.*

Of course, we have used the fact that |*ε*| ≤ *u*.

**2.5 MACHINE EPSILON**

In Section 2.3 upper bounds on the error involved in applying the operator *fl* were derived. Specifically, we found that the relative error satisfies

|*ε* | = |*x* − *fl*[*x*]|

{ 2−*t* 2−*t*+1 (rounding)

(chopping) *. (*2*.*43*)*

As suggested in Section 2.4, these bounds are often denoted by *u*; that is, *u* = 2−*t* for rounding, and *u* = 2−*t*+1 for chopping. The bound *u* is often called the *unit roundoff* [4, Section 2.4.2].

The details of how floating-point arithmetic is implemented on any given com- puting machine may not be known or readily determined by the user. Thus, *u* may not be known. However, an “experimental” approach is possible. One may run a simple program to “estimate” *u*, and the estimate is the *machine epsilon*, denoted *εM*. the next biggest The machine epsilon is defined to be the difference between 1.0 and floating-point number [6, Section 2.1]. Consequently, *εM* = 2−*t*+1. A pseudocode to compute *εM* is as follows:

stop := 1; eps := 1.0; while stop == 1 do begin

eps := eps/2.0; x := 1.0 + eps; if *x* ≤ 1.0 begin

TLFeBOOK

||*x*||

|*x*| ≤

**54** NUMBER REPRESENTATIONS

stop := 0; end; end; eps := 2.0 ∗ eps;

This code may be readily implemented as a MATLAB routine. MATLAB stores eps (= *εM*) as a built-in constant, and the reader may wish to test the code above to see if the result agrees with MATLAB eps (as a programming exercise).

In this book we shall (unless otherwise stated) regard machine epsilon and unit roundoff as practically interchangeable.

**APPENDIX 2.A REVIEW OF BINARY NUMBER CODES**

This appendix summarizes typical methods used to represent integers in binary. Extension of the results in this appendix to fractions is certainly possible. This material is normally to be found in introductory digital electronics books. The reader is here assumed to know Boolean algebra. This implies that the reader knows that + can represent either algebraic addition, or the logical OR operation. Similarly, *xy* might mean the logical AND of the Boolean variables *x* and *y*, or it might mean the arithmetic product of the real variables *x* and *y*. The context must be considered to ascertain which meaning applies.

Below we speak of “complements.” These are used to represent negative inte- gers, and also to facilitate arithmetic with integers. We remark that the results of this appendix are presented in a fairly general manner. Thus, the reader may wish, for instance, to see numerical examples of arithmetic using two’s complement (2’s comp.) codings. The reader can consult pp. 276–280 of Ref. 2 for such examples. Almost any other books on digital logic will also provide a source of numerical examples [3].

We may typically interpret a bit pattern in one of four ways, assuming that the bit pattern is to represent a number (negative or nonnegative integer). An example of this is as follows, and it provides a summary of common representations (e.g., for *n* = 3 bits):

Bit Pattern Unsigned Integer 2’s Comp. 1’s Comp. Sign Magnitude

0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 2 2 2 2 0 1 1 3 3 3 3 1 0 0 4 −4 −3 −0 1 0 1 5 −3 −2 −1 1 1 0 6 −2 −1 −2 1 1 1 7 −1 −0 −3

TLFeBOOK

REVIEW OF BINARY NUMBER CODES **55**

In the four coding schemes summarized in this table, the interpretation of the bit pattern is always the same when the most significant bit (MSB) is zero. A similar table for *n* = 4 appears in Hamacher et al. [2, see p. 271].

Note that, philosophically speaking, the table above implies that a bit pattern can have more than one meaning. It is up to the engineer to decide what meaning it should have. Of course, this will be a function of purpose. Presently, our purpose is that bit patterns should have meaning with respect to the problems of numerical computing; that is, bit patterns must represent numerical information.

The relative merits of the three signed number coding schemes illustrated in the table above may be summarized as follows:

Coding Scheme Advantages Disadvantages

Circuit 2’s complement Simple adder/subtracter

for finding the 2’s comp. circuit

more complex than circuit for Only one code for 0

finding the 1’s comp. 1’s complement Easy to obtain the 1’s comp.

of a number

Circuit for addition and

subtraction more complex than for the 2’s comp. adder/subtracter Two codes for 0 Sign magnitude Intuitively obvious code Has the most complex

adder/subtracter circuit Two codes for 0

The following is a summary of some formulas associated with arithmetic (i.e., addition and subtraction) with *r*’s and *(r* − 1*)*’s complements. In binary arithmetic *r* = 2, while in decimal arithmetic *r* = 10. We emphasize the case *r* = 2.

Let *A* be an *n*-digit base-*r* number (integer)

*A* = *An*−1*An*−2 ···*A*1*A*0

where *Ak* ∈ {0*,*1*,...,r* − 2*,r* − 1}. Digit *An*−1 is the most significant digit (MSD), while digit *A*0 is the least significant digit (LSD). Provided that *A* is not negative (i.e., is unsigned), we recognize that to convert *A* to a base-10 repre- sentation (i.e., ordinary decimal number) requires us to compute

∑*n*−1*k*=0

*Akrk.*

If *A* is allowed to be a negative integer, the usage of this summation needs modi- fication. This is considered below.

The *r*’s complement of *A* is defined to be

*r*’s complement of *A* = *A*∗ =

{ *rn* − *A, A* = 0

0*, A* = 0 *(*2.A.1*)*TLFeBOOK

**56** NUMBER REPRESENTATIONS

The *(r* − 1*)*’s complement of *A* is defined to be

*(r* − 1*)*’s complement of *A* =

*A* = *(rn* − 1*)* − *A (*2.A.2*)*

It is important not to confuse the bar over the *A* in (2.A.2) with the Boolean NOT operation, although for the special case of *r* = 2 the bar will denote complemen- tation of each bit of *A*; that is, for *r* = 2

*A*0

where the bar now denotes the logical NOT operation. More generally, if *A* is a base-*r* number

*A* = *(r* − 1*)* − *An*−1 *(r* − 1*)* − *An*−2 ···*(r* − 1*)* − *A*1 *(r* − 1*)* − *A*0

Thus, to obtain

*A*, each digit of *A* is subtracted from *r* − 1. As a consequence, comparing (2.A.1) and (2.A.2), we see that

*A*∗ =

*A* + 1 *(*2.A.3*)*

where the plus denotes algebraic addition (which takes place in base *r*).

Now we consider the three (previously noted) different methods for coding integers when *r* = 2:

1. Sign-magnitude coding 2. One’s complement coding 3. Two’s complement coding

In all three of these coding schemes the most significant bit (MSB) is the *sign bit*. Specifically , if *An*−1 = 0, the number is nonnegative, and if *An*−1 = 1, the number is negative. It can be shown that when the complement (either one’s or two’s) of a binary number is taken, this is equivalent to placing a minus sign in front of the number. As a consequence, when given a binary number *A* = *An*−1*An*−2 ···*A*1*A*0 coded according to one of these three schemes, we may convert that number to a base-10 integer according to the following formulas:

1. *Sign-Magnitude Coding.* The sign-magnitude binary number *A* = *An*−1*An*−2 ···*A*1*A*0 (*Ak* ∈ {0*,*1}) has the base-10 equivalent

*A* =



∑*n*−2*i*=0

*Ai*2*i, An*−1 = 0

−

∑*n*−2*Ai*2*i, An*−1 = 1 *(*2.A.4*)i*=0

TLFeBOOK

*A* =

*An*−1

*An*−2 ···

*A*1

*)*}

2 *n*

2. *One’s Complement Coding.* In this coding we represent −*A* as

*)*}

2 *n*

3. *Two’s Complement Coding.* In this coding we represent −*A* as *A*∗ (=

REVIEW OF BINARY NUMBER CODES **57**

With this coding scheme there are two codings for zero:

*(*0*)*10 = *(*000···00 }

*A*. The one’s complement binary number *A* = *An*−1*An*−2 ···*A*1*A*0 (*Ak* ∈ {0*,*1}) has the base-10 equivalent

*A* =



∑*n*−2*i*=0

*Ai*2*i ,An*−1 = 0

−

*(*2.A.5*)*

With this coding scheme there are also two codes for zero:

*(*0*)*10 = *(*000···00 }

*n*−2∑*i*=0

*A* + 1). The two’s complement binary number *A* = *An*−1*An*−2 ···*A*1*A*0 (*Ak* ∈ {0*,*1}) has the base-10 equivalent*A* = −2*n*−1*An*−1 +

∑*n*−2*i*=0

*Ai*2*i (*2.A.6*)*

The proof equivalent base-2 to one is is as *A* in follows. = base-10), ∑*n*−2 *i*=0 If *AA*which *i*2*n*−1 *i* = 0, then *A* ≥ 0 and immediately the base-10 (via the procedure for converting a number in is (2.A.6) for *An*−1 = 0. Now, if *An*−1 = 1, then *A <* 0, and so if we take the two’s complement of *A* we must get |*A*|:

|*A*| =

*A* + 1

= *(*1 − *An*−1*)(*1 − *An*−2*)*···*(*1 − *A*1*)(*1 − *A*0*)* + 00···01 } } *n*

=

∑*n*−1*i*=0*(*1 − *Ai)*2*i* + 1

= 2*n*−1*(*1 − *An*−1*)* +

∑*n*−2*i*=0*(*1 − *Ai)*2*i* + 1

=

∑*n*−2*n*−2∑*i*=0 *i*=0 2*i* + 1 −

*Ai*2*i(An*−1 = 1*)*

TLFeBOOK

{{

{{

} *n*

} *n*

*Ai*2*i ,An*−1 = 1

*)*2 = *(*100···00 }

*)*2 = *(*111···11 }

{{

{{

{{

**58** NUMBER REPRESENTATIONS

= 1 − 2*n*−1

)

= 2*n*−1 −

*n*−2∑*i*=0

*Ai*2*i*

and so *A* = −2*n*−1 + ∑*n*−2 *i*=0 *Ai*2*i*, which is (2.A.6) for *An*−1 = 1. In this coding scheme there is only one code for zero:

*(*0*)*10 = *(*000···00 }

*)*2

When *n*-bit integers are added together, there is the possibility that the sum may not fit in *n* bits. This is *overflow*. The condition is easy to detect by monitoring the signs of the operands and the sum. Suppose that *x* and *y* are *n*-bit two’s complement coded integers, so that the sign bits of these operands are *xn*−1 and *yn*−1. Suppose that the sum is denoted by *s*, implying that the sign bit is *sn*−1. The Boolean function that tests for overflow of *s* = *x* + *y* (algebraic sum of *x* and *y*) is

*T* = *xn*−1*yn*−1

} *n*

*yn*−1*sn*−1*.*

The first term will be logical 1 if the operands are negative while the sum is positive. The second term will be logical 1 if the operands are positive but the sum is negative. Either condition yields *T* = 1, thus indicating an overflow. A similar test may be obtained for subtraction, but we omit this here.

The following is both the procedure and the justification of the procedure for adding two’s complement coded integers.

**Theorem 2.A.1: Two’s Complement Addition** If *A* and *B* are *n*-bit two’s complement coded numbers, then compute *A* + *B* (the sum of *A* and *B*) as though they were unsigned numbers, discarding any carryout.

***Proof*** Suppose that *A >* 0*,B >* 0; then *A* + *B* will generate no carryout from the bit position *n* − 1 since *An*−1 and the result will be correct if *A* = + *BB n*−1 *<* 2= *n*−10 (i.e., . the sign bits are zero-valued), (If this inequality is not satisfied, then the sign bit will be one, indicating a negative answer, which is wrong. This amounts to an overflow.)

Suppose that *A* ≥ *B >* 0; then

*A* + *(*−*B)* = *A* + *B*∗ = *A* + 2*n* − *B* = 2*n* + *A* − *B,*

and if we discard the carryout, this is equivalent to subtracting 2*n* (because the carryout has a weight of 2*n*). Doing this yields *A* + *(*−*B)* = *A* − *B*.

TLFeBOOK

1 − 2 + 1 −

*n*−2∑*i*=0

*sn*−1 +

*Ai*2*i*

*xn*−1

(via

{{

*n*∑*i*=0

*ai* = 1 − *an*+1

1 − *a*

PROBLEMS **59**

Similarly

*(*−*A)* + *B* = *A*∗ + *B* = 2*n* − *A* + *B* = 2*n* + *B* − *A,*

and discarding the carry out yields *(*−*A)* + *B* = *B* − *A*.

Again, suppose that *A* ≥ *B >* 0, then

*(*−*A)* + *(*−*B)* = *A*∗ + *B*∗ = 2*n* − *A* + 2*n* − *B* = 2*n* + [2*n* − *(A* + *B)*]

= 2*n* + *(A* + *B)*∗

so discarding the carryout gives *(*−*A)* + *(*−*B)* = *(A* + *B)*∗, which is the desired two’s complement representation of −*(A* + *B)*, provided *A* + *B* ≤ 2*n*−1. (If this latter inequality is not satisfied, then we have an overflow.)

The procedure for subtraction (and its justification) follows similarly. We omit these details.

**REFERENCES**

1. J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood

Cliffs, NJ, 1963. 2. V. C. Hamacher, Z Vranesic, and S. G. Zaky, *Computer Organization*, 3rd ed.,

McGraw-Hill, New York, 1990. 3. J. F. Wakerly, *Digital Design Principles and Practices*, Prentice-Hall, Englewood Cliffs,

NJ, 1990. 4. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., Johns Hopkins Univ.

Press, Baltimore, MD, 1989. 5. G. E. Forsythe, M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical*

*Computations*, Prentice-Hall, Englewood Cliffs, NJ, 1977. 6. N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA,

1996.

**PROBLEMS**

**2.1.** Let *fx*[*x*] denote the operation of reducing *x* (a fixed-point binary fraction) to *t* + 1 bits (including the sign bit) according to the Wilkinson rounding (ordinary rounding) procedure in Section 2.2. Suppose that *a* = *(*0*.*1000*)*2, *b* = *(*0*.*1001*)*2, and *c* = *(*0*.*0101*)*2, so *t* = 4 here. In arithmetic of unlimited precision, we always have *a(b* + *c)* = *ab* + *ac*. Suppose that a practical com- puting machine applies the operator *fx*[·] after every arithmetic operation.

**(a)** Find *x* = *fx*[*fx*[*ab*] + *fx*[*ac*]]. **(b)** Find *y* = *fx*[*af x*[*b* + *c*]].

Do you obtain *x* = *y*?

TLFeBOOK

**60** NUMBER REPRESENTATIONS

This problem shows that the order of operations in an algorithm implemented on a practical computer can affect the answer obtained.

**2.2.** Recall from Section 2.2 that*q* = (1

01*)*2*.*

Find the absolute error in representing *q* as a *(t* + 1*)*-bit binary number. Find the relative error. Assume both ordinary rounding and chopping (defined at the end of Section 2.3 with respect to floating-point arithmetic).

**2.3.** Recall that we define a floating-point number in base *r* to have the form

*x* = *x*}

0*.x*1*x*2 ···*xt*−1*xt* ×*re,*

where is the exponent *x*0 ∈ {+*,*−} (a signed (sign digit), integer), *xk* and ∈ {0*,*1*,...,r x*1 = 0 (so − *r*1} −1 for *k* = 1*,*2*,...,t*, *e* ≤ |*f*| *<* 1) if *x* = 0. Show that for *x* = 0

*m* ≤ |*x*| ≤ *M,*

where for *L* ≤ *e* ≤ *U*, we have

*m* = *rL*−1*, M* = *rU(*1 − *r*−*t).*

**2.4.** Suppose *r* = 10. We may consider the *result* of a decimal arithmetic operation

in the floating-point representation to be

*x* = ±

*xk*10−*k*)

× 10*e.*

**(a)** If *fl*[*x*] is the operator for *chopping*, then

*fl*[*x*] = *(*±*.x*1*x*2 ···*xt*−1*xt)* × 10*e,*

thus, all digits *xk* for *k>t* are forced to zero. **(b)** If *fl*[*x*] is the operator for *rounding* then it is defined as follows. Add

0*.*00···01 }

( ∑∞*k*=1

}

to the mantissa if *xt*+1 ≥ 5, but if *xt*+1 *<* 5, the mantissa is *t*+1 digits unchanged. Then all digits *xk* for *k>t* are forced to zero. Show that the *absolute error for chopping* satisfies the upper bound

|*x* − *fl*[*x*]| ≤ 10−*t*10*e,*

and that the *absolute error for rounding* satisfies the upper bound

|*x* − *fl*[*x*]| ≤ 1

{{

3)10 = *(*0*.*

{{

} =*f*

210−*t*10*e.*

TLFeBOOK

PROBLEMS **61**

Show that the *relative errors* satisfy

|*ε*| = |*x* − *fl*[*x*]|

2101−*t* (rounding) *.*

**2.5.** Suppose that *t* = 4 and *r* = 2 (i.e., we are working with floating-point binary

numbers). Suppose that we have the operands

*x* = 0*.*1011 × 10−3*, y* = −0*.*1101 × 102*.*

Find *x* + *y*, *x* − *y*, and *xy*. Clearly show the steps involved.

**2.6.** Suppose that *A* ∈ **R***n*×*n*, *x* ∈ **R***n*, and that *fl*[*Ax*] represents the result of computing the product *Ax* on a floating-point computer. Define |*A*| = [|*ai,j*|]*i,j*=0*,*1*,...,n*−1, and |*x*| = [|*x*0||*x*1|···|*xn*−1|]*T* . We have

*fl*[*Ax*] = *Ax* + *e,*

where *e* ∈ **R***n* is the error vector. Of course, *e* models the rounding errors involved in the actual computation of product *Ax* on the computer. Justify the bound

|*e*| ≤ 1*.*01*nu*|*A*||*x*|*.*

**2.7.** Explain why a conditional test such as

if *x* = *y* then begin *f* := *f/(x* − *y)*; end;

is unreliable.

(*Hint:* Think about dynamic range limitations in floating-point arithmetic.) **2.8.** Suppose that *x* = [*x*0*x*1 ···*xn*−1]*T* is a real-valued vector, ||*x*||∞ =

max0≤*k*≤*n*−1 |*xk*|, and that we wish to compute ||*x*||2 =

[∑*n*−1 *k*=0 *xk*2]1*/*2. Explain respect to the computational advantages, and efficiency disadvantages (number of of the arithmetic following operations, algorithm with and comparisons), and dynamic range limitations in floating-point arithmetic:

*m* := ||*x*||∞; *s* := 0; for *k* := 0 to *n* − 1 do begin

*s* := *s* + *(xk/m)*2; ||*x*||end;

2 := *m*√

*s*;

Comments regarding computational efficiency may be made with respect to the pseudocode algorithm in Example 2.4.

TLFeBOOK

|*x*| ≤

{ 101−*t* (chopping)

1