

COMPUTATIONAL MATHEMATICS SERIES

INSIDE
the
FFT BLACK
BOX

Serial and Parallel Fast
Fourier Transform
Algorithms

COMPUTATIONAL MATHEMATICS SERIES

INSIDE the FFT BLACK BOX

Serial and Parallel Fast
Fourier Transform
Algorithms

Eleanor Chu

*University of Guelph
Ontario, Canada*

Alan George

*University of Waterloo
Ontario, Canada*



CRC Press

Boca Raton London New York Washington, D.C.

Library of Congress Cataloging-in-Publication Data

Catalog record is available from the Library of Congress

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the author and the publisher cannot assume responsibility for the validity of all materials or for the consequences of their use.

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

The consent of CRC Press LLC does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press LLC for such copying.

Direct all inquiries to CRC Press LLC, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431, or visit our Web site at www.crcpress.com

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation, without intent to infringe.

© 2000 by CRC Press LLC

No claim to original U.S. Government works
International Standard Book Number 0-8493-0270-6
Library of Congress Card Number 99-048017

Printed in the United States of America 2 3 4 5 6 7 8 9 0
Printed on acid-free

Contents

I Preliminaries

1 An Elementary Introduction to the Discrete Fourier Transform

- 1.1 Complex Numbers
 - 1.2 Trigonometric Interpolation
 - 1.3 Analyzing the Series
 - 1.4 Fourier Frequency Versus Time Frequency
 - 1.5 Filtering a Signal
 - 1.6 How Often Does One Sample?
 - 1.7 Notes and References
-

2 Some Mathematical and Computational Preliminaries

- 2.1 Computing the Twiddle Factors
 - 2.2 Multiplying Two Complex Numbers
 - 2.2.1 Real floating-point operation (FLOP) count
 - 2.2.2 Special considerations in computing the FFT
 - 2.3 Expressing Complex Multiply-Adds in Terms of Real Multiply-Adds
 - 2.4 Solving Recurrences to Determine an Unknown Function
-

II Sequential FFT Algorithms

3 The Divide-and-Conquer Paradigm and Two Basic FFT Algorithms

- 3.1 Radix-2 Decimation-In-Time (DIT) FFT
 - 3.1.1 Analyzing the arithmetic cost
 - 3.2 Radix-2 Decimation-In-Frequency (DIF) FFT
 - 3.2.1 Analyzing the arithmetic cost
 - 3.3 Notes and References
-

4 Deciphering the Scrambled Output from In-Place FFT Computation

- 4.1 Iterative Form of the Radix-2 DIF FFT

- 4.2 Applying the Iterative DIF FFT to a $N = 32$ Example
 - 4.3 Storing and Accessing Pre-computed Twiddle Factors
 - 4.4 A Binary Address Based Notation and the Bit-Reversed Output
 - 4.4.1 Binary representation of positive decimal integers
 - 4.4.2 Deciphering the scrambled output
 - 4.5 Shorthand Notation for the Twiddle Factors
-

5 Bit-Reversed Input to the Radix-2 DIF FFT

- 5.1 The Effect of Bit-Reversed Input
 - 5.2 A Taxonomy for Radix-2 FFT Algorithms
 - 5.3 Shorthand Notation for the DIF_{RN} Algorithm
 - 5.3.1 Shorthand notation for the twiddle factors
 - 5.3.2 Applying algorithm 5.2 to a $N = 32$ example
 - 5.4 Using Scrambled Output for Input to the Inverse FFT
 - 5.5 Notes and References
-

6 Performing Bit-Reversal by Repeated Permutation of Intermediate Results

- 6.1 Combining Permutation with Butterfly Computation
 - 6.1.1 The ordered radix-2 DIF_{NN} FFT
 - 6.1.2 The shorthand notation
 - 6.2 Applying the Ordered DIF FFT to a $N = 32$ Example
 - 6.3 In-Place Ordered (or Self-Sorting) Radix-2 FFT Algorithms
-

7 An In-Place Radix-2 DIT FFT for Input in Natural Order

- 7.1 Understanding the Recursive DIT FFT and its In-Place Implementation
 - 7.2 Developing the Iterative In-Place DIT FFT
 - 7.2.1 Identifying the twiddle factors in the DIT FFT
 - 7.2.2 The pseudo-code program for the DIT_{NR} FFT algorithm
 - 7.3 Shorthand Notation and a $N = 32$ Example
-

8 An In-Place Radix-2 DIT FFT for Input in Bit-Reversed Order

- 8.1 Developing the Iterative In-Place DIT_{RN} FFT
 - 8.1.1 Identifying the twiddle factors in the DIT_{RN} FFT
 - 8.1.2 The pseudo-code program for the DIT_{RN} FFT
 - 8.2 Shorthand Notation and a $N = 32$ Example
-

9 An Ordered Radix-2 DIT FFT

- 9.1 Deriving the (Ordered) DIT_{NN} FFT From Its Recursive Definition
 - 9.2 The Pseudo-code Program for the DIT_{NN} FFT
 - 9.3 Applying the (Ordered) DIT_{NN} FFT to a $N = 32$ Example
-

10 Ordering Algorithms and Computer Implementation of Radix-2 FFTs

- 10.1 Bit-Reversal and Ordered FFTs
 - 10.2 Perfect Shuffle and In-Place FFTs
 - 10.2.1 Combining a software implementation with the FFT
 - 10.2.2 Data adjacency afforded by a hardware implementation
 - 10.3 Reverse Perfect Shuffle and In-Place FFTs
 - 10.4 Fictitious Block Perfect Shuffle and Ordered FFTs
 - 10.4.1 Interpreting the ordered DIF_{NN} FFT algorithm
 - 10.4.2 Interpreting the ordered DIT_{NN} FFT algorithm
-

11 The Radix-4 and the Class of Radix-2^s FFTs

- 11.1 The Radix-4 DIT FFTs
 - 11.1.1 Analyzing the arithmetic cost
 - 11.2 The Radix-4 DIF FFTs
 - 11.3 The Class of Radix-2^s DIT and DIF FFTs
-

12 The Mixed-Radix and Split-Radix FFTs

- 12.1 The Mixed-Radix FFTs
 - 12.2 The Split-Radix DIT FFTs
 - 12.2.1 Analyzing the arithmetic cost
 - 12.3 The Split-Radix DIF FFTs
 - 12.4 Notes and References
-

13 FFTs for Arbitrary N

- 13.1 The Main Ideas Behind Bluestein's FFT
 - 13.1.1 DFT and the symmetric Toeplitz matrix-vector product
 - 13.1.2 Enlarging the Toeplitz matrix to a circulant matrix
 - 13.1.3 Enlarging the dimension of a circulant matrix to $M = 2^s$
 - 13.1.4 Forming the $M \times M$ circulant matrix-vector product
 - 13.1.5 Diagonalizing a circulant matrix by a DFT matrix
 - 13.2 Bluestein's Algorithm for Arbitrary N
-

14 FFTs for Real Input

- 14.1 Computing Two Real FFTs Simultaneously
 - 14.2 Computing a Real FFT
 - 14.3 Notes and References
-

15 FFTs for Composite N

- 15.1 Nested-Multiplication as a Computational Tool
 - 15.1.1 Evaluating a polynomial by nested-multiplication
 - 15.1.2 Computing a DFT by nested-multiplication
- 15.2 A 2D Array as a Basic Programming Tool

- 15.2.1 Row-oriented and column-oriented code templates
 - 15.3 A 2D Array as an Algorithmic Tool
 - 15.3.1 Storing a vector in a 2D array
 - 15.3.2 Use of 2D arrays in computing the DFT
 - 15.4 An Efficient FFT for $N = P \times Q$
 - 15.5 Multi-Dimensional Array as an Algorithmic Tool
 - 15.5.1 Storing a 1D array into a multi-dimensional array
 - 15.5.2 Row-oriented interpretation of ν -D arrays as 2D arrays
 - 15.5.3 Column-oriented interpretation of ν -D arrays as 2D arrays
 - 15.5.4 Row-oriented interpretation of ν -D arrays as 3D arrays
 - 15.5.5 Column-oriented interpretation of ν -D arrays as 3D arrays
 - 15.6 Programming Different ν -D Arrays From a Single Array
 - 15.6.1 Support from the FORTRAN programming language
 - 15.6.2 Further adaptation
 - 15.7 An Efficient FFT for $N = N_0 \times N_1 \times \dots \times N_{\nu-1}$
 - 15.8 Notes and References
-

16 Selected FFT Applications

- 16.1 Fast Polynomial Multiplication
 - 16.2 Fast Convolution and Deconvolution
 - 16.3 Computing a Toeplitz Matrix-Vector Product
 - 16.4 Computing a Circulant Matrix-Vector Product
 - 16.5 Solving a Large Circulant Linear System
 - 16.6 Fast Discrete Sine Transforms
 - 16.7 Fast Discrete Cosine Transform
 - 16.8 Fast Discrete Hartley Transform
 - 16.9 Fast Chebyshev Approximation
 - 16.10 Solving Difference Equations
-

III Parallel FFT Algorithms

17 Parallelizing the FFTs: Preliminaries on Data Mapping

- 17.1 Mapping Data to Processors
 - 17.2 Properties of Cyclic Block Mappings
 - 17.3 Examples of CBM Mappings and Parallel FFTs
-

18 Computing and Communications on Distributed-Memory Multiprocessors

- 18.1 Distributed-Memory Message-Passing Multiprocessors
- 18.2 The d -Dimensional Hypercube Multiprocessors
 - 18.2.1 The subcube-doubling communication algorithm
 - 18.2.2 Modeling the arithmetic and communication cost
 - 18.2.3 Hardware characteristics and implications on algorithm design

- 18.3 Embedding a Ring by Reflected-Binary Gray-Code
 - 18.4 A Further Twist-Performing Subcube-Doubling Communications on a Ring Embedded in a Hypercube
 - 18.5 Notes and References
 - 18.5.1 Arithmetic time benchmarks
 - 18.5.2 Unidirectional times on circuit-switched networks
 - 18.5.3 Bidirectional times on full-duplex channels
-

19 Parallel FFTs without Inter-Processor Permutations

- 19.1 A Useful Equivalent Notation: $\lfloor \text{PID} \rfloor_{\text{Local } M}$
 - 19.1.1 Representing data mappings for different orderings
 - 19.2 Parallelizing In-Place FFTs Without Inter-Processor Permutations
 - 19.2.1 Parallel DIF_{NR} and DIT_{NR} algorithms
 - 19.2.2 Interpreting the data mapping for bit-reversed output
 - 19.2.3 Parallel DIF_{RN} and DIT_{RN} algorithms
 - 19.2.4 Interpreting the data mapping for bit-reversed input
 - 19.3 Analysis of Communication Cost
 - 19.4 Uneven Distribution of Arithmetic Workload
-

20 Parallel FFTs with Inter-Processor Permutations

- 20.1 Improved Parallel DIF_{NR} and DIT_{NR} Algorithms
 - 20.1.1 The idea and a modified shorthand notation
 - 20.1.2 The complete algorithm and output interpretation
 - 20.1.3 The use of other initial mappings
 - 20.2 Improved Parallel DIF_{RN} and DIT_{RN} Algorithms
 - 20.3 Further Technical Details and a Generalization
-

21 A Potpourri of Variations on Parallel FFTs

- 21.1 Parallel FFTs without Inter-Processor Permutations
 - 21.1.1 The PID in Gray code
 - 21.1.2 Using an ordered FFT on local data
 - 21.1.3 Using radix-4 and split-radix FFTs
 - 21.1.4 FFTs for Connection Machines
 - 21.2 Parallel FFTs with Inter-Processor Permutations
 - 21.2.1 Restoring the initial map at every stage
 - 21.2.2 Pivoting on the right-most bit in local M
 - 21.2.3 All-to-all inter-processor communications
 - 21.2.4 Maintaining specific maps for input and output
 - 21.3 A Summary Table
 - 21.4 Notes and References
-

22 Further Improvement and a Generalization of Parallel FFTs

- 22.1 Algorithms with Specific Mappings for Ordered Output

- 22.1.1 Algorithm I
 - 22.1.2 Algorithm II
 - 22.2 A General Algorithm and Communication Complexity Results
 - 22.2.1 Phase I of the general algorithm
 - 22.2.2 Phase II of the general algorithm
-

23 Parallelizing Two-dimensional FFTs

- 23.1 The Computation of Multiple 1D FFTs
 - 23.2 The Sequential 2D FFT Algorithm
 - 23.2.1 Programming considerations
 - 23.2.2 Computing a single 1D FFT stored in a 2D matrix
 - 23.2.3 Sequential algorithms for matrix transposition
 - 23.3 Three Parallel 2D FFT Algorithms for Hypercubes
 - 23.3.1 The transpose split (TS) method
 - 23.3.2 The local distributed (LD) method
 - 23.3.3 The 2D block distributed method
 - 23.3.4 Transforming a rectangular signal matrix on hypercubes
 - 23.4 The Generalized 2D Block Distributed (GBLK) Method for Subcube-grids and Meshes
 - 23.4.1 Running hypercube (subcube-grid) programs on meshes
 - 23.5 Configuring an Optimal Physical Mesh for Running Hypercube (Subcube-grid) Programs
 - 23.5.1 Minimizing multi-hop penalty
 - 23.5.2 Minimizing traffic congestion
 - 23.5.3 Minimizing channel contention on a circuit-switched network
 - 23.6 Pipelining Subcube-doubling Communications on All Hypercube Channels
 - 23.7 Changing Data Mappings During Parallel 2D FFT Computation
 - 23.8 Parallel Matrix Transposition By Changing Data Mapping
 - 23.9 Notes and References
-

24 Computing and Distributing Twiddle Factors in the Parallel FFTs

- 24.1 Twiddle Factors for Parallel FFT Without Inter-Processor Permutations
 - 24.2 Twiddle Factors for Parallel FFT With Inter-Processor Permutations
-

IV Appendices

A Fundamental Concepts of Efficient Scientific Computation

- A.1 Time and Space Consumed by the DFT and FFT Algorithms
 - A.1.1 Relating operation counts to execution times
 - A.1.2 Relating MFLOPS to execution times and operation counts

- A.2 Comparing Algorithms by Orders of Complexity
 - A.2.1 An informal introduction via motivating examples
 - A.2.2 Formal notations and terminologies
 - A.2.3 The big-Oh and big-Omega notations
 - A.2.4 Some common uses of the Θ -notation

B Solving Recurrence Equations by Substitution

- B.1 Deriving Recurrences From a Known Function
 - B.2 Solving Recurrences to Determine an Unknown Function
 - B.3 Mathematical Summation Formulas
 - B.4 Solving Generalized Recurrence Equations
 - B.5 Recurrences and the Fast Fourier Transforms
-

Bibliography

Preface

The fast Fourier transform (FFT) algorithm, together with its many successful applications, represents one of the most important advancements in scientific and engineering computing in this century. The wide usage of computers has been instrumental in driving the study of the FFT, and a very large number of articles have been written about the algorithm over the past thirty years. Some of these articles describe modifications of the basic algorithm to make it more efficient or more applicable in various circumstances. Other work has focused on implementation issues, in particular, the development of parallel computers has spawned numerous articles about implementation of the FFT on multiprocessors. However, to many computing and engineering professionals, the large collection of serial and parallel algorithms remain hidden inside the FFT black box because: (1) coverage of the FFT in computing and engineering textbooks is usually brief, typically only a few pages are spent on the algorithmic aspects of the FFT; (2) cryptic and highly variable mathematical and algorithmic notation; (3) limited length of journal articles; and (4) important ideas and techniques in designing efficient algorithms are sometimes buried in software or hardware-implemented FFT programs, and not published in the open literature.

This book is intended to help rectify this situation. Our objective is to bring these numerous and varied ideas together in a common notational framework, and make the study of FFT an inviting and relatively painless task. In particular, the book employs a unified and systematic approach in developing the multitude of ideas and computing techniques employed by the FFT, and in so doing, it closes the gap between the often brief introduction in textbooks and the equally often intimidating treatments in the FFT literature. The unified notation and approach also facilitates the development of new parallel FFT algorithms in the book.

This book is self-contained at several levels. First, because the fast Fourier transform (FFT) is a fast “algorithm” for computing the discrete Fourier transform (DFT), an “algorithmic approach” is adopted throughout the book. To make the material fully accessible to readers who are not familiar with the design and analysis of computer algorithms, two appendices are given to provide necessary background. Second, with the help of examples and diagrams, the algorithms are explained in full. By exercising the appropriate notation in a consistent manner, the algorithms are explicitly connected to the mathematics underlying the FFT—this is often the “missing link” in the literature. The algorithms are presented in pseudo-code and a complexity analysis of each is provided.

Features of the book

- *The book is written to bridge the gap between textbooks and literature.* We believe this book is unique in this respect. The majority of textbooks largely focus on the underlying mathematical transform (DFT) and its applications, and only a small part is devoted to the FFT, which is a fast algorithm for computing the DFT.

- *The book teaches up-to-date computational techniques relevant to the FFT.* The book systematically and thoroughly reviews, explains, and unifies FFT ideas from journals across the disciplines of engineering, mathematics, and computer science from 1960 to 1999. In addition, the book contains several parallel FFT algorithms that are believed to be new.

- *Only background found in standard undergraduate mathematical science, computer science, or engineering curricula is required.* The notations used in the book are fully explained and demonstrated by examples. As a consequence, this book should make FFT literature accessible to senior undergraduates, graduate students, and computing professionals. The book should serve as a self-teaching guide for learning about the FFT. Also, many of the ideas discussed are of general importance in algorithm design and analysis, efficient numerical computation, and scientific programming for both serial or parallel computers.

Use of the book

It is expected that this book will be of interest and of use to senior undergraduate students, graduate students, computer scientists, numerical analysts, engineering professionals, specialists in parallel and distributed computing, and researchers working in computational mathematics in general.

The book also has potential as a supplementary text for undergraduate and graduate courses offered in mathematical science, computer science, and engineering programs. Specifically, it could be used for courses in scientific computation, numerical analysis, digital signal processing, the design and analysis of computer algorithms, parallel algorithms and architectures, parallel and distributed computing, and engineering courses treating the discrete Fourier transform and its applications.

Scope of the book

The book is organized into 24 chapters and 2 appendices. It contains 97 figures and 38 tables, as well as 25 algorithms presented in pseudo-code, along with numerous code segments. The bibliography contains more than 100 references dated from 1960 to 1999. The chapters are organized into three parts.

I. Preliminaries Part I presents a brief introduction to the discrete Fourier transform through a simple example involving trigonometric interpolation. This part is included to make the book self-contained. Some details about floating point arithmetic as it relates to FFT computation is also included in Part I.

II. Sequential FFT Algorithms This part contains fourteen relatively short chapters (3 through 16). Although the FFT, like binary search and quicksort, is commonly used in textbooks to illustrate the divide and conquer paradigm and recursive algorithms, the FFT has a unique feature: the application of the basic FFT algorithm

to “naturally ordered” input, if performed “in place,” yields output in “bit-reversed” order. While this feature may be taken for granted by FFT insiders, it is often not addressed in detail in textbooks. Again, partly because of the lack of notation linking the underlying mathematics to the algorithm, and because it is understood by FFT professionals, this aspect of the FFT is either left unexplained or explained very briefly in the literature. This phenomenon, its consequences, and how to deal with it, is one of the topics of Part II.

Similarly, the basic FFT algorithm is generally introduced as most efficient when applied to vectors whose length N is a power of two, although it can be made even more efficient if N is a power of four, and even more so if it is a power of eight, and so on. These situations, as well as the case when N is arbitrary, are considered in Part II. Other special situations, such as when the input is real rather than complex, and various programming “tricks,” are also considered in Part II, which concludes with a chapter on selected applications of FFT algorithms.

III. Parallel FFT Algorithms The last part deals with the many and varied issues that arise in implementing FFT algorithms on multiprocessor computers. Part III begins with a chapter that discusses the mapping of data to processors, because the designs of the parallel FFTs are mainly driven by data distribution, rather than by the way the processors are physically connected (through shared memory or by way of a communication network.) This is a feature not shared by parallel numerical algorithms in general.

Distributed-memory multiprocessors are discussed next, because implementing the algorithms on shared-memory architecture is straightforward. The hypercube multiprocessor architecture is particularly considered because it is so naturally compatible with the FFT algorithm. However, the material discussed later does not specifically depend on the hypercube architecture.

Following that, a series of chapters contains a large collection of parallel algorithms, including some that are believed to be new. All of the algorithms are described using a common notation that has been derived from one introduced in the literature. As in part II, dealing with the bit-reversal phenomenon is considered, along with balancing the computational load and avoiding communication congestion. The last two chapters deal with two-dimensional FFTs and the task of distributing the “twiddle factors” among the individual processors.

Appendix A contains basic information about efficient computation, together with some fundamentals on complexity notions and notation. Appendix B contains techniques that are helpful in solving recurrence equations. Since FFT algorithms are recursive, analysis of their complexity leads naturally to such equations.

Acknowledgments

This book resulted from our teaching and research activities at the University of Guelph and the University of Waterloo. We are grateful to both Universities for providing the environment in which to pursue these activities, and to the Natural Sciences and Engineering Research Council of Canada for our research support. At a personal level, Eleanor Chu owes a special debt of gratitude to her husband, Robert Hiscott, for his understanding, encouragement, and unwavering support.

We thank the reviewers of our book proposal and draft manuscript for their helpful suggestions and insightful comments which led to many improvements.

Our sincere thanks also go to Robert Stern (Publisher) and his staff at CRC Press for their enthusiastic support of this project.

Eleanor Chu
Guelph, Ontario
Alan George
Waterloo, Ontario

Part I

Preliminaries

Chapter 1

An Elementary Introduction to the Discrete Fourier Transform

This chapter is intended to provide a brief introduction to the discrete Fourier transform (DFT). It is not intended to be comprehensive; instead, through a simple example, it provides an illustration of how the computation that is the subject of this book arises, and how its results can be used. The DFT arises in a multitude of other contexts as well, and a dozen more DFT-related applications, together with information on a number of excellent references, are presented in Chapter 16 in Part II of this book. Readers familiar with the DFT may safely skip this chapter.

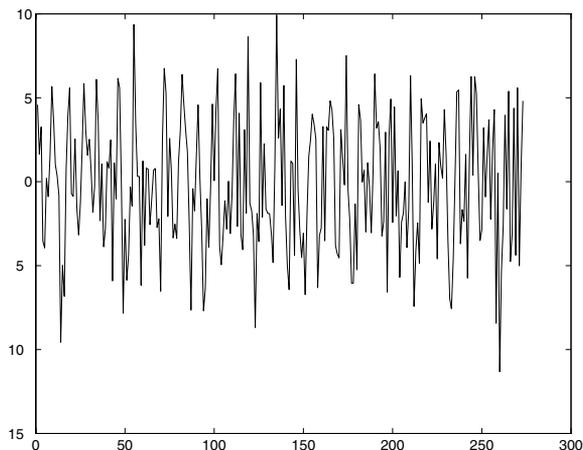
A major application of Fourier transforms is the analysis of a series of observations: $x_\ell, \ell = 0, \dots, N-1$. Typically, N will be quite large: 10000 would not be unusual. The sources of such observations are many: ocean tidal records over many years, communication signals over many microseconds, stock prices over a few months, sonar signals over a few minutes, and so on. The assumption is that there are repeating patterns in the data that form *part* of the x_ℓ . However, usually there will be other phenomena which may not repeat, or repeat in a way that is not discernably cyclic. This is called “noise.” The DFT helps to identify and quantify the cyclic phenomena. If a pattern repeats itself m times in the N observations, it is said to have *Fourier frequency* m .

To make this more specific, suppose one measures a signal from time $t = 0$ to $t = 680$ in steps of 2.5 seconds, giving 273 observations. The measurements might appear as shown in [Figure 1.1](#). How does one make any sense out of it? As shown later, the DFT can help.

1.1 Complex Numbers

Effective computation of the DFT relies heavily on the use of complex numbers, so it is useful to review their basic properties. This material is elementary and probably well-known to most readers of this book, but it is included for completeness. Historically, complex numbers were introduced to deal with polynomial equations, such as $x^2 + 1 =$

Figure 1.1 Example of a noisy signal.



0, which have no real solutions. Informally, they can be defined as the set \mathcal{C} of all “numbers” of the form $a + jb$ where a and b are real numbers and $j^2 = -1$.

Addition, subtraction, and multiplication are performed among complex numbers by treating them as binomials in the unknown j and using $j^2 = -1$ to simplify the result. Thus

$$(a + jb) + (c + jd) = (a + c) + j(b + d)$$

and

$$(a + jb) \times (c + jd) = (ac - bd) + j(ad + bc).$$

For the complex number $z = a + jb$, a is the *real part* of z and b is the *imaginary part* of z . The zero element of \mathcal{C} is $0 + 0i$, and the additive inverse of $z = a + jb$ is $-a + j(-b)$. The multiplicative inverse z^{-1} is

$$z^{-1} = \frac{a - jb}{a^2 + b^2}.$$

The complex conjugate of $z = a + jb$ is denoted by \bar{z} and is equal to $a - jb$. The *modulus* of z , denoted by $|z|$, is

$$\sqrt{z\bar{z}} = \sqrt{a^2 + b^2}.$$

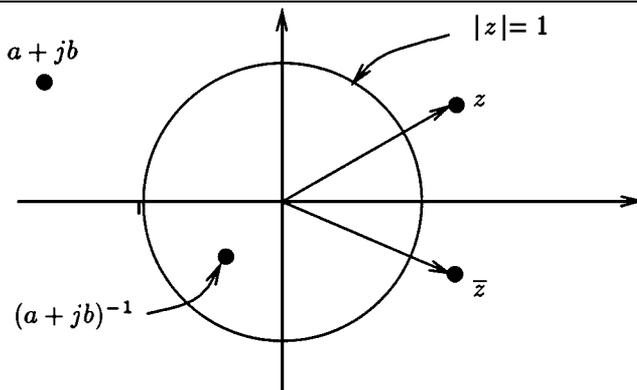
Some additional facts that will be used later are

$$e^z = e^{(a+jb)} = e^a e^{jb} \quad \text{and} \quad e^{jb} = \cos b + j \sin b.$$

Thus, $Re(e^z) = e^a \cos b$ and $Im(z) = e^a \sin b$.

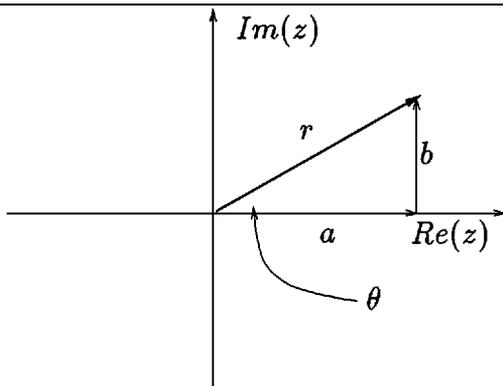
Just as a real number can be pictured as a point lying on a line, a complex number can be pictured as a point lying in a plane. With each complex number $a + jb$ one can associate a vector beginning at the origin and terminating at the point (a, b) . These notions are depicted in Figure 1.2.

Figure 1.2 Visualizing complex numbers.



Instead of the pair (a, b) , one can use the “length” (modulus) together with the angle the number makes with the real axis. Thus, $a + jb$ can be represented as $r \cos \theta + jr \sin \theta = re^{j\theta}$, where $r = |z| = \sqrt{a^2 + b^2}$ and $\theta = \arctan(b/a)$. This representation of a complex number is depicted in Figure 1.3.

Figure 1.3 Polar representation of a complex number.



Multiplication of complex numbers in polar form is straightforward: if $z_1 = a + jb = r_1 e^{j\theta_1}$ and $z_2 = c + jd = r_2 e^{j\theta_2}$, then

$$z_1 z_2 = r_1 r_2 e^{j(\theta_1 + \theta_2)}.$$

The moduli are multiplied together, and the angles are added. Note that if $z = e^{j\theta}$, then $|z| = 1$ for all values of θ .

1.2 Trigonometric Interpolation

Suppose a function $f(\theta)$ is defined on the interval $(0, 2\pi)$, with f assumed to be periodic on the interval; thus, $f(\theta) = f(\theta \pm 2\pi)$.

Now consider constructing a trigonometric polynomial $p(\theta)$ to interpolate $f(\theta)$ of the form

$$(1.1) \quad p(\theta) = a_0 + \sum_{k=1}^n a_k \cos k\theta + b_k \sin k\theta.$$

This function has $2n + 1$ coefficients, so it should be possible to interpolate f at $2n + 1$ points. In the applications considered in this book, the points at which to interpolate are always *equally spaced* on the interval:

$$(1.2) \quad \theta_\ell = \frac{2\ell\pi}{2n+1}, \quad \ell = 0, 1, \dots, 2n.$$

Let $x_\ell = f(\theta_\ell)$, and consider an example with $n = 2$. Then the interpolation conditions are $x_\ell = p(\theta_\ell)$, or

$$x_\ell = a_0 + a_1 \cos \theta_\ell + b_1 \sin \theta_\ell + a_2 \cos 2\theta_\ell + b_2 \sin 2\theta_\ell, \quad \ell = 0, 1, \dots, 4.$$

This leads to the system of equations

$$\begin{bmatrix} 1 & \cos \theta_0 & \sin \theta_0 & \cos 2\theta_0 & \sin 2\theta_0 \\ 1 & \cos \theta_1 & \sin \theta_1 & \cos 2\theta_1 & \sin 2\theta_1 \\ 1 & \cos \theta_2 & \sin \theta_2 & \cos 2\theta_2 & \sin 2\theta_2 \\ 1 & \cos \theta_3 & \sin \theta_3 & \cos 2\theta_3 & \sin 2\theta_3 \\ 1 & \cos \theta_4 & \sin \theta_4 & \cos 2\theta_4 & \sin 2\theta_4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ b_1 \\ a_2 \\ b_2 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}.$$

Recall that $e^{j\theta} = \cos \theta + j \sin \theta$, which implies that

$$\cos \theta = \frac{e^{j\theta} + e^{-j\theta}}{2} \quad \text{and} \quad \sin \theta = \frac{e^{j\theta} - e^{-j\theta}}{2j}.$$

Using these in (1.1) with $n = 2$ yields

$$\begin{aligned} p(\theta) &= a_0 + \left(\frac{a_1}{2}\right) e^{j\theta} + \left(\frac{a_1}{2}\right) e^{-j\theta} + \left(\frac{b_1}{2j}\right) e^{j\theta} - \left(\frac{b_1}{2j}\right) e^{-j\theta} \\ &\quad + \left(\frac{a_2}{2}\right) e^{2j\theta} + \left(\frac{a_2}{2}\right) e^{-2j\theta} + \left(\frac{b_2}{2j}\right) e^{2j\theta} - \left(\frac{b_2}{2j}\right) e^{-2j\theta} \\ &= \left(\frac{a_2 + j b_2}{2}\right) e^{-2j\theta} + \left(\frac{a_1 + j b_1}{2}\right) e^{-j\theta} \\ &\quad + a_0 + \left(\frac{a_1 - j b_1}{2}\right) e^{j\theta} + \left(\frac{a_2 - j b_2}{2}\right) e^{2j\theta}. \end{aligned}$$

Giving the coefficients names corresponding to the powers of $e^{j\theta}$ yields

$$(1.3) \quad p(\theta) = X_{-2}e^{-2j\theta} + X_{-1}e^{-j\theta} + X_0 + X_1e^{j\theta} + X_2e^{2j\theta}.$$

Note that the coefficients appear in complex conjugate pairs. When the x_ℓ are real, it is straightforward to show that this is true in general. (See the next section.)

Recall (see (1.2)) that the points at which interpolation occurs are *evenly spaced*; that is, $\theta_\ell = \ell\theta_1$. Let $\omega = e^{j\theta_1} = e^{\frac{2j\pi}{2n+1}}$. Then all $e^{j\theta_\ell}$ can be expressed in terms of ω :

$$e^{j\theta_\ell} = e^{j\ell\theta_1} = \omega^\ell, \quad \ell = 0, 1, \dots, 2n.$$

Also, note that $\omega^\ell = \omega^{\ell \pm (2n+1)}$ and $\omega^{-\ell} = \omega^{-\ell \pm (2n+1)}$. For the example with $n = 2$, $\omega = e^{\frac{2i\pi}{5}}$, and the interpolation condition at θ_ℓ in (1.3) is

$$f(\theta_\ell) = x_\ell = p(\theta_\ell) = X_{-2}\omega^{-2\ell} + X_{-1}\omega^{-\ell} + X_0\omega^0 + X_1\omega^\ell + X_2\omega^{2\ell}.$$

Using the fact that $\omega^{-\ell} = \omega^{(2n+1-\ell)}$, and renaming the coefficients similarly ($X_{-\ell} \rightarrow X_{2n+1-\ell}$), the interpolation condition at x_ℓ becomes

$$x_\ell = X_0 + X_1\omega^\ell + X_2\omega^{2\ell} + X_3\omega^{3\ell} + X_4\omega^{4\ell},$$

which has to be satisfied for $\ell = 0, 1, \dots, 4$:

$$(1.4) \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega^8 \\ 1 & \omega^3 & \omega^6 & \omega^9 & \omega^{12} \\ 1 & \omega^4 & \omega^8 & \omega^{12} & \omega^{16} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}.$$

This can be written as a matrix equation

$$\mathbf{MX} = \mathbf{x}.$$

It will be useful to have some additional properties of ω . First note that

$$1 + \omega + \omega^2 + \dots + \omega^{2n} = 0.$$

This can be established by observing that the expression on the left side is a geometric sum equal to

$$\frac{1 - \omega^{2n+1}}{1 - \omega},$$

and this quantity is zero because $\omega^{2n+1} = 1$. For integers r and s one can show in a similar way that

$$(1.5) \quad \sum_{k=0}^{2n} \omega^{(kr-ks)} = \begin{cases} 0 & \text{if } r \neq s \\ 2n + 1 & \text{if } r = s. \end{cases}$$

These simple results make solving $\mathbf{MX} = \mathbf{x}$ easy. To begin, let

$$\bar{\mathbf{M}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \bar{\omega} & \bar{\omega}^2 & \bar{\omega}^3 & \bar{\omega}^4 \\ 1 & \bar{\omega}^2 & \bar{\omega}^4 & \bar{\omega}^6 & \bar{\omega}^8 \\ 1 & \bar{\omega}^3 & \bar{\omega}^6 & \bar{\omega}^9 & \bar{\omega}^{12} \\ 1 & \bar{\omega}^4 & \bar{\omega}^8 & \bar{\omega}^{12} & \bar{\omega}^{16} \end{bmatrix}.$$

Then using (1.5) above, together with the fact that $\bar{\omega}^\ell = \omega^{-\ell}$, shows that $\bar{\mathbf{M}}\mathbf{M}$ is

$$\begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}.$$