


OLIVER ABERTH



Introduction to  
Precise Numerical  
Methods

2ND EDITION



# **Introduction to Precise Numerical Methods**



This page intentionally left blank

# Introduction to Precise Numerical Methods



**Oliver Aberth**

*Mathematics Department  
Texas A & M University*



ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON  
NEW YORK • OXFORD • PARIS • SAN DIEGO  
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Academic Press is an imprint of Elsevier



Acquisitions Editor Tom Singer  
Project Manager Jay Donahue  
Marketing Manager L Leah Ackerson  
Cover Design Eric DeCicco  
Composition Integra Software Services Pvt. Ltd., India  
Cover Printer Phoenix Color  
Interior Printer Sheridan Books

Academic Press is an imprint of Elsevier  
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA  
525 B Street, Suite 1900, San Diego, California 92101-4495, USA  
84 Theobald's Road, London WC1X 8RR, UK

This book is printed on acid-free paper. ☺

Copyright © 2007, Elsevier Inc. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone (+44) 1865 843830, fax (+44) 1865 853333, email: [permissions@elsevier.com](mailto:permissions@elsevier.com). You may also complete your request on-line via the Elsevier homepage (<http://elsevier.com>), by selecting "Support & Contact" then "Copyright and Permission" and then "Obtaining Permissions."

#### Library of Congress Cataloging-in-Publication Data

Aberth, Oliver.

Introduction to precise numerical methods/Oliver Aberth.

p. cm.

ISBN 0-12-373859-8

1. Computer science—Mathematics. 2. Numerical analysis—Data processing. I. Title.

QA76.9.M35A24 2007

518.0285—dc22

2007000712

#### British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

ISBN 13: 978-0-12-373859-2

ISBN 10: 0-12-373859-8

For information on all Academic Press publications  
visit our website at [www.books.elsevier.com](http://www.books.elsevier.com)

Printed in the United States of America

07 08 09 10 9 8 7 6 5 4 3 2 1

Working together to grow  
libraries in developing countries

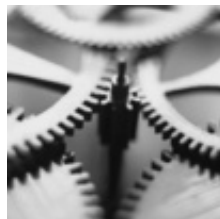
[www.elsevier.com](http://www.elsevier.com) | [www.bookaid.org](http://www.bookaid.org) | [www.sabre.org](http://www.sabre.org)

ELSEVIER

BOOK AID  
International

Sabre Foundation

# Contents



**Preface xi**

**Acknowledgments xiii**

## **1 Introduction 1**

- 1.1 Open-source software 1
- 1.2 Calling up a program 2
- 1.3 Log files and print files 3
- 1.4 More on log files 4
- 1.5 The tilde notation for printed answers 5

## **2 Computer Arithmetics 9**

- 2.1 Floating-point arithmetic 9
- 2.2 Variable precision floating-point arithmetic 10
- 2.3 Interval arithmetic 11
- 2.4 Range arithmetic 13
- 2.5 Practical range arithmetic 15
- 2.6 Interval arithmetic notation 15
- 2.7 Computing standard functions in range arithmetic 17
- 2.8 Rational arithmetic 18
  - Software Exercises A 20
  - Notes and References 23

## **3 Classification of Numerical Computation Problems 25**

- 3.1 A knotty problem 25
- 3.2 The impossibility of untying the knot 27

- 3.3 Repercussions from nonsolvable problem 3.1 27
- 3.4 Some solvable and nonsolvable decimal place problems 29
- 3.5 The solvable problems handled by `calc` 32
- 3.6 Another nonsolvable problem 32
- 3.7 The trouble with discontinuous functions 33
- Notes and References 35

## 4 Real-Valued Functions 37

- 4.1 Elementary functions 37
- Software Exercises B 39

## 5 Computing Derivatives 41

- 5.1 Power series of elementary functions 41
- 5.2 An example of series evaluation 48
- 5.3 Power series for elementary functions of several variables 49
- 5.4 A more general method of generating power series 52
- 5.5 The demo program `deriv` 54
- Software Exercises C 54
- Notes and References 54

## 6 Computing Integrals 57

- 6.1 Computing a definite integral 57
- 6.2 Formal interval arithmetic 59
- 6.3 The demo program `integ` for computing ordinary definite integrals 61
- 6.4 Taylor's remainder formula generalized 63
- 6.5 The demo program `mulint` for higher dimensional integrals 64
- 6.6 The demo program `impint` for computing improper integrals 66
- Software Exercises D 67
- Notes and References 68

## 7 Finding Where a Function $f(x)$ is Zero 69

- 7.1 Obtaining a solvable problem 69
- 7.2 Using interval arithmetic for the problem 72
- 7.3 Newton's method 73
- 7.4 Order of convergence 75
- Software Exercises E 77

## 8 Finding Roots of Polynomials 79

- 8.1 Polynomials 79
- 8.2 A bound for the roots of a polynomial 85
- 8.3 The Bairstow method for finding roots of a real polynomial 86
- 8.4 Bounding the error of a rational polynomial's root approximations 90
- 8.5 Finding accurate roots for a rational or a real polynomial 92
- 8.6 The demo program `roots` 95
  - Software Exercises F 95
  - Notes and References 96

## 9 Solving $n$ Linear Equations in $n$ Unknowns 97

- 9.1 Notation 97
- 9.2 Computation problems 98
- 9.3 A method for solving linear equations 100
- 9.4 Computing determinants 102
- 9.5 Finding the inverse of a square matrix 104
- 9.6 The demo programs `equat`, `r_equat`, and `c_equat` 105
  - Software Exercises G 106
  - Notes and References 107

## 10 Eigenvalue and Eigenvector Problems 109

- 10.1 Finding a solution to  $Ax = 0$  when  $\det A = 0$  110
- 10.2 Eigenvalues and eigenvectors 113
- 10.3 Companion matrices and Vandermonde matrices 118
- 10.4 Finding eigenvalues and eigenvectors by Danilevsky's method 122
- 10.5 Error bounds for Danilevsky's method 127
- 10.6 Rational matrices 134
- 10.7 The demo programs `eigen`, `c_eigen`, and `r_eigen` 135
  - Software Exercises H 136

## 11 Problems of Linear Programming 137

- 11.1 Linear algebra using rational arithmetic 137
- 11.2 A more efficient method for solving rational linear equations 140
- 11.3 Introduction to linear programming 141
- 11.4 Making the simplex process foolproof 145
- 11.5 Solving  $n$  linear interval equations in  $n$  unknowns 148



- 11.6 Solving linear interval equations via linear programming 152
- 11.7 The program `linpro` for linear programming problems 155
- 11.8 The program `i_equat` for interval linear equations 156
  - Software Exercises I 156
  - Notes and References 157

## 12 Finding Where Several Functions are Zero 159

- 12.1 The general problem for real elementary functions 159
- 12.2 Finding a suitable solvable problem 160
- 12.3 Extending the  $f(x)$  solution method to the general problem 163
- 12.4 The crossing parity 165
- 12.5 The crossing number and the topological degree 166
- 12.6 Properties of the crossing number 170
- 12.7 Computation of the crossing number 171
- 12.8 Newton's method for the general problem 175
- 12.9 Searching a more general region for zeros 176
  - Software Exercises J 178
  - Notes and References 180

## 13 Optimization Problems 181

- 13.1 Finding a function's extreme values 181
- 13.2 Finding where a function's gradient is zero 184
- 13.3 The demo program `extrema` 188
  - Software Exercises K 188
  - Notes and References 189

## 14 Ordinary Differential Equations 191

- 14.1 Introduction 191
- 14.2 Two standard problems of ordinary differential equations 193
- 14.3 Difficulties with the initial value problem 196
- 14.4 Linear differential equations 197
- 14.5 Solving the initial value problem by power series 198
- 14.6 Degree 1 interval arithmetic 201
- 14.7 An improved global error 205
- 14.8 Solvable two-point boundary-value problems 208
- 14.9 Solving the boundary-value problem by power series 210
- 14.10 The linear boundary-value problem 213
  - Software Exercises L 214
  - Notes and References 216

**15 Partial Differential Equations 217**

- 15.1 Partial differential equation terminology 217
- 15.2 ODE and PDE initial value problems 219
- 15.3 A power series method for the ODE problem 220
- 15.4 The first PDE solution method 223
- 15.5 A simple PDE problem as an example 227
- 15.6 A defect of the first PDE method 228
- 15.7 The revised PDE method with comparison computation 229
- 15.8 Higher dimensional spaces 230
- 15.9 Satisfying boundary conditions 231
  - Software Exercises M 232
  - Notes and References 233

**16 Numerical Methods with Complex Functions 235**

- 16.1 Elementary complex functions 235
- 16.2 The demo program `c_deriv` 237
- 16.3 Computing line integrals in the complex plane 237
- 16.4 Computing the roots of a complex polynomial 238
- 16.5 Finding a zero of an elementary complex function  $f(z)$  239
- 16.6 The general zero problem for elementary complex functions 242
  - Software Exercises N 245
  - Notes and References 247

**The Precise Numerical Methods Program PNM 248****Index 249**

This page intentionally left blank

# Preface



Now that powerful PCs and Macs are everywhere available, when solving a numerical problem, we should no longer be content with an indefinite answer, that is, an answer where the error bound is either unknown or a vague guess. This book's software allows you to obtain your numerical answers to a prescribed number of correct decimal places. For instance, one can compute a definite integral  $\int_a^b f(x) dx$  to a wide choice of correct decimal places.

The problems treated in this book are standard problems of elementary numerical analysis, including a variety of problems from the field of ordinary differential equations and one standard problem from the field of partial differential equations. Most programs allow you to choose the number of correct decimal places for a problem's solution, with the understanding that more correct decimals require more computer time.

Besides the availability of powerful computers, two other advances permit the easy generation of accurate numerical answers. One is the development of efficient methods for accurately bounding computation errors, stemming from Ramon Moore's invention of interval arithmetic in 1962. The other is the development of methods for analyzing computation tasks, stemming from Alan Turing's groundbreaking work in the 1930s.

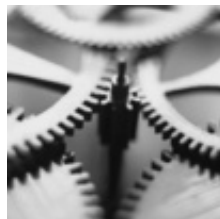
The CD that comes with this book contains a set of demonstration programs that will run on any PC using the Microsoft Windows XP operating system. Page 248 explains how to load the demonstration programs onto your PC's hard disk. After you follow those directions and read the short first chapter, you are ready to use any program. A beginning numerical analysis student can use this software to solve numerical problems that arise in the student's other science or engineering courses.

The text gives the mathematics behind the various numerical techniques and also describes in general terms the procedures followed by the various computation programs. The software is open-source; that is, the source code for each

computation program is available for inspection. Thus a student is able, when conversant with programming languages, to adapt these programs to other uses.

Chapters 1 through 15 can be read by a student who has completed the calculus sequence and an elementary linear algebra course. The final chapter, Chapter 16, requires some acquaintance with complex analysis.

# Acknowledgments



In the writing of this book and the creation of the accompanying software, I have had help from many sources. Two people have made fundamental contributions. Mark J. Schaefer, formerly of Tübingen University, helped write some of the computation programs. His brilliant programming skills were much needed, and it was his idea to identify quantities correct to the last decimal place with a terminal tilde. Ramon Moore of Ohio State University, who made precise numerical computation possible, has been supportive through many decades and helped test the various computation programs.

Rudolf Lohner of Karlsruhe University showed me how to improve my treatment of ordinary differential equations, by using his ingenious computation methods. R. Baker Kearfott of the University of Southwestern Louisiana helped me understand the crossing number concept.

I am indebted to Brian Hassard of SUNY at Buffalo, for his inspiring early attempts, with his students, at precise computation of specific partial differential equation problems. His experiments encouraged me to develop the `pde` program, described in Chapter 15.

I also wish to express my thanks to Grady Wright of the University of Utah, who carefully read the early manuscript, corrected some errors, and made many valuable suggestions for improvement.

Three reviewers of the early text, Gus Jacobs of Brown University, Arnd Scheel of the University of Minnesota, and Sylvia Veronese of the University of Utah, improved the book in many ways. I greatly appreciate their time and thoughtful comments.

Oliver Aberth

This page intentionally left blank

# Introduction



The programs that come with this book not only obtain numerical approximations, but also bound the errors, and in this way are able to display answers correct to the last decimal digit. This first chapter provides the information needed to use the software easily and to understand any numerical results obtained. The next section gives some background for the software, the three sections after that describe how to use the software, and the last section illustrates how numerical approximations are displayed and how these displays should be interpreted.

## 1.1 Open-source software

Because precision in numerical computation is still a novelty, we thought it important to provide the code for every computation program. To keep the source code relatively simple, all computation programs are MS-DOS programs instead of Windows programs. The successive Windows operating systems all allow a user to run an MS-DOS program via a command subsystem.

Our Windows program **PNM** lets you avoid extensive keyboard typing, as was necessary in the MS-DOS era. We need to review the fundamentals of how to call up a program using the command subsystem of Windows.

In general, a command line, entered at the computer keyboard, specifies two files and has the form

```
name1 name2
```

Here `name1` specifies a hard disk file, `name1.exe`, containing the computer execution instructions. (Each hard disk filename has a three letter file extension that is separated from the main part of its name by a period.) A following



`name2` may not be present in the command line, but if present, it specifies some additional hard disk file containing information needed by the executing program. With our command lines, the `name2` file extension is always `log`, so when `name2` is present in a command line, the file `name2.log` holds the needed data.

## 1.2 Calling up a program

As a simple example problem, which will be solved in detail in this section, suppose we require the solution of the two linear equations

$$x_1 + x_2 = 1$$

$$x_1 - x_2 = 2$$

You can become familiar with controlling the software by imitating the following steps on your PC.

We need to call up an appropriate program to solve this problem, and we suppose that either we do not know the name of the program or have forgotten it. In this situation, call up the general program `problem`. That is, click on **PNM** in your Windows “Programs” display, and after the **PNM** form appears, click on the **Command** menu, then click on the **Exe part** subsection, and finally, choose the `problem` program from the list of programs that appears.

The **PNM** form caption now will be “Command: problem”. Next click on the **Command** menu again, and this time click on the **Go** subsection. The **PNM** form will disappear, and the next step is to get to the Windows command subsystem (review page 248), type just the single letter **g** (for “go”) and hit the ENTER key.

The program `problem` will display various options and, according to your responses, eventually displays the name of an appropriate computation program. To solve our simple example, we first enter the integer 6, followed by the integer 1. The program `problem`, in response to these entries, displays the program name “`equat`”.

Now, knowing the program name, the next step is to call it up. We need to exit the Windows command subsystem, and this can always be done by entering the letter **e** (for “exit”) and hitting the ENTER key.

Once more, click on **PNM** in your Windows “Programs” display, and after the **PNM** form appears, click on the **Command** menu, then click on the **Exe part** subsection, and choose `equat` from the list of programs. The displayed caption changes to “Command: equat”. Next click on the **Command** menu again, and click on the **Go** subsection. Again the **PNM** form disappears, and once more we need to get to the Windows command subsystem, type the letter **g** and hit the ENTER key.

We now see a message identifying `equat` as a program for solving  $n$  linear equations in  $n$  unknowns. This program requires a user to view simultaneous

equations in the matrix–vector form  $AX = B$ , so let us recast our simple problem into that form:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Specify the number of equations by entering 2, and then enter the four coefficient matrix values of 1, 1, 1, and  $-1$ , followed by the two vector entries of 1 and 2. Then select the number of decimal places, say 10, by entering the integer 10. The solution is now displayed to 10 decimal places.

### 1.3 Log files and print files

Most, but not all, of the computation programs create both a log file and a print file. If a hypothetical program `abc` creates a log file, then the file `abc.log` will be found alongside the hard disk file holding the `abc` execution code (which would be `abc.exe`) as soon as the program `abc` obtains from you all the data needed to completely specify your computation problem, and before the program `abc` starts a solution run. The `abc.log` file lists each keyboard line you entered, with a description of what the entered line controls. A log file makes it easy to modify the problem for another `abc` run, because you need only change the `abc.log` file appropriately (using the **PNM** form to do this), and then give the command `abc abc` instead of the command `abc`. Whenever there are two names in a command line that are separated by one or more spaces, the second name designates a log file that defines the problem. Thus with the command line `abc abc`, the program `abc` (held in the file `abc.exe`) does not request keyboard entries. Instead it uses the file `abc.log` to specify the problem.

If the program `abc` creates a print file, the file `abc.prt`, containing a summary of the problem with a list of the answers obtained, will be found alongside the file `abc.exe`, after the program `abc` completes a solution run on a problem. The file `abc.prt` can be sent to your PC's printer to obtain a record of the problem's solution. The **PNM** form will also do this task.

We now return to the simple example of the preceding section, which we presume has just been solved by using the program `equat`. To see the `equat.prt` file, obtain the **PNM** form, click on the **Prt** menu, and then click on the **Open** subsection. The **PNM** form now holds the contents of the `equat.prt` file, although a part is obscured. Click on the right side of the **PNM** form and extend it so that the complete contents of the print file are in view. When the **Print** subsection of the **Prt** menu is selected, your printer copies whatever is visible in the **PNM** form, so before printing, it is important to adjust the **PNM** form size in both dimensions appropriately.

To see the `equat.log` file, first click on the **Log** menu, then click on the **Open** subsection, and finally click on the single line labeled `equat`. The **PNM** form now holds the contents of the `equat.log` file.

Let us suppose that immediately after we obtain the solution of our initial example problem, we find we need to solve the related problem

$$x_1 + x_2 = 3$$

$$x_1 - x_2 = 4$$

Here the equation right side values have been changed from their previous values to 3 and 4. Edit the log file to specify this new problem by changing the two vector values from 1 and 2 to their new values of 3 and 4, and then click on the **Save** subsection of the **Log** menu.

Our new problem can be solved now by clicking on the **Log** part of the **Command** menu, then clicking on the single line labeled `equat`. The **PNM** caption changes to “Command: `equat equat`”. Next click on the **Go** subsection of the **Command** menu, and, as usual, go to the **Windows** command subsection, type a **g** and hit the **ENTER** key. The solution to our new problem is now displayed.

## 1.4 More on log files

This section need be read only if you repeatedly use a particular program to solve a collection of related problems. We continue to use `abc` as the name of a hypothetical program creating a log file. The reader can think of `abc` as being a computation program (like `equat`) used earlier to solve some problem.

If the `abc.log` file already exists and you give the one word command `abc`, then after you specify the computation problem, the `abc.log` file is cleared and refilled with the new problem’s keyboard lines. An existing `abc.log` file can be saved by being renamed. This way the file is not cleared by an `abc` command, and the renamed file can still be used as a problem specifier.

To rename the `abc` log file, the **PNM** form caption must be either “Command `abc`” or “Command `abc abc`”. If this is not the case, click on the **Command** menu, then on the **Exe part** subsection, and choose `abc` from the list of programs. Now with the needed **PNM** form caption, click on the **Log** menu, then on the **Open** subsection and choose the `abc` log file from the list of log files. The **PNM** form now displays the log file. Next click on the **Log** menu a second time, and then on the **Save As** subsection. There is now a request for an addend to `abc` to generate a new log file name. Thus if you specify the addend as 1, the `abc.log` file is renamed `abc1.log`. Later, when you want to rerun the previous `abc` problem, give the command `abc abc1`.

Any alphabetic or numeric characters can be appended to `abc` to make up a new log file name. Thus `abc123` or `abcxyz` are both acceptable new log file names.

## 1.5 The tilde notation for printed answers

The number of decimal places to which an answer is computed is set by you, the program caller, and the decimals usually can be specified as either fixed-point or scientific floating-point. Let us suppose that three fixed-point decimal places are requested. It is possible with this decimal place choice that a computed answer is displayed this way:

$$111.234\sim$$

The tilde ( $\sim$ ) indicates that the displayed result has a positive error bound. Nevertheless, the displayed result is correct to the last decimal place shown. Section 3.4 has a discussion of the meaning of the phrase “correct to the last decimal place”, but this can be understood here to mean that the magnitude of the error is no larger than one-half of a unit in the last decimal place, or 5 units in the decimal place that would follow the last digit displayed. Thus, for the sample answer just shown, 0.0005 is the error bound on the answer. The tilde may be mentally converted to  $\pm\frac{1}{2}$  and so this particular answer also may be interpreted as

$$111.234 \pm \frac{1}{2}$$

Here the displayed  $\frac{1}{2}$  is of course to be associated with the terminal digit 4 of the answer.

Occasionally, when  $k$  fixed-point decimal digits are requested, an answer may appear showing  $k + 1$  decimal digits after the decimal point. Whenever an extra decimal place appears, the extra decimal digit is always a **5**. Thus, continuing with our supposition that three fixed-point decimal digits are requested, it is possible that an answer might appear this way

$$111.2345\sim$$

Section 3.4 explains why it is necessary sometimes to give an answer to one more decimal place than requested.

More rarely, when  $k$  fixed-point decimal digits are requested, an answer may appear to  $k$  decimal places, but without the tilde. The lack of a tilde indicates that the displayed answer has a zero error bound, and accordingly the answer is exact. For instance, continuing with our three fixed-point decimal supposition, an answer might appear this way:

$$111.234$$

In this case the answer is exactly the rational number 111.234. As an example, if we call up the `calc` program to obtain a three fixed-point decimal approximation to  $\cos(0)$ , the exact answer obtained appears as shown below:

1.000

Suppose now that you decide that scientific floating-point decimals are more convenient for representing answers. A  $k$  decimal scientific floating-point number is a number in the form

$$(\text{sign}) d_0.d_1d_2 \dots d_k \cdot 10^e \tag{1.1}$$

with the requirement that the exponent  $e$  be an integer, and that the leading decimal digit  $d_0$  not be zero. For instance,  $-3.4444 \cdot 10^{12}$  is a four decimal scientific floating-point value, but  $-3.4444 \cdot 10^{\sqrt{2}}$  or  $0.4444 \cdot 10^{12}$  are not. The leading term ‘(sign)  $d_0.d_1d_2 \dots d_k$ ’ is the *mantissa* and the trailing factor ‘ $10^e$ ’ is the *exponent part* of the number. With scientific floating-point notation, it is permissible to indicate that a number is zero by simply displaying a zero, that is, displaying 0 without any exponent part.

When a computation program is directed to obtain answers to  $k$  floating-point decimal places, the exponent part of an answer is shown using “E notation.” Thus if the exponent part is  $10^{-3}$ , this is displayed as E-3. With mantissas, the tilde notation is used. Thus the display  $2.1234\sim E5$  indicates that the mantissa is correct to the last digit, and that the error of the mantissa is no larger than one-half a unit in the last displayed decimal place. If a tilde does not appear in a floating-point mantissa, the mantissa has a zero error bound and is exact. As an example, if we direct the `calc` program to print out a four decimal floating-point approximation to  $\cos(0)$ , the exact answer appears this way:

1.0000 E0

A four decimal floating-point approximation to  $\sin(0)$  appears as

0

and illustrates the point that exact zeros appear as a single digit 0 without an exponent part. A four decimal floating-point approximation to  $\tan(\pi/4)$  appears this way:

1.0000 $\sim$  E0

Note that the tilde appears even though the tangent of  $\pi/4$  is exactly 1. Whenever a tilde appears with a displayed answer known to be exact, it means merely that the accompanying error bound computation yielded a positive result.

As explained in Section 3.4, computed error bounds serve to ensure the correctness of displayed answer digits. A computed error bound is a bound guaranteed not to be too small, but it is not necessarily exact.

With floating-point display, as explained in Section 3.4, sometimes no mantissa digits can be obtained. A computed approximation with a computed error bound defines an interval on the real number line, and if this interval contains the zero point, then a mantissa correct to the last decimal place is not forthcoming. If the interval obtained is actually a point, that is, the approximation is zero and the error bound is zero, then the display of 0 in place of a mantissa and an exponent part is correct. But for intervals that are not points, the failure to obtain a mantissa is indicated by the display of  $0.\tilde{\phantom{0}}$  for the mantissa, with an accompanying negative exponent of magnitude equal to or greater than the number of floating-point decimal places requested. Thus if we request four floating-point decimal places, the display shown below might occur:

$$0.\tilde{\phantom{0}} E - 4$$

The leading 0 signals the failure to obtain a mantissa (because a floating-point mantissa should start with a nonzero decimal digit), and the tilde after the decimal point signals that the displayed 0 is not necessarily exact. The interval containing the zero point in which the answer lies is obtained by interpreting the tilde in the usual way to mean plus or minus one-half a unit in the last decimal place displayed, or 5 units in the decimal place just beyond the last displayed one. For the answer just shown, the interval indicated is  $(0 \pm 0.5) \cdot 10^{-4}$ . The larger the number of requested floating-point digits, the smaller the displayed intervals containing the zero point must be, because the magnitude of the negative exponent must match or exceed the number of requested decimal places.

This page intentionally left blank

# Computer Arithmetics



Suppose our computation problem is a simple one: to compute various mathematical constants to a prescribed number of correct decimal digits. Typical constants might be  $e^{\sqrt{2}}$  or  $\tan 31^\circ$ . Two demonstration programs (called demo programs from now on) do precisely this task. The demo program `calc` computes real constants and the program `r_calc` computes rational constants. This chapter gives background information needed to understand how these programs operate.

In this chapter we describe a computer arithmetic called range arithmetic, which automatically generates answers with an error bound. With this arithmetic, it is easy to compute constants correct to a prescribed number of decimal places. Because range arithmetic is a variety of interval arithmetic, interval arithmetic must be described first. But before we do that, let us first describe the usual computer arithmetic, provided by all the common programming languages.

## 2.1 Floating-point arithmetic

In this system, as commonly used today, each floating-point number is assigned a fixed block of computer memory of several allowed sizes, and the programmer chooses the size. Usually a programming language allows at least two floating-point sizes. The number stored in memory has the form

$$(\text{sign}) .d_1 d_2 \dots d_k \cdot B^e$$

The leading digit  $d_1$  is unequal to zero,  $e$  is the exponent, and  $B$  is the base used for the exponent part. Here various number systems can be used in this representation: binary, octal, decimal, or hexadecimal. Accordingly, the base  $B$



could be 2, 8, 10, or 16. The range arithmetic system uses the decimal system, so we will concentrate on decimal floating-point representations, which have the form

$$(\text{sign}) .d_1d_2 \dots d_k \cdot 10^e \quad (2.1)$$

This representation is similar to what was called scientific floating-point in Chapter 1, except that the decimal point precedes rather than follows the first digit of the mantissa. This convention makes multiplication and division a little easier to execute on the computer. The mantissa now is  $(\text{sign}) .d_1d_2 \dots d_k$ . A certain amount of the assigned memory space is for the exponent  $e$ . Let us suppose that enough bits are allocated so that an exponent can vary between plus or minus 9,999, because from computational experience this is about as much variation as is ever needed. The mantissa digits take up most of the memory space, with the number of digits,  $k$ , depending on the floating-point size we choose. For example, we may have three choices available, giving us 6, 14, or 25 mantissa digits. When two numbers of a particular size are added, subtracted, or multiplied, the mantissa needed for an exact computed result may be longer than the particular size allows. Digits then must be discarded to make the result fit into the memory space reserved for the mantissa.

Floating-point arithmetic is obtained either by programming the computer to do it (a software floating-point), or having it done by a silicon chip (a hardware floating-point). A drawback of this conventional system is that the size must be chosen before a program is run. This precludes having the program choose the size, perhaps after doing a sample computation. Let us pass then to a more flexible system.

## 2.2 Variable precision floating-point arithmetic

With this arithmetic the format of a number, shown here,

$$(\text{sign}) .d_1d_2 \dots d_n \cdot 10^e \quad (2.2)$$

is identical to that given previously, except that  $n$ , the number of mantissa digits, now is variable instead of being fixed at the few choices previously allowed. Such a system will have an upper bound,  $N$ , on the number of mantissa digits possible, with  $N$  typically being several thousand. Thus any number of the form just shown is allowed if  $n$  is in the range  $1 \leq n \leq N$ . Because  $n$  is not fixed, the amount of memory needed for a number is not known in advance of forming the number. The storage of a number in memory now requires a variable number of bytes to hold the mantissa. The integer  $n$ , stored in a designated initial part of a number's memory block, indicates how many of the following memory bytes hold mantissa digits.

This system is now able to correctly represent even a very long input decimal constant. We simply convert the number to the form (2.2) with  $n$  just large enough to represent all digits. Thus program constants, such as 2.75 or 2222211111, obtain the forms  $+.275 \cdot 10^1$  or  $+.2222211111 \cdot 10^{10}$ , respectively. When performing divisions with such numbers, we must decide how many mantissa digits to form. And when doing additions, subtractions, or multiplications, generating an exact result may require an inconveniently large number of mantissa digits. A control parameter is needed, bounding the length of all mantissas formed by arithmetic operations. This control parameter, always a positive integer, can be given the name `PRECISION`. Thus `PRECISION` can be set to any value between a default minimum and the system upper bound  $N$ .

Before any computation, we set `PRECISION` to some value that seems appropriate, and if the computation reveals that more mantissa digits are needed, then we increase `PRECISION` and repeat the computation. Note that `PRECISION` does not determine the mantissa length for any decimal constants used by a program. Such constants always are converted to floating-point form with a mantissa just long enough to represent them exactly.

This system is clearly an improvement over the previous floating-point system, but there still is a major drawback. We usually cannot tell from our computed numbers how accurate our results are, especially after a long and complicated computation, so we do not know when to increase `PRECISION`. We need some way of determining how big our computational error is.

## 2.3 Interval arithmetic

In 1962 Ramon Moore proposed a way for the computer to keep track of errors. Instead of using a single computer representation for a number, two would be used, one to represent the number, and another to represent the maximum error of the number. First note that a number's "error" is defined as the *absolute value* of the number's deviation from the true value, so a number's error is always positive or zero. Each number now has a representation  $m \pm \epsilon$ , where  $m$  is a computer floating-point representation of the number, and  $\epsilon$  is the computer representation for the error. Program constants that are exactly represented by a computer number have  $\epsilon$  set to zero. As the computer does each arithmetic operation of addition, subtraction, multiplication, and division, besides forming the result  $m$ , it forms the result's error bound  $\epsilon$ . The mathematical relations used for the four arithmetic operations are the following:

$$m_1 \pm \epsilon_1 + m_2 \pm \epsilon_2 = (m_1 + m_2) \pm (\epsilon_1 + \epsilon_2) \quad (2.3)$$

$$m_1 \pm \epsilon_1 - m_2 \pm \epsilon_2 = (m_1 - m_2) \pm (\epsilon_1 + \epsilon_2) \quad (2.4)$$

$$m_1 \pm \epsilon_1 \times m_2 \pm \epsilon_2 = (m_1 m_2) \pm (\epsilon_1 |m_2| + |m_1| \epsilon_2 + \epsilon_1 \epsilon_2) \quad (2.5)$$

$$m_1 \pm \epsilon_1 \div m_2 \pm \epsilon_2 = \begin{cases} \left( \frac{m_1}{m_2} \right) \pm \left( \frac{\epsilon_1 + \left| \frac{m_1}{m_2} \right| \epsilon_2}{|m_2| - \epsilon_2} \right) & \text{if } |m_2| > \epsilon_2 \\ \text{Division error} & \text{if } |m_2| \leq \epsilon_2 \end{cases} \quad (2.6)$$

The first two relations for addition and subtraction are easy to understand, because the result error bound is just the sum of the two operand error bounds. To verify the multiplication relation, multiply the two operands together to obtain four terms; one is the result value, and the other three are error terms. The result error bound is the sum of the magnitudes of these three error terms.

A division operation with intervals cannot succeed unless the divisor interval does not contain the zero point. Thus division is not possible if  $|m_2| \leq \epsilon_2$ . When division is possible, the result error bound does not depend on the signs of  $m_1$  or  $m_2$ , and to find this bound, we can restrict our attention to  $|m_1| \pm \epsilon_1 \div |m_2| \pm \epsilon_2$ . Here we see that the maximum error occurs either when the smallest possible numerator is divided by the largest possible denominator or when the largest possible numerator is divided by the smallest possible denominator. In the first case the error is

$$\begin{aligned} \frac{|m_1|}{|m_2|} - \frac{|m_1| - \epsilon_1}{|m_2| + \epsilon_2} &= \frac{|m_1|(|m_2| + \epsilon_2) - (|m_1| - \epsilon_1)|m_2|}{|m_2|(|m_2| + \epsilon_2)} \\ &= \frac{\epsilon_1 |m_2| + |m_1| \epsilon_2}{|m_2|(|m_2| + \epsilon_2)} \\ &= \frac{\epsilon_1 + \left| \frac{m_1}{m_2} \right| \epsilon_2}{|m_2| + \epsilon_2} \end{aligned}$$

In the other case the error is

$$\begin{aligned} \frac{|m_1| + \epsilon_1}{|m_2| - \epsilon_2} - \frac{|m_1|}{|m_2|} &= \frac{(|m_1| + \epsilon_1)|m_2| - |m_1|(|m_2| - \epsilon_2)}{|m_2|(|m_2| - \epsilon_2)} \\ &= \frac{\epsilon_1 |m_2| + |m_1| \epsilon_2}{|m_2|(|m_2| - \epsilon_2)} \\ &= \frac{\epsilon_1 + \left| \frac{m_1}{m_2} \right| \epsilon_2}{|m_2| - \epsilon_2} \end{aligned}$$

If  $\epsilon_2$  is positive, this second value for the error is greater, and is the error shown in (2.6).

With interval arithmetic we let the computer form an error bound as it forms each numerical value. We are repaid for this extra computation by always having rigorous error bounds for all our computed answers.

## 2.4 Range arithmetic

When interval arithmetic is combined with the variable precision system described in Section 2.2, we obtain the means of knowing when too few digits have been used in our computation. To the floating-point representation we add a single decimal digit  $r$  for the error, and obtain the representation

$$(\text{sign}) .d_1 d_2 \dots d_n \pm r \cdot 10^e \quad (2.7)$$

The digit  $r$  is the *range* digit, used to define the maximum error of the number, with the decimal significance of  $r$  identical to that of the last mantissa digit  $d_n$ . By adding  $r$  to  $d_n$ , or subtracting  $r$  from  $d_n$ , we form the endpoints of the mantissa *interval*. From the range digit we obtain the mantissa's error bound, by taking account of the range digit's mantissa position and ignoring the exponent part. When the mantissa error is multiplied by the exponent part,  $10^e$ , we obtain the number's error bound. Below, these two error values are given for some typical numbers.

Ranged number	Mantissa error bound	Number error bound
$+ .8888 \pm 9 \cdot 10^1$	$.0009 = 9 \cdot 10^{-4}$	$9 \cdot 10^{-3}$
$- .7244666 \pm 2 \cdot 10^{-2}$	$.0000002 = 2 \cdot 10^{-7}$	$2 \cdot 10^{-9}$
$+ .2355555555 \pm 3 \cdot 10^4$	$.0000000003 = 3 \cdot 10^{-10}$	$3 \cdot 10^{-6}$

When decimal constants are converted to this range arithmetic form, as before, the number of mantissa digits that appear is the minimum needed to fully represent the constant; and the range digit is 0 because the constant is without error. Thus the constant 2.5 would obtain the ranged form  $+ .25 \pm 0 \cdot 10^1$ . Ranged numbers like this one, with a zero range digit, are called *exact*.

When exact numbers now are added, subtracted, or multiplied, the result also is exact if the PRECISION bound is not overstepped. But if PRECISION is exceeded, then the mantissa gets truncated to the PRECISION length, and the number gets a range digit of 1, indicating a maximum error of one unit in the last mantissa digit. As before, the accuracy of the four rational operations is changed simply by changing PRECISION.

When an arithmetic operation is performed with one or both operands having a nonzero range digit, the result also gets a nonzero range digit, computed using

the interval relations of the preceding section. Generally the result mantissa error bound is computed first, then rounded up to a one-digit value to form the range digit. This fixes  $n$ , the number of mantissa digits. Then the result mantissa is formed to this length, and if this means some mantissa digits are discarded, then the result range digit is incremented by one.

As a computation proceeds, with the numbers now having nonzero range digits, the successive mantissas generally obtain increasing error bounds, and, accordingly, their length tends to decrease. In one respect this is helpful, in that suspect mantissa digits get discarded automatically. Shorter mantissas mean faster arithmetic operations, so usually the later part of a computation runs quicker than the earlier part. If the final results have too few mantissa digits to yield the required number of correct decimal places, then the whole computation is repeated at a higher PRECISION. The cases where this occurs are likely to be cases where ordinary floating-point computation would yield inaccurate results. The amount to increase PRECISION may be determined by examining each answer that was insufficiently accurate, to see by how many digits it failed to meet its target form. If the greatest deficit in correct decimal places equals  $D$ , then PRECISION can be increased by  $D$ , plus some small safety margin perhaps, and a repetition of the computation at this higher precision will likely succeed.

To illustrate the general idea of range arithmetic, in Table 2.1 we show a sample range arithmetic computation to obtain the constant  $(5/4)^{32}$  to 5 correct fixed-point decimal places. The first computation is done with PRECISION set at 6, and the second with PRECISION set at 10. Each computation consists of five multiplications that are squarings of powers of  $5/4$ .

Of course not all computations are as simple as the one shown in Table 2.1, which is a straightforward evaluation from an initial constant to an answer. Usually the numerical solution of a mathematical problem is more complicated, and initial gross approximations to desired results are progressively refined. When we treat such cases we will need some method of determining the error of the final approximants.

**TABLE 2.1** Range arithmetic computation of  $(\frac{5}{4})^{32}$  by 5 squarings

Step	Step result	PRECISION = 6	PRECISION = 10
0	$\frac{5}{4}$	$+.125 \pm 0 \cdot 10^1$	$+.125 \pm 0 \cdot 10^1$
1	$(\frac{5}{4})^2$	$+.15625 \pm 0 \cdot 10^1$	$+.15625 \pm 0 \cdot 10^1$
2	$(\frac{5}{4})^4$	$+.244140 \pm 1 \cdot 10^1$	$+.244140625 \pm 0 \cdot 10^1$
3	$(\frac{5}{4})^8$	$+.596043 \pm 5 \cdot 10^1$	$+.5960464477 \pm 1 \cdot 10^1$
4	$(\frac{5}{4})^{16}$	$+.355267 \pm 6 \cdot 10^2$	$+.3552713678 \pm 2 \cdot 10^2$
5	$(\frac{5}{4})^{32}$	$+.126214 \pm 5 \cdot 10^4$	$+.1262177447 \pm 2 \cdot 10^4$

To 5 correct fixed-point decimal places,  $(\frac{5}{4})^{32} = 1262.17745\sim$

## 2.5 Practical range arithmetic

In the description of range arithmetic just given, we allowed the mantissa of a ranged number to change in length by a single decimal digit, and used a single decimal digit for the range. We presented this type of range arithmetic because it is easy to describe and easy to understand. When programming range arithmetic, however, one must take into account the number of mantissa digits that can be stored in `integer` elements, a basic type comprising a specific number of memory bytes, and a type that is available in almost all programming languages. Let us designate the number of decimal digits that can be stored in the type `integer`, as its *decimal width*. The decimal width depends on how many memory bytes of storage are allotted to the type `integer`. Let us suppose, for instance, that the decimal width is four. Because the mantissa digits are composed of `integer` units, the number of mantissa digits is always a multiple of the decimal width, and the number of digits in the range is also the decimal width.

To revise the range arithmetic system to accommodate an arbitrary decimal width of  $W$  digits, change each mantissa digit  $d_i$  and range digit  $r$  in the representation

$$(\text{sign}) .d_1 d_2 \dots d_n \pm r \cdot 10^e$$

from a single digit to a block of  $W$  decimal digits, and denote these by using capital letters in place of small letters. The exponent base, which was 10, is changed to  $B = 10^W$ , that is, 1 followed by  $W$  zeros. A ranged number now has the representation

$$[(\text{sign}) .D_1 D_2 \dots D_n \pm R] \cdot B^E$$

For example, with a decimal width of four, the constant 17.52 has the representation

$$[+(.0017)(5200) \pm (0000)] \cdot 10000^1$$

Thus a programmed range arithmetic system is essentially identical to the range arithmetic system described, except that base 10 with its 10 digits, 0 through 9, is replaced by a base with more digits, which, for a decimal width of 4, is a base with  $10^4 = 10,000$  different digits.

## 2.6 Interval arithmetic notation

There are two ways to designate intervals: with the notation  $m \pm \epsilon$  that we have been using, and with the endpoint notation  $[a, b]$ . One notation is easily converted into the other;  $m \pm \epsilon$  is also  $[m - \epsilon, m + \epsilon]$ , and  $[a, b]$  is also  $\frac{a+b}{2} \pm \frac{b-a}{2}$ . Most often we use the  $m \pm \epsilon$  notation, and we need names for the two parts. The

number  $m$  is the *midpoint* of the interval, and because the width of the interval is  $2\epsilon$ , the number  $\epsilon$  is the *halfwidth* of the interval.

For any mathematical constant, such as  $a$ , it is convenient to use an underline, as in  $\underline{a}$ , to denote the constant's range arithmetic interval approximation obtained at the current PRECISION.

Now consider two real numbers  $a$  and  $b$ , with range arithmetic intervals  $\underline{a}$  and  $\underline{b}$ . If the  $\underline{a}$  interval intersects the  $\underline{b}$  interval we indicate this by writing  $\underline{a} \dot{=} \underline{b}$ . For instance,  $2.22 \pm 2 \dot{=} 2.24 \pm 3$ . Here we count intervals as intersecting even if they just touch, so that  $2.22 \pm 1 \dot{=} 2.24 \pm 1$ . The new symbol  $\dot{=}$  for interval intersection is needed, and using  $=$  for this case would be misleading, because when  $\underline{a}$  intersects  $\underline{b}$ , any of the relations  $a < b$ ,  $a = b$ , or  $a > b$  may hold for the numbers  $a$  and  $b$ . We write  $\underline{a} > \underline{b}$  if the  $\underline{a}$  interval is to the right of the  $\underline{b}$  interval, and we write  $\underline{a} < \underline{b}$  if the  $\underline{a}$  interval is to the left of the  $\underline{b}$  interval. Thus  $3.33 \pm 1 > 3.30 \pm 1$  and  $3.33 \pm 1 < 3.36 \pm 1$ . We always have exactly one of three possibilities:  $\underline{a} < \underline{b}$ ,  $\underline{a} \dot{=} \underline{b}$ , or  $\underline{a} > \underline{b}$ . From  $\underline{a} < \underline{b}$ , it follows that  $a < b$ , and from  $\underline{a} > \underline{b}$ , it follows that  $a > b$ , but as already mentioned, from  $\underline{a} \dot{=} \underline{b}$  any of the relations  $a < b$ ,  $a = b$ , or  $a > b$  could be true.

Note that intervals may also be compared with exact numbers, because an exact number may be viewed as defining an interval of length zero. Thus, for any interval  $\underline{a}$ , we always have one of the three possibilities:  $\underline{a} > 0$ ,  $\underline{a} < 0$ , or  $\underline{a} \dot{=} 0$ . We say an interval is "positive", "negative", or "overlaps 0", depending on which of the three listed relations holds.

An interval relationship is obtained at some specific computation precision. So if  $\underline{a} \dot{=} \underline{b}$  holds at a certain precision, then when precision is increased we may still find  $\underline{a} \dot{=} \underline{b}$ , but we could find  $\underline{a} > \underline{b}$  or we could find  $\underline{a} < \underline{b}$ . We write  $\underline{a} \neq \underline{b}$  to indicate that the two intervals  $\underline{a}$  and  $\underline{b}$  do not intersect. Then it is always true that  $\underline{a} \dot{=} \underline{b}$  or  $\underline{a} \neq \underline{b}$ , because two intervals either intersect or they do not intersect. It may be useful here to list the conclusions that can be reached when various interval relations hold:

Interval relation	Implied mathematical relation
$\underline{a} < \underline{b}$	$a < b$
$\underline{a} > \underline{b}$	$a > b$
$\underline{a} \neq \underline{b}$	$a \neq b$
$\underline{a} \dot{=} \underline{b}$	None certain

Conversely, if we know that a certain mathematical order relation holds between two numbers, this information limits the interval relations we could find:

Mathematical relation	Possible interval relation
$a < b$	$\underline{a} < \underline{b}$ or $\underline{a} \dot{=} \underline{b}$
$a > b$	$\underline{a} > \underline{b}$ or $\underline{a} \dot{=} \underline{b}$
$a = b$	$\underline{a} \dot{=} \underline{b}$

In this textbook, because range arithmetic is the computer arithmetic used, the phrase “if  $a < b$ , then” should be interpreted as meaning “if at the current precision  $\underline{a} < \underline{b}$ , then”. Similarly, the phrase “if  $a \neq b$ , then” should be interpreted as “if at the current precision  $\underline{a} \neq \underline{b}$ , then”.

## 2.7 Computing standard functions in range arithmetic

After the four arithmetic operations  $+$ ,  $-$ ,  $\times$  and  $\div$  have been coded for range arithmetic, one can compose simple programs to use this arithmetic. But often we also need standard functions like  $\ln x$ ,  $\sin x$ , or  $\cos x$ . For instance, we may need to find  $\cos 3.12222 \pm 1$ , where the cosine argument is in radians.

In range arithmetic, standard functions can be formed in much the same way they are formed with ordinary floating-point arithmetic, that is, by using power series expansions in combination with various useful identities that apply to the particular function. Thus to form  $\cos x$ , we can use the three identities  $\cos(-x) = \cos x$ ,  $\cos x = \cos(x - 2\pi n)$ , and  $\cos x = -\cos(x - \pi)$  to replace the argument  $x$  by a value  $x'$  between 0 and  $\pi/2$ . Then we halve  $x'$  as many times as necessary to confine its new value  $x''$  within a narrower interval, say  $[0, 0.1]$ . We record the number of halvings, intending to perform the scaling operation  $2y^2 - 1$  on  $y = \cos x''$  as many times as we halved, in accordance with the identity  $\cos(2x) = 2\cos^2(x) - 1$ .

To compute  $\cos x''$ , we use the Maclaurin series expansion

$$\cos u = 1 - \frac{u^2}{2!} + \frac{u^4}{4!} - \frac{u^6}{6!} + \cdots + (-1)^n \frac{u^{2n}}{2n!} + \cdots$$

How many terms of the series should we use to form  $\cos x''$ ? In the process of summing the series, we would be adding consecutive terms to a summation variable  $S$ . The terms rapidly decrease in magnitude, and when finally we come to a term  $T$  whose magnitude  $|T|$  is less than the  $S$  halfwidth, then clearly this is a good point to break off the summation.

The  $S$  halfwidth bounds the computation error in forming the partial sum for  $\cos x''$ , but does not include the series truncation error that we make by ending the summation. Because the series is an alternating series, with successive terms decreasing in magnitude, the truncation error is bounded by the magnitude of the first series term not summed. We need to increase the halfwidth of  $S$  by an amount sufficient to account for the truncation error  $|T|$ .

This evaluation example shows that it is useful to introduce another operation to range arithmetic, the operation

$$S \oplus T$$

The result of  $S \oplus T$  is the first operand  $S$ , but with an increased halfwidth, the increase being equal to the largest possible  $|T|$  value in the  $T$  interval.



The operation  $\oplus$  has another application besides its use in standard function computations. Before giving this application, first note that when a mathematical constant is computed in range arithmetic, we get an interval with a halfwidth that can be examined and tested. If we wish to print the constant to a certain number of decimal places, the size of this halfwidth determines whether we can get the specified number of correct places from the interval. If we can not, the higher precision needed can be determined, and the computation then is repeated at this precision.

Now suppose we want to solve some mathematical problem numerically. Here we may have one procedure for computing an approximation  $A$  to the answer, and another procedure for computing an upper bound  $E$  on the error of our approximation. Both computations yield range arithmetic numbers with halfwidths bounding their computational error. The operation  $A \oplus E$  then gives a correctly ranged approximation to the answer that can be tested to determine whether the required number of correct decimal places can be obtained from it. If not, we increase precision, recompute both  $A$  and  $E$ , and try again.

## 2.8 Rational arithmetic

A useful byproduct of range arithmetic is that rational arithmetic becomes easy to obtain. For rational arithmetic we use a pair of exact numbers to represent each rational, namely a numerator integer  $p$  and a denominator integer  $q$  to represent the rational in the form  $p/q$ . The denominator integer  $q$  cannot be zero, but there is no restriction on the numerator  $p$ . The rational operations of addition, subtraction, multiplication, and division are executed according to the following rules:

$$\begin{aligned} \frac{p_1}{q_1} + \frac{p_2}{q_2} &= \frac{p_1q_2 + q_1p_2}{q_1q_2} \\ \frac{p_1}{q_1} - \frac{p_2}{q_2} &= \frac{p_1q_2 - q_1p_2}{q_1q_2} \\ \frac{p_1}{q_1} \times \frac{p_2}{q_2} &= \frac{p_1p_2}{q_1q_2} \\ \frac{p_1}{q_1} \div \frac{p_2}{q_2} &= \begin{cases} \frac{p_1q_2}{q_1p_2} & \text{if } p_2 \neq 0 \\ \text{Division error} & \text{if } p_2 = 0 \end{cases} \end{aligned} \quad (2.8)$$

Before executing any of the four operations shown above, PRECISION is temporarily set to its highest value,  $N$ . The resulting  $p$  and  $q$  integers then are computed in range arithmetic, following the rules given above. Because no division operation is ever needed, the two integers obtained are always exact.

If the range of one of these integers is ever nonzero, then  $N$ , the implementation bound on the number of mantissa digits, is not large enough for the computation.

It is convenient to restrict denominator integers  $q$  to positive values only. It is then easier to decide whether we are dealing with a positive or negative rational, because we need examine only the sign of  $p$ , rather than the signs of both  $p$  and  $q$ . With this restriction, the equations for arithmetic operations given earlier must be reexamined. The equations for addition, subtraction, or multiplication need no changes, because each of these operations yields a result with a positive denominator when both operands have positive denominators. This is not the case for division, however, and the revised operation is

$$\frac{p_1}{q_1} \div \frac{p_2}{q_2} = \begin{cases} \frac{p_1 q_2}{q_1 p_2} & \text{if } p_2 > 0 \\ \frac{-p_1 q_2}{q_1 |p_2|} & \text{if } p_2 < 0 \\ \text{Division error} & \text{if } p_2 = 0 \end{cases}$$

It is convenient also to always eliminate common divisors from the pair of integers  $p$ ,  $q$ , and obtain each rational in what is called *reduced* form. This way, we can conclude that a rational  $p_1/q_1$  equals another rational  $p_2/q_2$ , after we check that  $p_1$  equals  $p_2$  and that  $q_1$  equals  $q_2$ . If common divisors are not eliminated, we reach the same conclusion only after we find that  $p_1 q_2$  equals  $p_2 q_1$ , and this requires two multiplications. Elimination of common divisors also keeps the integers  $p$  and  $q$  smaller, making arithmetic operations faster, and making it less likely that we exceed  $N$ , the bound on mantissa length. Accordingly, as each rational  $p/q$  is obtained by an initial construction or through an arithmetic operation, the greatest common divisor  $D$  of  $|p|$  and  $q$  is found, and if  $D$  is greater than one, then the pair  $p$  and  $q$  is replaced by the pair  $p'$  and  $q'$ , with  $p' = p/D$  and  $q' = q/D$ .

The method we use for finding  $D$  is the Euclidean algorithm, which finds the greatest common divisor of any two positive integers  $n_1$  and  $n_2$  by repeated divisions. The procedure is as follows. We divide the larger integer by the smaller to obtain an integer quotient and an integer remainder. Assume now that  $n_1 \geq n_2$  (the integers are switched if this inequality is not true). Let  $d_1$  be the quotient and  $n_3$  the remainder when  $n_1$  is divided by  $n_2$ , so that

$$n_1 = d_1 n_2 + n_3$$

with  $0 \leq n_3 < n_2$ . From the last equation, it is clear that any common divisor of  $n_1$  and  $n_2$  is also a divisor of  $n_3$ , so the greatest common divisor of  $n_1$  and  $n_2$  is a common divisor of  $n_2$  and  $n_3$ . On the other hand, any common divisor of  $n_2$  and  $n_3$  is also a divisor of  $n_1$ , so the greatest common divisor of  $n_2$  and  $n_3$  is a

common divisor of  $n_1$  and  $n_2$ . The two statements together imply that these two greatest common divisors are identical, so the problem of finding the greatest common divisor can be shifted from the pair  $n_1$  and  $n_2$  to the pair  $n_2$  and  $n_3$ , and the entire process can be repeated on the new pair. We see that  $n_3$ , the smaller number of the new pair, is less than  $n_2$ , the smaller number of the original pair. Thus if this procedure is repeated enough times, it must lead eventually to a pair of integers with one member being zero. When this happens the desired greatest common divisor equals the nonzero member of this final pair. Thus for the integer pair 144 and 78, we arrive at their greatest common divisor in the following four steps:

Pair	Division relation
144, 78	$144 = 1 \cdot 78 + 66$
78, 66	$78 = 1 \cdot 66 + 12$
66, 12	$66 = 5 \cdot 12 + 6$
12, 6	$12 = 2 \cdot 6 + 0$
6, 0	The greatest common divisor is 6

## Software Exercises A

Software exercises here and in later chapters are designed to clarify how to use the demo programs to best advantage. All the exercises in this section concern the demo program `calc`. Our goal with `calc` is to evaluate to 10 correct decimal places the two constants mentioned at the beginning of this chapter,  $e^{\sqrt{2}}$  and  $\tan 31^\circ$ . Along the way to doing that, some of the common properties of demo programs are explained. The last exercise gives some guidance in viewing the source code of a demo program, with the `calc` program used as an example.

1. Call up the program `calc` on your PC. Here you need to choose the **PNM** program from your list of Windows programs, and after the **PNM** form appears, click on the **Exe part** subsection of the **Command** menu. After the list of demo program names appears, click on the `calc` line. The form caption now is "PNM Command: calc". Next click on the **Go** subsection of the **Command** menu. The **PNM** form disappears. You next need to get to the Windows command subsection (see page 248), type the letter **g** (for "go") and hit the ENTER key to obtain the `calc` demo program.

The `calc` program requests that you specify the number of decimal places desired, but instead just type the letter **q** and hit the ENTER key to exit the program. Every demo program can be exited while it is waiting for an entry with the single letter **q**, for "quit".

2. While still in the Windows command subsystem, call up the `calc` program again by typing a `g` and hitting the ENTER key. When a demo program selection is made using the `PNM` form and selecting the `Go` subsection of the `Command` menu, that demo program is called whenever you enter the Windows command subsystem, type a `g` and hit the ENTER key. Now exit the program by typing a **control-c**, that is, hold the `CNTRL` key down and hit the `c` key. Every demo program can be exited with a **control-c** at any time, whether or not the program is waiting for a keyboard entry. This type of exit is needed when a demo program is making some computation, and you decide not to wait for the result.

3. Once more call up the `calc` program by typing a `g` and hitting the ENTER key. This time enter the integer 10 to choose 10 fixed-point decimal places for `calc`'s computed constants. Demo programs generally allow a user to choose  $n$  fixed-point decimal places by entering an unsigned integer  $n$ , and to choose  $n$  floating-point decimal places by entering the negative integer  $-n$ . If you enter an integer outside the range of allowed integer values, you get a message telling you what the bound is. The `calc` program bound is 150 decimal places. After you enter the 10, you see next a display of the symbols that can be used to define a real number constant. The five arithmetic operations handled by `calc` are `+`, `-`, `*` (multiplication), `/` (division), and `^` for exponentiation. The order in which arithmetic operation are executed can be controlled by adding parentheses. When a series of arithmetic operations appear without parentheses to delimit them, they are interpreted as described in the `SYMBOLS RECOGNIZED` display. For instance (you can test any example mentioned in an exercise by entering it),  $1+2*3$  is computed as if it were  $1+(2*3)$ , because the rank of `*` is higher than the rank of `+`. Similarly,  $2^2*2$  is computed as if it were  $(2^2)*2$ , because the rank of `^` is higher than the rank of `*`. When two or more operations of the same rank appear without parentheses, evaluation is left to right, except for the case of operations of "hi" rank, where evaluation is then right to left. Thus  $4*3/2$  is computed as if it was  $(4*3)/2$ , but  $-2^2$  is computed as if it was  $-(2^2)$ .

4. Decimal constants supplied to `calc` may use a sign prefix, and may have a single decimal point, but should not have any commas. Thus the entry `2,100` will yield a "syntax" error report. The helpful 'e' notation may be used here. Thus `2.1e3` is interpreted as 2100, and `2e-3` as 0.002. One may view the 'e' following a decimal constant as denoting  $*10^{\text{exponent}}$ , so `2.1e3` may be viewed as  $2.1*10^3$ . If desired, the letter 'e' can be space separated from the constant's decimal part or exponent part, so `2.1 e3` may also be entered as `2.1 e3` or `2.1 e 3`. Exit the `calc` program and call it up again, but this time enter `-10` in response to the request for decimal places. Now when you evaluate a constant such as  $2^{100}$ , you obtain the constant with 'e' notation, but the 'e' is now capitalized.

5. Exit `calc` and call it up again and specify 10 fixed-point decimal places for the remainder of these exercises. We can now evaluate the first target constant,  $e^{\sqrt{2}}$ . The function `exp()` is the exponential function, so the keystroke entry needed for  $e^{\sqrt{2}}$  is `exp(sqrt(2))`. Each function computed by `calc`, such as `exp()` or `sqrt()`, requires a set of parentheses to delimit the function's argument.

6. The function `ln()` and the function `log()` are identical, both being the "natural logarithm" function  $\ln$ . Thus `ln(exp(1))` is 1. To obtain other logarithms, use the identity

$$\log_b x = \frac{\ln x}{\ln b}$$

Thus to obtain  $\log_{10} 97$  use the entry `ln(97)/ln(10)`.

7. We are ready now to evaluate the other constant mentioned at the beginning of this chapter,  $\tan 31^\circ$ . The three trigonometric functions `sin()`, `cos()`, and `tan()` all use radian angle measure, so degrees must be specified in terms of radians; 180 degrees equals  $\pi$  radians, so one degree is  $\pi/180$  radians. Accordingly the keystroke entry to use for  $\tan 31^\circ$  is `tan(31*pi/180)`. One or more spaces may separate the tokens that make up an entry, so `tan(31 * pi / 180)` is acceptable, but `tan(31*pi/180)` yields an error message, because the letters of a function must be together.

8. The three functions `asin()`, `acos()`, and `atan()` are the three standard inverse trigonometric functions  $\sin^{-1}$ ,  $\cos^{-1}$ , and  $\tan^{-1}$ . The angle supplied by each of these functions is a radian angle, so if degrees are desired, the conversion factor  $180/\pi$  is needed. Thus to obtain  $\tan^{-1} 1$  in degrees, the entry `atan(1)*180/pi` is needed.

9. The source code of all demo programs is written in the C++ programming language. A demo program `abc` has its central controlling program in a file with the name `abc.cc`. To see this file for `calc`, return to the **PNM** form, choose the **Source** menu and click on the `cc files` subsection. After the list of files appears, click on the `calc.cc` line to see the `calc.cc` source code. One can often get a general idea of how a demo program does its task by viewing its source code, even if you have never used C++.

The first line of a demo source code file lists the files that are linked together to produce the demo execution program, the list always beginning with the demo program name. Thus the first line of `calc.cc` is

```
// link: calc real
```

This indicates that `calc` is linked with one other file of name `real`. The source file `real.cc` supplies the range arithmetic capabilities needed by `calc`. Any secondary file of the link list (such as `real`) in general has a source file (`real.cc`) and a descriptive header file (`real.h`). All such files can be viewed by using the Source menu.

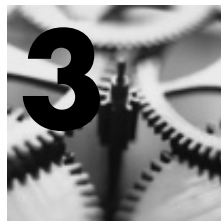
## Notes and References

- A. The earliest description of interval arithmetic appeared in the Ph.D. thesis [11] of Ramon Moore. The interval arithmetic concept made precise numerical analysis possible.
- B. Five general texts on interval arithmetic [5,8,12,13,14] are listed below.
- C. The articles by Demmel and Krückeberg [6] and Krückeberg and Leisen [10] describe pioneering experiments in combining variable precision with interval arithmetic. Krückenberg [9] has given a general description of the advantages of this type of computation.
- D. Range arithmetic can be programmed as a binary system [1] or as a decimal system [2,3,4].
- E. Another software package for variable precision interval arithmetic has been constructed by J. Ely [7].

- [1] Aberth, O., A precise numerical analysis program, *Comm. ACM* **17** (1974), 509–513.
- [2] Aberth, O., Precise scientific computation with a microprocessor, *IEEE Trans. Comput.* **C-33** (1984), 685–690.
- [3] Aberth, O., *The conversion of a high order programming language from floating-point arithmetic to range arithmetic*, in *Computer Aided Proofs in Analysis*, pp 1–6, IMA Volumes in Mathematics and its Applications, Vol. 28, edited by K. R. Meyer and D. S. Schmidt, Springer-Verlag, New York, 1990.
- [4] Aberth, O. and Schaefer, M. J., Precise computation using range arithmetic, *ACM Trans. Math. Software* **18** (1992), 481–491.
- [5] Alefeld, G. and Herzberger, J., *Introduction to Interval Computation*, Translated by Jon Rokne, Academic Press, New York, 1983.
- [6] Demmel, J. W. and Krückeberg, F., An interval algorithm for solving systems of linear equations to prespecified accuracy, *Computing* **34** (1985), 117–129.
- [7] Ely, J. S., The VPI software package for variable precision interval arithmetic, *Interval Comput.* **2** (1993), 135–153.
- [8] Hansen, E., *Global Optimization using Interval Analysis*, Marcel Dekker, Inc., New York, 1992.
- [9] Krückeberg, F., *Arbitrary accuracy with variable precision arithmetic*, in *Interval Mathematics 1985*, edited by Karl Nickel, Lecture Notes in Computer Science 212, Springer-Verlag, Berlin, 1986.
- [10] Krückeberg, F. and Leisen, R., *Solving initial value problems of ordinary differential equations to arbitrary accuracy with variable precision arithmetic*, in *Proceedings of the 11th IMACS World Congress on System Simulation and Scientific Computation*, Vol. 1, Oslo, Norwegen, 1985.

- [11] Moore, R. E., *Interval Arithmetic and Automatic Error Analysis in Digital Computing*, Ph.D. Dissertation, Stanford University, 1962.
- [12] Moore, R. E., *Interval Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1966.
- [13] Moore, R. E., *Methods and Applications of Interval Analysis*, *SIAM Studies in Applied Mathematics*, SIAM, Philadelphia, 1979.
- [14] Neumaier, A., *Interval Methods of Systems of Equations*, *Encyclopedia of Mathematics and its Applications*, Cambridge University Press, Cambridge, 1990.

# Classification of Numerical Computation Problems



To achieve precision in numerical computation, a numerical analyst must be aware of the potential difficulty lurking in certain simple computation tasks. We begin by examining an especially simple computation task, deciding whether or not a real number is zero.

## 3.1 A knotty problem

Imagine we are computing a certain quantity  $a$  and must determine whether  $a$  equals zero. If  $a$  equals zero we take branch `zero`, otherwise we take branch `nonzero`. The number  $a$  may be the value of a solution to a complicated differential equation problem at some particular point, or a number obtained through some other intricate computation.

There is no difficulty in carrying out this task in ordinary floating-point arithmetic, because we take branch `zero` if the  $a$  floating-point approximation comes out as zero, otherwise we take branch `nonzero`. However, the  $a$  floating-point approximation has an unknown error, so we may end up taking the wrong branch.

Now consider doing the test in range arithmetic, and here we imagine that we obtain various numbers of correct fixed-point decimal places for  $a$ , displayed in the usual manner. We list some sample cycles below. Keep in mind that if  $a$  is actually zero, we cannot expect to obtain an exact range arithmetic zero when we compute  $a$ , because this occurs only for the simplest computations.

The problem with the last case is that we never are certain when to take branch `zero`. Indeed, what should we do when we obtain a sequence of better and better approximations to zero, defining smaller and smaller intervals? We cannot continue to form better and better approximations indefinitely, because  $a$  may equal zero, and then we would be in an endless loop. And if we adjust



our program so that we take branch `zero` when the  $a$  interval containing the zero point becomes “sufficiently” small, we really are taking branch `zero` not just for  $a = 0$ , but for  $a$  equal to any of the real numbers lying in the sufficiently small interval. Anyone examining our program could demonstrate to us our error by arranging things so that  $a$  was not equal to zero, but merely close enough so that it passed our criterion for taking branch `zero`.

$a$	Outcome
0.144~	take branch <code>nonzero</code>
0.00000~	recompute to higher precision
0.0000000003~	take branch <code>nonzero</code>
0.00000~	recompute to higher precision
0.000000000~	recompute to higher precision
0.000000000000000~	recompute to higher precision
⋮	⋮

Thus deciding for a real number whether it is zero has a potential difficulty. The difficulty is not a peculiarity of range arithmetic, but merely becomes more evident with this mode of computation. The difficulty for the problem of deciding whether an arbitrary real number is zero cannot be overcome by using a more complicated computation arithmetic or a more complicated analysis technique. In this chapter we draw attention to problems that have an intrinsic difficulty, the “nonsolvable” problems. The mathematical problems free of such difficulties are “solvable” problems.

In the next section we give a brief introduction to the mathematical basis for this terminology. Our first classified problem is

**Nonsolvable Problem 3.1** For any real number  $a$ , decide whether  $a$  equals 0.

Often a nonsolvable problem can be changed in some convenient manner to become solvable. For instance, in the example just used, perhaps taking branch `zero` for a nonzero  $a$  leads to no difficulty as long as  $|a|$  is “very small”. In this case, we should switch from trying to decide whether  $a$  is exactly zero, to deciding whether it is close to zero, as described in the following

**Solvable Problem 3.2** For any real number  $a$  and positive integer  $k$ , decide that  $a$  is unequal to 0, or decide that  $|a| < 10^{-k}$ .

A decision procedure is easy to carry out in range arithmetic. Once more we use underlining, as in  $\underline{a}$ , to indicate the range arithmetic interval obtained for

a mathematical constant. We evaluate  $a$  at some chosen precision and compare  $\underline{a}$  with 0. If we find  $\underline{a} \neq 0$ , then  $a \neq 0$ . If we find  $\underline{a} \doteq 0$  and determine that the  $\underline{a}$  width  $< 10^{-k}$ , then  $|a| < 10^{-k}$ . This is because  $\underline{a}$  contains the zero point, so every real number in  $\underline{a}$  is  $< 10^{-k}$ . If  $\underline{a} \doteq 0$  but the  $\underline{a}$  width is  $\geq 10^{-k}$ , we increase precision appropriately to obtain a decision on the next trial.

Notice that with this first solvable problem, the two outcomes are not mutually exclusive. It is possible that at one precision we decide that  $|a| < 10^{-k}$ , and at a higher precision we decide that  $a \neq 0$ .

## 3.2 The impossibility of untying the knot

In the 1930s, revolutionary methods were introduced into mathematics that have changed our understanding of the subject. Turing, in a brilliant paper [5] wherein he defined the machines now known as Turing machines, proved that a long-standing logical problem, with the German name “Entscheidungsproblem”, was not solvable by any mechanical method. Turing’s stunning conclusion was obtained by assuming that the result in question could be achieved, and then showing that this presumption leads to a contradiction.

The difficulty of Nonsolvable Problem 3.1 occurs because in numerical computation any real number we deal with is a “computable number”, that is, a real number for which a constructive algorithm is known that can deliver rational approximations to the number that are as accurate as we please. Although we can obtain arbitrarily accurate approximations to a computable number, nevertheless these approximations do not allow us to decide in all cases whether the number is zero. And there can never be a constructive way of determining whether computable numbers are zero, perhaps by some method of examining the internal structure of the approximation algorithm, because if such a method existed, then with a Turing-style argument, one can show how to define a computable number that thwarts the decision method.

In a sense, when a problem is shown to be nonsolvable, this is a liberating result. We no longer need to wonder how to tackle the resistant difficult cases of the problem, because we see that any method we use can be only partially successful. It is more reasonable to consider how to avoid the problem altogether.

## 3.3 Repercussions from nonsolvable problem 3.1

Given a nonsolvable problem, suppose we have a second problem for which a constructive solution implies a method of solving the first problem. This second problem then must be nonsolvable too. For instance, as a consequence of Nonsolvable Problem 3.1, we have also

**Nonsolvable Problem 3.3** For any two real numbers  $a$  and  $b$ , decide whether  $a = b$ .

Because we can take  $b$  as zero, a method for solving this problem is also a method for solving Problem 3.1. The nonsolvable problem just obtained then implies the following

**Nonsolvable Problem 3.4** For any two real numbers  $a$  and  $b$ , decide whether  $a > b$ .

If it were possible to always determine whether  $a > b$ , then from our **yes** or **no** results from testing  $a > b$  and  $b > a$ , we would know whether  $a = b$ , contradicting Nonsolvable Problem 3.3.

For real numbers  $a$  and  $b$ , exactly one of the three possibilities can hold:  $a < b$ ,  $a = b$ , or  $a > b$ . We know now that deciding which of the three holds is a nonsolvable problem, and because this problem arises often, we list it below, with a revised, solvable version.

**Nonsolvable Problem 3.5** For any two real numbers  $a$  and  $b$ , select the single correct relation from these three:  $a < b$ ,  $a = b$ , or  $a > b$ .

**Solvable Problem 3.6** For any two real numbers  $a$  and  $b$ , and any positive integer  $k$ , select a correct relation from these three:  $a < b$ ,  $|a - b| < 10^{-k}$ , or  $a > b$ .

Problem 3.6 is solvable because we can compute  $\underline{a}$  and  $\underline{b}$  to higher and higher precision until both intervals in width are  $< \frac{1}{2} \cdot 10^{-k}$ . If  $\underline{a} \doteq \underline{b}$ , then  $|a - b| < 10^{-k}$  because the sum of the widths of  $\underline{a}$  and  $\underline{b}$  is  $< 10^{-k}$ . And if the two intervals do not overlap, we easily determine whether  $\underline{a} < \underline{b}$  or  $\underline{a} > \underline{b}$ . Here we do not try to decide whether  $|a - b| < 10^{-k}$ , for that would be nonsolvable too, forcing us to decide, in some cases, whether  $|a - b|$  was exactly  $10^{-k}$ . The three choices listed in Problem 3.6 are not mutually exclusive. It is quite possible that at a certain precision we decide that  $a$  is within  $10^{-k}$  of  $b$ , but at a higher precision we decide that one of the other possibilities occurs.

Another way the difficulty of Problem 3.5 can be overcome, for inequality tests at least, is to have two distinct numbers to test against, as in the following

**Solvable Problem 3.7** For any real number  $a$  and any two unequal real numbers  $b_1$  and  $b_2$ , choose either  $b_1$  or  $b_2$  as a number unequal to  $a$ .

Because we know that  $b_1$  and  $b_2$  are distinct, by computation to sufficiently high precision the intervals  $\underline{b}_1$  and  $\underline{b}_2$  will not intersect, and a certain rational width  $\delta$  will separate them. If we now compute  $a$  to a precision high enough

that the  $\underline{a}$  width  $< \delta$ , then  $\underline{a}$  can intersect at most one of the  $\underline{b}_1$  and  $\underline{b}_2$  intervals, and so at least one of the  $b_1$  and  $b_2$  numbers is found to be unequal to  $a$ .

A third way the difficulty of Nonsolvable Problem 3.5 can be overcome is by putting some restriction on the type of numbers that are compared. Certainly if the two numbers are rational, that is, they are expressed in the form  $p/q$  with the integers  $p$  and  $q$  known, there is no difficulty deciding any order relation. However, it does not suffice merely to know that the two numbers  $a$  and  $b$  are rational, if we can only obtain range arithmetic approximations to them.

### 3.4 Some solvable and nonsolvable decimal place problems

Previously, we identified the difficulty in deciding whether one number was equal to, less than, or greater than a second number. When ranged approximations to the two numbers yield overlapping intervals, and this keeps happening as we increase the precision, we are in a quandary as to which relation holds. This difficulty shows up sometimes when we attempt to obtain a correct  $k$ -place fixed-point approximation to a number. For instance, suppose we are trying to get a correct five-place approximation to a certain number  $a$  and we obtain the following sequence of ranged approximations to  $a$ :

$$\begin{aligned} &.1111150 \pm 2 \\ &.1111150000 \pm 1 \\ &.11111500000000 \pm 3 \\ &\dots \end{aligned}$$

We are unable to decide whether our correct five-place result should be  $.11111\sim$  or  $.11112\sim$ , because the various  $a$  approximations all contain within their intervals the point  $.111115$  which is midway between  $.11111$  and  $.11112$ . If  $a < .111115$ , the only correct five-place fixed-point approximation is  $.11111\sim$ ; if  $a > .111115$ , the only correct one is  $.11112\sim$ ; and if  $a = .111115$ , either  $.11111\sim$  or  $.11112\sim$  can be used. Thus choosing which five-place approximation is correct hinges on deciding the correct order relation between  $a$  and the rational number  $.111115$ . None of the  $a$  approximations shown above allows us to make this determination. Thus we encounter Nonsolvable Problem 3.5, and because it is clear the difficulty is independent of the number of decimal places desired, we have

**Nonsolvable Problem 3.8** For any real number  $a$  and positive integer  $k$ , obtain a correct  $k$  decimal place fixed-point approximation to  $a$ .

A way out of the impasse is easy to find. Whenever it becomes difficult to decide between two neighboring  $k$ -place approximations, give one extra, correct

decimal-place. This should not be difficult, because if the decimally expressed rational  $d_0.d_1d_2 \dots d_k5$  is inside a succession of  $a$  intervals, then  $d_0.d_1d_2 \dots d_k5$  is a correct  $k + 1$ -place approximation as soon as the error of the  $a$  approximation becomes small enough. For instance, with our example above, a six-place approximation, namely  $.111115\sim$ , can be obtained from the very first ranged value, where the difficulty first appeared. The error of this initial  $a$  approximation, 2 units in the seventh decimal position, is below the allowance for six correct places, namely  $.0000005$ . In general, we have

**Solvable Problem 3.9** For any real number  $a$  and positive integer  $k$ , obtain a correct  $k$  decimal place or a correct  $k + 1$  decimal place fixed-point approximation to  $a$ .

With our demo programs, when results are to be displayed to a certain number of correct decimal places in fixed-point format, an extra decimal place may occasionally be noticed for some answers. The extra place is always a 5, supplied automatically if the difficulty we have been describing shows up.

An interesting question is what error bound must we achieve to approximate a quantity  $Q$  to either  $k$  or  $k + 1$  correct decimal places in the manner allowed by Solvable Problem 3.9? We consider the case where  $k$  equals three. Suppose that we have obtained an approximation  $m \pm \epsilon$  to  $Q$ . Here  $\epsilon$  is a bound on all errors made in computing  $m$ . We can form a three-place decimal approximation from the rational number  $m$ , but then we make an additional rounding error  $\epsilon_R$ , which can be as large as  $0.5 \cdot 10^{-3}$ . Thus the three possible  $m$  values,  $.111422$ ,  $.1115$ , or  $.11065$ , all yield the same three-place approximation  $.111\sim$ , with rounding errors, respectively, of  $.422 \cdot 10^{-3}$ ,  $.5 \cdot 10^{-3}$ , and  $.35 \cdot 10^{-3}$ . We want our three-place value to be correct, so  $\epsilon + \epsilon_R \leq .5 \cdot 10^{-3}$ , or  $\epsilon \leq .5 \cdot 10^{-3} - \epsilon_R$ . When the rounding error  $\epsilon_R$  gets close to  $.5 \cdot 10^{-3}$ , forcing  $\epsilon$  to be small, we try instead to form a four-place approximation from  $m$ . Now we have  $\epsilon \leq .5 \cdot 10^{-4} - \epsilon_R$ . If  $m$  is between  $.111475$  and  $.111525$ , our four-place representation is  $.1115\sim$ , with a rounding error no larger than  $.25 \cdot 10^{-4}$ , so that our inequality is  $\epsilon \leq .25 \cdot 10^{-4}$ . And if  $m$  is between  $.1110$  and  $.111475$ , or between  $.111525$  and  $.112$ , the three-place representations  $.111\sim$  or  $.112\sim$  have rounding errors no larger than  $.475 \cdot 10^{-3}$ , again giving the inequality  $\epsilon \leq .25 \cdot 10^{-4}$ . Thus for  $m$  varying between  $.111$  and  $.112$ , the largest rounding errors, occurring when  $m = .111475$  or  $m = .111525$ , force  $\epsilon$  to be no larger than  $.25 \cdot 10^{-4}$ . In general, to be certain that we can form a correct  $k$  or  $k + 1$  decimal place fixed-point approximation to a quantity  $Q$ , we must keep the  $Q$  error bound to at most

$$\epsilon_k = .25 \cdot 10^{-(k+1)} \quad (3.1)$$

We consider next the problem of forming correct  $k$ -place scientific floating-point approximations. It is clear that the difficulty of forming exactly  $k$  decimal places can occur here too.

**Nonsolvable Problem 3.10** For any real number  $a$  and positive integer  $k$ , obtain a correct  $k$  decimal place scientific floating-point approximation to  $a$ .

Here there is even an additional difficulty that is sometimes encountered. Suppose we are attempting to find a correct four-place floating-point approximation to a number  $a$ , and the series of range arithmetic approximations we obtain, at ever increasing precision, is as follows:

$$\begin{aligned} &.1 \pm 2 \cdot 10^{-5} \\ &.0 \pm 3 \cdot 10^{-12} \\ &.1 \pm 4 \cdot 10^{-17} \\ &\dots \end{aligned}$$

The problem here is that we can not find even one correct decimal place if the zero point is in the  $a$  interval. If  $a$  is actually zero, we should display 0 instead of floating-point digits. Thus with scientific floating-point, when attempting to obtain a certain number of correct places, we may be bumping into the nonsolvable problem of determining whether the number equals zero. One way of revising the problem to make it solvable is the following:

**Solvable Problem 3.11** For any real number  $a$  and positive integer  $k$ , obtain a correct  $k$  decimal place or a correct  $k + 1$  decimal place, scientific floating-point approximation to  $a$ , or else indicate that  $|c| \leq 10^{-k}$ .

Now we allow an escape from the decimal place problem when we are uncertain whether  $a$  is zero. The magnitude bound  $10^{-k}$  suggested here is arbitrary, and can be replaced by some other convenient bound, for instance,  $10^{-2k}$  or  $10^{-k^2}$ . By making the escape bound depend on  $k$ , we get smaller magnitude indications as more decimal places are requested.

The escape magnitude bound used by our demo programs is one-half the amount suggested in Problem 3.11. Suppose results are to be printed to five correct places in floating-point format. The escape is indicated by replacing the normal display, such as  $3.22222 \sim E12$ , with the display  $0. \sim E - n$ , where the integer  $n$  is always at least equal to  $k$ , the number of correct decimal places requested. For instance, in five-place floating-point format,  $0. \sim E - 5$  or  $0. \sim E - 9$  might be displayed. A floating-point display normally starts with a nonzero integer, so displaying a leading zero makes the escape unmistakable. Because the error of  $0. \sim$  is at most one-half of a unit in the 0 position, this makes  $\frac{1}{2} \cdot 10^{-n}$  the magnitude bound of  $0. \sim E - n$ .

**In the remainder of this text, whenever we use the phrase “to  $k$  decimal places” in the statement of a solvable problem, we mean “to  $k$  or  $k + 1$  correct decimal places in either fixed-point format or scientific floating-point format, with a size escape allowed for floating-point”.**

The difficulties in obtaining precisely  $k$  decimal places for a number  $a$  do not occur if  $a$  is a rational number  $p/q$  and the integers  $p$  and  $q$  are known.

Here, when we form a  $k$  decimal-place approximation to  $a$ , our only error is the rounding error, which always can be kept to no more than  $\frac{1}{2} \cdot 10^{-k}$ .

**Solvable Problem 3.12** For any rational number  $p/q$  and positive integer  $k$ , obtain a correct  $k$ -place fixed-point approximation, and, if  $p \neq 0$ , a correct  $k$ -place scientific floating-point approximation.

### 3.5 The solvable problems handled by `calc`

In the science of numerical analysis, it is advantageous to treat only solvable problems because such problems have no intrinsic difficulties. Different treatments of a particular solvable problem may be compared with respect to how completely they solve the problem.

Consider the demo program `calc` of Software Exercises A. This program may be considered as solving either Problem 3.9, determining  $k$  correct fixed-point decimal places for an arbitrary real number, or solving Problem 3.11, determining  $k$  correct floating-point decimal places for an arbitrary real number. Both problems are solvable because they allow escapes from computation difficulties.

How far does `calc` go towards the complete solution of the two problems cited? First we note that the two problems set no bounds on  $k$ , the number of correct decimal places, whereas `calc` does. This is not a major flaw because any solution program must require some bound of this type, because of limitations of computer memory and computer speed. A more serious flaw is that the real numbers `calc` treats are merely those constants that the program allows a user to enter. To handle either problem in full generality, a solution program must allow a user to define any real number, by specifying, in an appropriate language, the algorithm by which the real number is to be calculated. So `calc` is very far indeed from a complete solution, even ignoring its decimal place restriction. Better programs than `calc` would allow a greater variety of standard functions perhaps, but the point being made here is that any solution program for a solvable problem never can solve all cases of the problem, so it is always possible to compare various solution efforts with each other and rate them in various ways.

### 3.6 Another nonsolvable problem

Nonsolvable Problem 3.1 is a basic nonsolvable problem in that many other computation problems can be shown to be nonsolvable as a consequence. One other basic nonsolvable problem, which arises in later chapters, is

**Nonsolvable Problem 3.13** For any real number  $a$ , either decide that  $a \geq 0$ , or decide that  $a \leq 0$ , with either conclusion being allowed when  $a = 0$ .

Here imagine that for various real numbers  $a$ , on the basis of our test of  $a$  we take either the program branch `non-negative` or the branch `non-positive`. Once more consider various cases:

<u>a</u>	<b>Outcome</b>
0.144~	take branch <code>non-negative</code>
0.00000~	recompute to higher precision
-0.0000000003~	take branch <code>non-positive</code>
0.00000~	recompute to higher precision
0.000000000~	recompute to higher precision
0.0000000000000~	recompute to higher precision
⋮	⋮

Again, the last case is the troublesome case. Whatever we choose to do for a number  $a$  of this variety, we could be wrong. There is no constructive way of making a correct decision that works for all computable real numbers  $a$ .

### 3.7 The trouble with discontinuous functions

Suppose we have a certain real function  $f(x)$  defined over an interval  $I$ . What should we require of  $f(x)$  so it is “realizable”? Clearly, from an  $x$  argument lying in the domain  $I$ , we should be able to obtain an  $f(x)$  value. That is, if we can obtain arbitrarily accurate ranged approximations to  $x$ , then we should be able to obtain arbitrarily accurate ranged approximations to  $f(x)$ . Generally we need a computer routine whereby a ranged  $x$  approximation is used to compute a ranged  $f(x)$  approximation. And further, the accuracy of the  $f(x)$  approximation must improve in some fashion as the accuracy of the  $x$  approximation improves.

Now consider a function with a discontinuity at some argument. As a simple example we use the well-known sign function, usually designated by the abbreviation **sgn**. Its definition is

$$\text{sgn } x = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

This function has a discontinuity at  $x = 0$ . Ideally a routine realizing this function would return one of three exact values,  $+1$ ,  $0$ , or  $-1$ , depending on whether



it finds  $x > 0$ ,  $x = 0$ , or  $x < 0$ , respectively. But deciding which of the three  $x$  relations holds is Nonsolvable Problem 3.5. We can decide only whether  $\underline{x} > 0$ ,  $\underline{x} \doteq 0$ , or  $\underline{x} < 0$  at some finite precision. If  $\underline{x} > 0$  or  $\underline{x} < 0$ , the value of  $\text{sgn } x$  is determined, but if  $\underline{x} \doteq 0$ , and the  $\underline{x}$  interval contains positive and negative rationals, the interval value for  $\text{sgn } x$  so far determined is  $0 \pm 1$ . Of course we can compute the  $x$  argument at increasingly higher precisions, but if  $x$  is zero, generally every  $\underline{x}$  interval contains positive and negative rationals. Only in the special case where we happened to obtain an exact zero for  $\underline{x}$ , would we be able to supply an exact zero as the value for  $\text{sgn } x$ . So for many cases where the argument  $x$  is zero, a better value for  $\text{sgn } x$  than  $0 \pm 1$  never is determined. The function  $\text{sgn}$  is not realizable. We see that the  $\text{sgn } x$  function we can implement is

$$\text{sgn } x = \begin{cases} +1 & \text{if } x > 0 \\ \text{sometimes } 0, \text{ but often undefined} & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Let us call a mathematical idea a “nonrealizable concept” if it is not possible to constructively implement the idea. Thus “the  $\text{sgn}$  function defined for all numbers  $x$ ” is a nonrealizable concept. It is clear that if  $\text{sgn } x$  were realizable we would have a finite means for deciding whether any number was zero or not, contradicting Nonsolvable Problem 3.1. Just as with a nonsolvable problem, where we try to determine a solvable version of it, with a nonrealizable concept it is illuminating to try to revise the idea to make it realizable. There is no difficulty realizing a  $\text{sgn}$  function defined for all numbers *except* zero, the point of discontinuity. For an arbitrary argument  $x$ , our program forms  $\underline{x}$  approximations to increasingly higher precision endlessly, until at last it obtains  $\underline{x} \neq 0$ , and then the program can return a  $\text{sgn}$  value of  $+1$  or  $-1$  depending on whether  $\underline{x} > 0$  or  $\underline{x} < 0$ . Note that if a zero argument is mistakenly supplied, our program enters an endless loop. If we wish, we could make the program check whether the  $x$  approximation is an exact zero and return the proper  $\text{sgn}$  value of  $0$  in that case. Still, this adjustment would not eliminate all the endless loops for possible zero arguments, and the  $\text{sgn}$  function still would not be defined at  $x = 0$ .

The same difficulty is obtained with any function  $f(x)$  with a discontinuity at some point  $x_0$  within the function’s domain. The discontinuity requires us to treat arguments  $x$  in such a way that we do one thing if  $x = x_0$  and another thing if  $x \neq x_0$ . We can decide only whether  $\underline{x} \doteq x_0$  or  $\underline{x} \neq x_0$ , and this is insufficient.

In general, a mathematical function  $f(x)$  with discontinuities is not realizable. What is realizable is a function equal to  $f(x)$  at all points where  $f(x)$  is continuous, and *undefined* at the points where  $f(x)$  is discontinuous.

**Throughout this text, any function mentioned in a theorem, in a solvable problem, or in a nonsolvable problem, is to be presumed continuous in its domain.**

## Notes and References

- A. Turing's fundamental paper [5] is discussed in Davis's book [3].
- B. A complete proof that Problem 3.1 is nonsolvable is given in the book [1]; two other books that analyze similar problems are [2] and [4].
- [1] Aberth, O., *Computable Analysis*, McGraw-Hill, New York, 1980.
- [2] Aberth, O., *Computable Calculus*, Academic Press, San Diego, 2001.
- [3] Davis, M., *Computability and Unsolvability*, McGraw-Hill, New York, 1958.
- [4] Kushner, B. A., *Lectures on Constructive Mathematical Analysis*, Translations of Mathematical Monographs, Vol. 60, American Mathematical Society, 1980.
- [5] Turing, A. M., On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc., Ser. 2* **42** (1937), 230–265.

This page intentionally left blank

# Real-Valued Functions



This chapter describes certain commonly encountered functions. The demo program `fun` evaluates these functions, and the demo program `eval` shows how these functions are encoded.

## 4.1 Elementary functions

In a calculus course the great majority of functions used in the examples and problems can be expressed in terms of standard functions, such as  $e^x$ , also denoted  $\exp(x)$ , its inverse  $\ln x$ , or various trigonometric functions such as  $\sin x$ . Such functions are what we call elementary functions, for which it is easy to calculate derivatives and to obtain Taylor series expansions.

**Definition 4.1** A function of a finite number of real variables  $x_1, x_2, \dots, x_n$  is an *elementary function* if it can be expressed in terms of its variables and real constants  $c_1, c_2, \dots, c_m$  by a finite number of the binary operations of addition, subtraction, multiplication, division, or exponentiation, and the unary operations of function evaluation with  $\sin, \cos, \tan, \sin^{-1}, \cos^{-1}, \tan^{-1}, \exp$ , or  $\ln$ .

A prefix minus sign, as in  $-x$ , is viewed as a special case of a subtraction operation with an implied zero operand, that is,  $-x = 0 - x$ . The exponentiation operation with operands  $A$  and  $B$  produces  $A^B$ .

In later chapters sometimes we define an elementary function with one of its constants being used as a parameter. In those cases the parameter is designated by a subscript, as in  $f_c(x)$ , where the letter  $c$  here denotes the constant parameter.

The hyperbolic functions  $\cosh(x)$ ,  $\sinh(x)$ , or  $\tanh(x)$  are all elementary functions, because they can be expressed in terms of  $\exp(x)$ . For instance,  $\cosh(x) = [\exp(x) + \exp(-x)]/2$ . Similarly, the function  $\text{abs}(x) = |x|$  is an

elementary function, because it can be expressed in terms of exponentiation as  $(x^2)^{1/2} = \sqrt{x^2}$ . The two argument function  $\max(x, y)$ , which equals the maximum of the two real numbers  $x$  and  $y$ , may be viewed as an elementary function, because it can be expressed as  $\frac{a+b}{2} + \frac{|a-b|}{2}$ , which is  $\frac{a+b}{2} + \frac{\text{abs}(a-b)}{2}$ . Similarly,  $\min(x, y)$ , which equals the minimum of the two real numbers  $x$  and  $y$ , is an elementary function, because it can be expressed as  $\frac{a+b}{2} - \frac{|a-b|}{2}$ .

To simplify the task of specifying an elementary function, the demo programs allow a user the option of choosing `abs()`, `max(, )`, `min(, )`, or hyperbolic functions when defining an elementary function by keyboard entry. For instance, the keyboard line `tanh(x)^2` designates the elementary function  $\tanh^2 x$ . (Not  $\tanh x^2$ , because the argument part `(x)` of `tanh` is inseparable from `tanh`.)

Consider a typical elementary function

$$f(x) = \frac{e^{-x} \cos 3.2y + 0.56x^4}{1 + \tan^2 3.15z} \quad (4.1)$$

that is specified with the keyboard line

$$(\exp(-x) * \cos(3.2*y) + 0.56*x^4) / (1 + \tan(3.15*z)^2) \quad (4.2)$$

To evaluate the function on a computer, we need a list of operations to be done, called the function's *evaluation list*. Normally, the compiler for a programming language does the job of converting a string expression into an evaluation list; but if we want our range arithmetic programs to be able to compute elementary functions entered from the computer keyboard, then code to generate an evaluation list is needed. The source file `entry.cc` shows the code that does this task for demo programs, where function constants are usually just simple decimal quantities. For a general elementary function, it is possible that certain constants require long computer subroutines for their specification.

In the particular case of the function (4.1) just displayed, the evaluation list is given in Table 4.1. The list presumes an array of registers is available to hold ranged numbers. The array is initially empty, and then filled as needed, one register at a time, with a counter indicating the last register "active" or in use. Initially the counter is set to 0, indicating no active registers. Every evaluation operation is one of three types, a binary arithmetic or exponentiation operation, a unary sign change or function evaluation operation, or an operation which fills the next available register. If the operation is binary, the last two active registers supply the two operands, the operation result is placed in the lower numbered of these two registers, and the counter is decreased by 1. If the operation is unary, the last active register supplies the operand, the result is returned to this register, and the counter is left unchanged. There are two evaluation operations that increment the counter and add an element to the newly active register. The "variable to array end" enters a designated function variable, and the "constant to array end" enters a constant.

**TABLE 4.1** Evaluation List for the Function  $f(x)$  of Eq. (4.1) or (4.2)

Term No.	Operation	Active array
1	Variable $x$ to array end	$x$
2	Unary $-$	$-x$
3	$\exp()$	$e^{-x}$
4	Constant 3.2 to array end	$e^{-x}$ , 3.2
5	Variable $y$ to array end	$e^{-x}$ , 3.2, $y$
6	$\times$	$e^{-x}$ , 3.2 $y$
7	$\cos()$	$e^{-x}$ , $\cos(3.2y)$
8	$\times$	$e^{-x} \cos(3.2y)$
9	Constant 0.56 to array end	$e^{-x} \cos(3.2y)$ , 0.56
10	Variable $x$ to array end	$e^{-x} \cos(3.2y)$ , 0.56, $x$
11	Constant 4 to array end	$e^{-x} \cos(3.2y)$ , 0.56, $x$ , 4
12	exponentiation	$e^{-x} \cos(3.2y)$ , 0.56, $x^4$
13	$\times$	$e^{-x} \cos(3.2y)$ , 0.56 $x^4$
14	$+$	$e^{-x} \cos(3.2y) + 0.56x^4$
15	Constant 1 to array end	Numerator, 1
16	Constant 3.15 to array end	Numerator, 1, 3.15
17	Variable $z$ to array end	Numerator, 1, 3.15, $z$
18	$\times$	Numerator, 1, 3.15 $z$
19	$\tan()$	Numerator, 1, $\tan(3.15z)$
20	Constant 2 to array end	Numerator, 1, $\tan(3.15z)$ , 2
21	exponentiation	Numerator, 1, $\tan^2(3.15z)$
22	$+$	Numerator, Denominator
23	$\div$	$f(x)$

## Software Exercises B

The exercises with the demo program `fun` show how to use log files, and the exercises with the demo program `eval` show how demo programs encode simple elementary functions.

1. Call up the demo program `fun`. After obtaining the **PNM** form, click on the **Command** menu section **Exe part**, and choose `fun` from the list of program names displayed. Note that after this choice is made, the **Log part** section of the **Command** menu is inactive, indicating that no `fun.log` file is present. The `fun` program does create a log file after you completely specify a function evaluation problem, but like all demo programs creating log files, a log file is not present on the first call. Next click on the **Command** section **Go**, get to the Windows command subsystem and energize the `fun` program in the usual way by typing the letter **g** and hitting the ENTER key. Specify the sine function

by typing `sin(x)`, take the initial  $x$  value to be 0, and make the  $x$  increment 10 degrees (type `10*pi/180`). Choose 9 increments so that the final sine evaluation occurs at  $90^\circ$ . Finally choose 10 fixed-point decimal places for both the  $x$  and the  $f(x)$  displays.

2. The  $\sin x$  display obtained in the preceding exercise is unsatisfactory, because the  $x$  value is a radian value, and a degree value is preferable. Now that we have a `fun.log` file, it is easy to change details in the display. Obtain the **PNM** form once more. Note that this time the **Log part** section of the **Command** menu is active, indicating the presence of a log file. Click on the **Log part** section of the **Command** menu, and choose the single log file displayed. Note the change in the form caption. Next, click on the **Open** section of the **Log** menu, and again choose the single log file displayed. The log file contents will fill the **PNM** form. Changes to the log file now can be made. Change the  $f(x)$  function from `sin(x)` to `sin(x*pi/180)`, which switches  $x$  from radians to degrees. Next change the  $x$  increment from the radian value `10*pi/180` to the degree value 10. Also, 10 decimal places for degrees seems excessive, so change the  $x$  decimal places from 10 to 2. Next click on the **Save** section of the **Log** menu, and finally click on the **Go** section of the **Command** menu to exit the **PNM** form. After you energize the `fun` program in the usual way, a better  $\sin x$  display is obtained.

3. The `fun` program not only creates a log file, but also makes its display available for later inspection by saving the display in the print file `fun.prt`. To view the  $\sin x$  display, obtain the **PNM** form, click on the **Open** section of the **Prt** menu, and the contents of the `fun.prt` file appears in the **PNM** form. If you have a printer available on your PC, you can print the  $\sin x$  display by clicking on the **Print** section of the **Prt** menu. However, note that what is printed is whatever is showing in the **PNM** form, so before printing a print file, the **PNM** form should be examined to be sure that all parts of the print file are on display. If not, the **PNM** form should be enlarged to accomplish this.

4. The program `eval`, which does not create a log file or a print file, can be used to see an elementary function's evaluation list. Call up this demo program in the usual way, choose `reals`, and then type `1.2 * 3.4` as an evaluation example. The evaluation list shows that two constants get entered into the evaluation array. Constants needed for the evaluation process are obtained as exact ranged numbers by referring the appropriate generating subroutine to the position in the typed string of symbols where the constant begins. The constant ends in the typed string whenever either the string ends or some symbol not used for constant specification is encountered.

5. Try other functions of your choice to see the corresponding evaluation list. The evaluation list given in Table 4.1 will be displayed if you type line (4.2).

# Computing Derivatives



The demo program `deriv` computes ordinary derivatives and partial derivatives of elementary functions. This program finds derivatives by forming power series. In this chapter, methods for generating power series are presented, and afterwards the program `deriv` is described.

## 5.1 Power series of elementary functions

Consider an elementary function  $f(x)$  with the power series (or Taylor series) shown in the next line:

$$f(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \cdots + a_k(x - x_0)^k + \cdots \quad (5.1)$$

The real number  $x_0$  is the *series expansion point*. Obtaining the  $f(x)$  power series enables us to find the derivatives of  $f(x)$  at the series expansion point, via the Taylor series formula for the coefficients:

$$a_k = \frac{f^{(k)}(x_0)}{k!} \quad (5.2)$$

We can use the evaluation list of  $f(x)$  to generate the series by reinterpreting all the evaluation operations. Instead of using an array of registers holding real numbers and obtaining just  $f(x)$ 's value, we use an array of power series representations, and obtain  $f(x)$ 's power series.

Suppose we want to form the  $f(x)$  power series up to the term in  $(x - x_0)^q$ . At any step in the evaluation process, each active array element defines for some function  $g(x)$  the corresponding power series terms

$$g_0 + g_1(x - x_0) + g_2(x - x_0)^2 + \cdots + g_q(x - x_0)^q$$



Accordingly, instead of a number as previously described, an active array register holds a set of  $q + 1$  power series coefficients  $g_0, g_1, g_2, \dots, g_q$ . Because  $g_0 = g(x_0)$ , we can view our previous use of the evaluation list to obtain an  $f(x)$  value as equivalent to generating single term power series.

Assume now that the evaluation array is an array of vectors, where each vector has  $q + 1$  components and holds the coefficients of some function's power series at a specified series expansion point  $x_0$ . We need to make new interpretations for the two operations of the  $f(x)$  evaluation list that increase the array size, which are the "constant to array end" and "variable to array end" operations. A constant  $c$  has the series representation

$$c = c + 0(x - x_0) + 0(x - x_0)^2 + \dots$$

so the first operation mentioned puts the vector  $(c, 0, 0, \dots, 0)$  at the array end. For the variable  $x$  we have the series representation

$$x = x_0 + 1(x - x_0) + 0(x - x_0)^2 + 0(x - x_0)^3 + \dots$$

so the second operation puts the vector  $(x_0, 1, 0, 0, \dots, 0)$  at the array end.

The five binary operations of an evaluation list need appropriate reinterpretations as series operations. Let  $g_k$  and  $h_k$  denote the coefficients of the two operand functions  $g(x)$  and  $h(x)$ . For addition, subtraction, and multiplication, the series operations are the following:

$$(g + h)_k = g_k + h_k \quad (5.3)$$

$$(g - h)_k = g_k - h_k \quad (5.4)$$

$$(g \cdot h)_k = \sum_{i=0}^k g_i h_{k-i} \text{ or } \sum_{i=0}^k h_i g_{k-i} \quad (5.5)$$

For division we have

$$\sum_{i=0}^{\infty} (g/h)_k (x - x_0)^k = \frac{\sum_{k=0}^{\infty} g_k (x - x_0)^k}{\sum_{k=0}^{\infty} h_k (x - x_0)^k}$$

Setting  $x$  equal to  $x_0$  implies  $(g/h)_0 = g_0/h_0$ . The division equation can be rewritten as

$$\sum_{k=0}^{\infty} g_k (x - x_0)^k = \sum_{k=0}^{\infty} (g/h)_k (x - x_0)^k \cdot \sum_{k=0}^{\infty} h_k (x - x_0)^k$$

Employing the multiplication relation (5.5), we get

$$g_k = \sum_{i=0}^k (g/h)_i h_{k-i} = \begin{cases} (g/h)_0 h_0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} (g/h)_i h_{k-i} + (g/h)_k h_0 & \text{if } k > 0 \end{cases}$$

This leads to a recursive relation for the coefficients  $(g/h)_k$  as follows:

$$(g/h)_k = \begin{cases} g_0/h_0 & \text{if } k = 0 \\ \frac{1}{h_0} \left( g_k - \sum_{i=0}^{k-1} (g/h)_i h_{k-i} \right) & \text{if } k > 0 \end{cases} \quad (5.6)$$

To carry out the division operation, of course we must have  $h_0 \neq 0$ .

For the binary operation of exponentiation,  $g^h$ , there are two procedures, depending on whether the operand function  $h$  is a constant. If  $h$  is not a constant, then we have

$$(g^h)_k = \left( e^{\ln g \cdot h} \right)_k \quad (5.7)$$

and our exponentiation operation is converted to one multiplication and two function evaluations. The unary function evaluation operations are given later in this section.

If  $h$  is a constant  $a$ , then, as with the division operation, upon setting  $x$  to  $x_0$ , it becomes clear that  $(g^a)_0 = (g_0)^a$ . To obtain a general relation for  $(g^a)_k$ , we use the equation

$$\frac{d}{dx} g(x)^a = a g(x)^{a-1} g'(x) \quad (5.8)$$

After multiplication by  $g(x)$ , the equation becomes

$$g(x) \frac{d}{dx} g(x)^a = a g(x)^a g'(x)$$

Converting to power series, we obtain

$$\sum_{k=0}^{\infty} g_k (x - x_0)^k \cdot \sum_{k=1}^{\infty} k (g^a)_k (x - x_0)^{k-1} = a \sum_{k=0}^{\infty} (g^a)_k (x - x_0)^k \cdot \sum_{k=1}^{\infty} k g_k (x - x_0)^{k-1}$$

After we multiply on the right by  $(x - x_0)$ , we get

$$\sum_{k=0}^{\infty} g_k (x - x_0)^k \cdot \sum_{k=1}^{\infty} k (g^a)_k (x - x_0)^k = a \sum_{k=0}^{\infty} (g^a)_k (x - x_0)^k \cdot \sum_{k=1}^{\infty} k g_k (x - x_0)^k$$

which can be rewritten as shown below, with all summation indices starting at 0, because the series term added is 0.

$$\sum_{k=0}^{\infty} g_k (x - x_0)^k \cdot \sum_{k=0}^{\infty} k (g^a)_k (x - x_0)^k = a \sum_{k=0}^{\infty} (g^a)_k (x - x_0)^k \cdot \sum_{k=0}^{\infty} k g_k (x - x_0)^k$$

This leads via the multiplication relation (5.5) to the following equation for the coefficient of  $(x - x_0)^k$ :

$$\sum_{i=0}^k g_i(k-i)(g^a)_{k-i} = a \sum_{i=0}^k i g_i(g^a)_{k-i}$$

or

$$g_0 k (g^a)_k + \sum_{i=1}^k g_i(k-i)(g^a)_{k-i} = 0 + \sum_{i=1}^k a i g_i(g^a)_{k-i}$$

which can be rewritten as

$$g_0 k (g^a)_k = \sum_{i=1}^k (a i + i - k) g_i (g^a)_{k-i}$$

We obtain from this the recurrence relation

$$(g^a)_k = \begin{cases} (g_0)^a & \text{if } k = 0 \\ \frac{1}{g_0} \sum_{i=1}^k \left( \frac{(a+1)i}{k} - 1 \right) (g^a)_{k-i} g_i & \text{if } k > 0 \end{cases} \quad (5.9)$$

When  $g_0 \doteq 0$ , because of the division by  $g_0$ , we cannot form the terms  $(g^a)_k$  for  $k > 0$ . But when  $g_0 \doteq 0$  and  $a$  is a positive integer, the power series expansion  $g^a$  exists; and even if  $a$  is not an integer, but is positive, the Taylor series formula (5.2) indicates that the series terms are defined for  $k < a$ . When  $g_0 \doteq 0$ , for these two cases, an alternate method of obtaining the series terms is needed. If  $a$  is a small positive integer, we can use the relation

$$g^a = \underbrace{g \cdot g \cdots g}_{a \text{ factors}} \quad (5.10)$$

and obtain the series by repeated multiplication. For the other case, a method of specifying the terms can be derived from equation (5.8). This equation can be written in power series form as

$$\sum_{k=1}^{\infty} k (g^a)_k (x - x_0)^{k-1} = \sum_{k=0}^{\infty} a (g^{a-1})_k (x - x_0)^k \cdot \sum_{k=1}^{\infty} k g_k (x - x_0)^{k-1}$$

Multiplying this equation by  $(x - x_0)$  and equating  $(x - x_0)^k$  coefficients, we obtain

$$k (g^a)_k = a \sum_{i=1}^k (g^{a-1})_{k-i} i g_i$$

which leads to the following recursive formula for obtaining the terms of  $g^a$  from those of  $g^{a-1}$ :

$$(g^a)_k = \begin{cases} g_0^{a-1} g_0 & \text{if } k = 0 \\ \frac{a}{k} \sum_{i=1}^k (g^{a-1})_{k-i} i g_i & \text{if } k > 0 \end{cases} \quad (5.11)$$

With this formula we can construct the terms  $(g^a)_0, (g^a)_1, \dots, (g^a)_n$  from the terms  $(g^{a-1})_0, (g^{a-1})_1, \dots, (g^{a-1})_{n-1}$ . Suppose now that  $g_0 \doteq 0$  and that the largest integer less than the positive number  $a$  is  $q$ . We can start with just the single term  $(g^{a-q})_0 = (g_0)^{a-q}$  of the function  $g^{a-q}$ , and then generate in succession the terms for the functions  $g^{a-q+1}, g^{a-q+2}, \dots$ , each time finding one more term than we had previously, until we arrive at the full list of defined terms for  $g^a$ , namely  $(g^a)_0, (g^a)_1, \dots, (g^a)_q$ . All of these terms  $\doteq 0$ , but nevertheless they can be computed this way to define intervals, whereas they cannot be computed via (5.9) because of the division operation.

Next we reinterpret the evaluation operation for each standard function, the natural exponential function first. We have

$$e^{g(x)} = \sum_{k=0}^{\infty} (e^g)_k (x - x_0)^k$$

Taking derivatives, we get

$$\frac{d}{dx} (e^{g(x)}) = e^{g(x)} g'(x)$$

If we multiply this equation on the right by  $(x - x_0)$  and express it in terms of power series, we get

$$\sum_{k=1}^{\infty} k (e^g)_k (x - x_0)^k = \sum_{k=0}^{\infty} (e^g)_k (x - x_0)^k \cdot \sum_{k=1}^{\infty} k g_k (x - x_0)^k$$

Employing the series multiplication relation (5.5) again, we see that for  $k > 0$  we have

$$k (e^g)_k = \sum_{i=1}^k i g_i (e^g)_{k-i}$$

and we obtain the following recursive relation for the coefficients  $(e^g)_k$ :

$$(e^g)_k = \begin{cases} e^{g_0} & \text{if } k = 0 \\ \frac{1}{k} \sum_{i=1}^k i g_i (e^g)_{k-i} & \text{if } k > 0 \end{cases} \quad (5.12)$$

For the natural logarithm we have

$$\frac{d}{dx} \ln g(x) = \frac{g'(x)}{g(x)}$$

After multiplying by  $(x - x_0)$  and then converting to power series, we get

$$\sum_{k=1}^{\infty} k(\ln g)_k (x - x_0)^k = \frac{\sum_{k=1}^{\infty} k g_k (x - x_0)^k}{\sum_{k=0}^{\infty} g_k (x - x_0)^k}$$

Equating the coefficients of  $(x - x_0)^k$  on the two sides of the equals sign, we get, after using the division relation (5.6) for the right-hand side,

$$k(\ln g)_k = \frac{1}{g_0} \left( k g_k - \sum_{i=1}^{k-1} i(\ln g)_i g_{k-i} \right)$$

Note here that the summation is void or empty for  $k = 1$ , because the stopping index is less than the starting index. The void summation becomes clearer if we rewrite our equation as

$$k(\ln g)_k = \frac{1}{g_0} \left( k g_k - \sum_{0 < i < k} i(\ln g)_i g_{k-i} \right)$$

After accounting separately for  $(\ln g)_0$ , we get

$$(\ln g)_k = \begin{cases} \ln g_0 & \text{if } k = 0 \\ \frac{1}{g_0} \left( g_k - \frac{1}{k} \sum_{0 < i < k} i(\ln g)_i g_{k-i} \right) & \text{if } k > 0 \end{cases} \quad (5.13)$$

From the relations

$$\begin{aligned} \frac{d}{dx} \sin g(x) &= \cos g(x) \cdot g'(x) \\ \frac{d}{dx} \cos g(x) &= -\sin g(x) \cdot g'(x) \end{aligned}$$

we obtain in similar fashion the relations

$$\begin{aligned} (\sin g)_k &= \begin{cases} \sin g_0 & \text{if } k = 0 \\ \frac{1}{k} \sum_{i=1}^k i g_i (\cos g)_{k-i} & \text{if } k > 0 \end{cases} \\ (\cos g)_k &= \begin{cases} \cos g_0 & \text{if } k = 0 \\ -\frac{1}{k} \sum_{i=1}^k i g_i (\sin g)_{k-i} & \text{if } k > 0 \end{cases} \end{aligned} \quad (5.14)$$

These equations make it clear that it is necessary to generate both sets of coefficients,  $(\sin g)_k$  and  $(\cos g)_k$ , even when only one set is required.

The hyperbolic functions have similar power series relations

$$\begin{aligned}
 (\sinh g)_k &= \begin{cases} \sinh g_0 & \text{if } k = 0 \\ \frac{1}{k} \sum_{i=1}^k i g_i (\cosh g)_{k-i} & \text{if } k > 0 \end{cases} \\
 (\cosh g)_k &= \begin{cases} \cosh g_0 & \text{if } k = 0 \\ \frac{1}{k} \sum_{i=1}^k i g_i (\sinh g)_{k-i} & \text{if } k > 0 \end{cases}
 \end{aligned} \tag{5.15}$$

Next we consider the inverse trigonometric functions. We have

$$\frac{d}{dx} \tan^{-1} g(x) = \frac{g'(x)}{1 + [g(x)]^2}$$

The coefficients of the denominator function  $1 + g^2$  must be formed; this is easily accomplished as  $1 + g \cdot g$ . Using the division relation (5.6), we get

$$(\tan^{-1} g)_k = \begin{cases} \tan^{-1} g_0 & \text{if } k = 0 \\ \frac{1}{(1 + g^2)_0} \left[ g_k - \frac{1}{k} \sum_{0 < i < k} i (\tan^{-1} g)_i (1 + g^2)_{k-i} \right] & \text{if } k > 0 \end{cases} \tag{5.16}$$

In similar fashion, from

$$\frac{d}{dx} \sin^{-1} g(x) = \frac{g'(x)}{\sqrt{1 - [g(x)]^2}}$$

we obtain

$$(\sin^{-1} g)_k = \begin{cases} \sin^{-1} g_0 & \text{if } k = 0 \\ \frac{1}{(\sqrt{1 - g^2})_0} \left[ g_k - \frac{1}{k} \sum_{0 < i < k} i (\sin^{-1} g)_i (\sqrt{1 - g^2})_{k-i} \right] & \text{if } k > 0 \end{cases} \tag{5.17}$$

The  $\cos^{-1}$  function can be evaluated by using the relation  $\cos^{-1} x = \frac{\pi}{2} - \sin^{-1} x$ .

For the abs, max, and min functions we have

$$\begin{aligned}
 (\text{abs } g)_k &= \begin{cases} k=0 : & |g_0| \\ k>0 : & \begin{cases} g_k & \text{if } g_0 > 0 \\ -g_k & \text{if } g_0 < 0 \\ \text{Not defined} & \text{if } g_0 \doteq 0 \end{cases} \end{cases} \\
 (\text{max } g, h)_k &= \begin{cases} k=0 : & \max(g_0, h_0) \\ k>0 : & \begin{cases} g_k & \text{if } g_0 > h_0 \\ h_k & \text{if } g_0 < h_0 \\ \text{Not defined} & \text{if } g_0 \doteq h_0 \end{cases} \end{cases} \quad (5.18) \\
 (\text{min } g, h)_k &= \begin{cases} k=0 : & \min(g_0, h_0) \\ k>0 : & \begin{cases} g_k & \text{if } g_0 < h_0 \\ h_k & \text{if } g_0 > h_0 \\ \text{Not defined} & \text{if } g_0 \doteq h_0 \end{cases} \end{cases}
 \end{aligned}$$

This completes the description of the process for computing power series of an elementary function  $f(x)$ .

## 5.2 An example of series evaluation

As an example of the general procedure for using power series to obtain derivatives, suppose we want just the first and second derivatives of the simple function  $f(x) = x^2 + 5$  at the point  $x = 3$ . The evaluation list for  $f(x)$  consists of the following five operations:

$$\left[ \begin{array}{l} \text{variable } x \text{ to array end} \\ \text{constant } 2 \text{ to array end} \\ \text{exponentiation} \\ \text{constant } 5 \text{ to array end} \\ + \end{array} \right]$$

We need only generate our power series up to the term  $(x-3)^2$ , so our series coefficient vectors have just three components.

Here the series expansion point is at  $x = 3$ , so the “variable  $x$  to array end” operation constructs the vector  $(3, 1, 0)$ . Because the exponent 2 is a small positive integer, the exponentiation operation of  $x^2$  can be done by multiplying the  $x$  series by itself, giving the result  $(9, 6, 1)$ . The final  $+$  operation yields the vector  $(14, 6, 1)$ , so the series obtained for  $f(x)$  is

$$14 + 6(x-3) + 1(x-3)^2$$

From the series coefficients we find  $f(3) = 14$ ,  $f'(3) = 1! \cdot 6 = 6$ , and  $f''(3) = 2! \cdot 1 = 2$ .

### 5.3 Power series for elementary functions of several variables

The equations developed in Section 5.1 allow the generation of a power series for an elementary function  $f(x)$ , enabling the  $f(x)$  derivatives at the series expansion point to be obtained. Here we obtain similar equations for an elementary function of several variables  $f(x_1, \dots, x_n)$ . These relations enable one to find any  $f$  partial derivative with respect to its variables. Suppose all  $f$  partial derivatives exist in some region  $R$  defined by the  $n$  interval relations  $a_i < x_i < b_i$ ,  $i = 1, \dots, n$ . If a series expansion point  $(x_{1_0}, \dots, x_{n_0})$  is chosen in  $R$ , then  $f$  can be expressed as a power series of the form

$$\sum_{d=0}^{\infty} \sum_{k_1+\dots+k_n=d} f_{k_1, \dots, k_n} (x_1 - x_{1_0})^{k_1} \dots (x_n - x_{n_0})^{k_n} \quad (5.19)$$

The power series term

$$f_{k_1, \dots, k_n} (x_1 - x_{1_0})^{k_1} \dots (x_n - x_{n_0})^{k_n}$$

with coefficient  $f_{k_1, \dots, k_n}$  is said to be of degree  $d$ , where  $d = k_1 + \dots + k_n$ . To obtain the appropriate series relations, let  $(x_1, \dots, x_n)$  be a point in  $R$  close to the series expansion point, and define the function  $g(t)$  by the equation

$$g(t) = f(x_{1_0} + t(x_1 - x_{1_0}), \dots, x_{n_0} + t(x_n - x_{n_0}))$$

Let the variable  $t$  have the domain  $[0, 1]$ , so that  $g(0) = f(x_{1_0}, \dots, x_{n_0})$ , and  $g(1) = f(x_1, \dots, x_n)$ . Expanding  $g(t)$  in a power series about the point  $t = 0$ , we have

$$g(t) = g(0) + \frac{g'(0)}{1!}t + \frac{g''(0)}{2!}t^2 + \dots + \frac{g^{(d)}(0)}{d!}t^d + \dots \quad (5.20)$$

Because  $dg/dt$  at the point  $t = 0$  equals  $\sum_{i=1}^n (x_i - x_{i_0}) \frac{\partial f}{\partial x_i}$  with all partial derivatives taken at the point  $(x_{1_0}, \dots, x_{n_0})$ , the coefficient  $g^{(d)}(0)/d!$  equals

$$\frac{1}{d!} \left[ (x_1 - x_{1_0}) \frac{\partial}{\partial x_1} + \dots + (x_n - x_{n_0}) \frac{\partial}{\partial x_n} \right]^d f(x_{1_0}, \dots, x_{n_0})$$



After the multinomial is expanded, this is

$$\sum_{k_1+\dots+k_n=d} \frac{1}{k_1! \dots k_n!} \left[ \frac{\partial^d}{\partial x_1^{k_1} \dots \partial x_n^{k_n}} f(x_{1_0}, \dots, x_{n_0}) \right] (x_1 - x_{1_0})^{k_1} \dots (x_n - x_{n_0})^{k_n} \quad (5.21)$$

Thus the coefficient  $f_{k_1, \dots, k_n}$  of equation (5.19) equals

$$\frac{1}{k_1! \dots k_n!} \cdot \frac{\partial^d}{\partial x_1^{k_1} \dots \partial x_n^{k_n}} f(x_{1_0}, \dots, x_{n_0})$$

In equation (5.20), when we set  $t$  equal to 1, we obtain

$$f(x_1, \dots, x_n) = \sum_{d=0}^{\infty} \sum_{k_1+\dots+k_n=d} f_{k_1, \dots, k_n} (x_1 - x_{1_0})^{k_1} \dots (x_n - x_{n_0})^{k_n} \quad (5.22)$$

In obtaining the relations for generating the terms of an elementary function  $f(x)$  in Section 5.1, often a derivative with respect to  $x$  was taken, followed by the multiplication of the resulting equation by  $(x - x_0)$ . The two steps can be performed together by using the operator  $(x - x_0) \frac{d}{dx}$ . For a function of the variables  $x_1, \dots, x_n$ , the corresponding operator is

$$(x_1 - x_{1_0}) \frac{\partial}{\partial x_1} + \dots + (x_n - x_{n_0}) \frac{\partial}{\partial x_n} \quad (5.23)$$

Using this operator, we can derive equations for the power series terms of functions with an arbitrary number of variables. The steps are similar to those previously taken. Some additional notation is helpful here. We use the capital letter  $K$  to represent the index  $k_1, \dots, k_n$ , so the coefficient  $g_{k_1, \dots, k_n}$  of the series term

$$g_{k_1, \dots, k_n} (x_1 - x_{1_0})^{k_1} \dots (x_n - x_{n_0})^{k_n}$$

becomes  $g_K$ . The capital letter  $O$  is reserved for the index  $0, \dots, 0$ . Define  $[K]$  to be the integer  $k_1 + \dots + k_n$ , the degree of the term  $g_K$ . For two indices  $I$  and  $J$ , define  $I - J$  to be the index  $Q$  with  $q_1 = i_1 - j_1, \dots, q_n = i_n - j_n$ . The notation  $I = J$  has the obvious interpretation, and  $I \leq J$  signifies

$$i_1 \leq j_1, \dots, i_n \leq j_n \quad (5.24)$$

The notation  $I < J$  also signifies (5.24), but with the requirement that inequality occurs at least once. Thus if  $[I] = [J]$ , which implies that  $g_I$  and  $g_J$  have

the same degree, then neither  $I < J$  nor  $I > J$  is possible. The following equations can be derived by repeating the steps of Section 5.1:

$$(g + h)_K = g_K + h_K$$

$$(g - h)_K = g_K - h_K$$

$$(g \cdot h)_K = \sum_{0=I \leq K} g_I h_{K-I} \text{ or } \sum_{0=I \leq K} g_{K-I} h_I$$

$$(g/h)_K = \begin{cases} g_0/h_0 & \text{if } K = 0 \\ \frac{1}{h_0} \left( g_K - \sum_{0=I < K} (g/h)_I h_{K-I} \right) & \text{if } K > 0 \end{cases}$$

$$(e^g)_K = \begin{cases} e^{g_0} & \text{if } K = 0 \\ \frac{1}{[K]} \sum_{0 < I \leq K} [I] g_I (e^g)_{K-I} & \text{if } K > 0 \end{cases}$$

$$(\ln g)_K = \begin{cases} \ln g_0 & \text{if } K = 0 \\ \frac{1}{g_0} \left( g_K - \frac{1}{[K]} \sum_{0 < I < K} [I] (\ln g)_I g_{K-I} \right) & \text{if } K > 0 \end{cases}$$

$$(\sin g)_K = \begin{cases} \sin g_0 & \text{if } K = 0 \\ \frac{1}{[K]} \sum_{0 < I \leq K} [I] g_I (\cos g)_{K-I} & \text{if } K > 0 \end{cases}$$

$$(\cos g)_K = \begin{cases} \cos g_0 & \text{if } K = 0 \\ -\frac{1}{[K]} \sum_{0 < I \leq K} [I] g_I (\sin g)_{K-I} & \text{if } K > 0 \end{cases}$$

$$(\sinh g)_K = \begin{cases} \sinh g_0 & \text{if } K = 0 \\ \frac{1}{[K]} \sum_{0 < I \leq K} [I] g_I (\cosh g)_{K-I} & \text{if } K > 0 \end{cases}$$

$$(\cosh g)_K = \begin{cases} \cosh g_0 & \text{if } K = 0 \\ \frac{1}{[K]} \sum_{0 < I \leq K} [I] g_I (\sinh g)_{K-I} & \text{if } K > 0 \end{cases}$$

$$(g^a)_K = \begin{cases} (g_0)^a & \text{if } K = 0 \\ \frac{1}{g_0} \sum_{0 < I \leq K} \left( \frac{(a+1)[I]}{[K]} - 1 \right) (g^a)_{K-I} g_I & \text{if } K > 0 \end{cases}$$

$$(g^a)_K = \begin{cases} g_0^{a-1} g_0 & \text{if } K = 0 \\ \frac{a}{[K]} \sum_{0 < I \leq K} (g^{a-1})_{K-I} [I] g_I & \text{if } K > 0 \end{cases}$$

$$\begin{aligned}
(\tan^{-1} g)_K &= \begin{cases} \tan^{-1} g_O & \text{if } K = O \\ \frac{1}{(1+g^2)_O} \left[ g_K - \frac{1}{[K]} \sum_{O < I < K} [I] (\tan^{-1} g)_I (1+g^2)_{K-I} \right] & \text{if } K > O \end{cases} \\
(\sin^{-1} g)_K &= \begin{cases} \sin^{-1} g_O & \text{if } K = O \\ \frac{1}{(\sqrt{1-g^2})_O} \left[ g_K - \frac{1}{[K]} \sum_{O < I < K} [I] (\sin^{-1} g)_I (\sqrt{1-g^2})_{K-I} \right] & \text{if } K > O \end{cases} \\
(\text{abs } g)_K &= \begin{cases} K = O : |g_O| \\ K > O : \begin{cases} g_K & \text{if } g_O > 0 \\ -g_K & \text{if } g_O < 0 \\ \text{Not defined} & \text{if } g_O \doteq 0 \end{cases} \end{cases} \\
(\max g, h)_K &= \begin{cases} K = O : \max(g_O, h_O) \\ K > O : \begin{cases} g_K & \text{if } g_O > h_O \\ h_K & \text{if } g_O < h_O \\ \text{Not defined} & \text{if } g_O \doteq h_O \end{cases} \end{cases} \\
(\min g, h)_K &= \begin{cases} K = O : \min(g_O, h_O) \\ K > O : \begin{cases} g_K & \text{if } g_O < h_O \\ h_K & \text{if } g_O > h_O \\ \text{Not defined} & \text{if } g_O \doteq h_O \end{cases} \end{cases}
\end{aligned}$$

These formulas allow the generation of the power series for a function of any number of variables.

## 5.4 A more general method of generating power series

To generate the power series of an elementary function  $f(x)$  of just one variable, the evaluation array can be an array of vectors. Here an active vector's components give the coefficients of some intermediate function's power series, with the coefficient of the  $(x-x_0)^i$  series term held in vector component  $i+1$ . To generate the power series of a function  $f(x_1, \dots, x_n)$  of several variables, a more flexible series representation system is needed. The evaluation array now must be an array of lists, each list defining some intermediate function's series coefficients. For instance, suppose the function  $f$  whose power series we want has three variables  $x$ ,  $y$ , and  $z$ . Then an intermediate function  $g(x, y, z)$  is represented by a list giving the successive terms of the power series

$$\sum g_{i_1, i_2, i_3} (x-x_0)^{i_1} (y-y_0)^{i_2} (z-z_0)^{i_3}$$

A list element holds a coefficient  $g_{i_1, i_2, i_3}$ , the three integers  $i_1$ ,  $i_2$ , and  $i_3$  that identify the coefficient, and one other integer  $i_0$  equal to the degree  $i_1 + i_2 + i_3$  of the series term. Thus each list element has the ranged value of a coefficient and an identifying four-integer index vector  $(i_0, i_1, i_2, i_3)$ . If  $R$  is the value of the coefficient, then the list element will be denoted  $(i_0, i_1, i_2, i_3)[R]$ . For instance, suppose the  $f(x, y, z)$  series expansion point is  $(x_0, y_0, z_0) = (3, 4, 5)$ . Before attempting to generate the series for  $f(x, y, z)$ , the series for each variable is prepared in advance. For our example, these series are

$$x \text{ series : } (0, 0, 0, 0)[3] + (1, 1, 0, 0)[1]$$

$$y \text{ series : } (0, 0, 0, 0)[4] + (1, 0, 1, 0)[1]$$

$$z \text{ series : } (0, 0, 0, 0)[5] + (1, 0, 0, 1)[1]$$

Here we use a + sign to separate the elements on a series list. Suppose now that  $f(x, y, z)$  is the simple function  $xy + z + 10$ . The evaluation list for this function is

$$\left[ \begin{array}{l} \text{variable } x \text{ to array end} \\ \text{variable } y \text{ to array end} \\ \times \\ \text{variable } z \text{ to array end} \\ + \\ \text{constant } 10 \text{ to array end} \\ + \end{array} \right]$$

The series obtained after completing the multiplication operation is

$$(0, 0, 0, 0)[12] + (1, 0, 1, 0)[3] + (1, 1, 0, 0)[4] + (2, 1, 1, 0)[1]$$

and the final series generated after completing the second addition operation is

$$(0, 0, 0, 0)[27] + (1, 0, 0, 1)[1] + (1, 0, 1, 0)[3] + (1, 1, 0, 0)[4] + (2, 1, 1, 0)[1]$$

indicating that at the point  $(3, 4, 5)$  we have

$$f = 27, \quad \frac{\partial f}{\partial z} = 1, \quad \frac{\partial f}{\partial y} = 3, \quad \frac{\partial f}{\partial x} = 4 \text{ and } \frac{\partial^2 f}{\partial x \partial y} = 1$$

Here it is convenient to order the series terms by the index four-tuple, with index  $I$  preceding index  $J$  if, comparing index components in order and ignoring equal components, the first smaller component belongs to index  $I$ . This way, the constant term of a series is always first because its index is  $(0, 0, 0, 0)$ , and a degree  $d$  term always precedes a degree  $d + 1$  term.

## 5.5 The demo program `deriv`

This program computes  $f(x, y, z)$  series in the manner described in the preceding section, and is then able to obtain  $f$  derivatives at the series evaluation point  $(x_0, y_0, z_0)$  from the series coefficients  $f_{i,j,k}$ , using the formula

$$f_{i,j,k} = \frac{1}{i!j!k!} \cdot \frac{\partial^{i+j+k}}{\partial x^i \partial y^j \partial z^k} f(x_0, y_0, z_0)$$

If the derivatives found are not accurate enough to satisfy the accuracy requirement, then as is standard with range arithmetic, the computation is repeated at an appropriate higher precision.

## Software Exercises C

These exercises show some features of the `deriv` demo program.

1. Call up the program `deriv` to find the first eight derivatives of the function  $\sin x$  at the argument point  $x = 0$ , specifying, for example, 10 fixed-point decimal places.
2. Because `deriv` creates a log file, it is easy to change the point at which derivatives are found. Edit the `deriv.log` file, changing the argument point from 0 to  $\pi/2$ , save the file, and then call up `deriv deriv`.
3. Change the **PNM** command back to `deriv`, and then call up `deriv` to compute all  $\cos(x + y + z)$  derivatives of order three or less, at the origin. After the display of derivatives appears at your console, return to the **PNM** form and look at the `deriv.prt` file to see them again.
4. Call up `deriv` to obtain the first five derivatives of  $x^{3.2}$  at the point  $x = 1$ . Next edit the log file to change the evaluation point from 1 to 0. Save the log file, and then call up `deriv deriv`. Notice that only derivatives to the fourth order are obtained, because the function's series coefficients can be found only up to the term  $f_4(x - 0)^4$ .

## Notes and References

- A. R. Moore was an early advocate of power series methods for finding derivatives and integrals. We have used the notation of his book [1]. For a more complete discussion of derivatives and power series, see the book by Rall [2].

- B. Various functions that are not elementary are often encountered in applied mathematics. Examples are the gamma and beta functions, the Bessel functions, and the hypergeometric functions. There is no basic difficulty in extending the elementary function class to include additional function types, as long as the new function types can be generated using specific power series formulas (as is the case for the standard functions).
- [1] Moore, R. E., *Methods and Applications of Interval Analysis*, SIAM Studies in Applied Mathematics, SIAM, Philadelphia, 1979, 24–29.
- [2] Rall, L. B., *Automatic Differentiation: Techniques and Applications*, Lecture Notes in Computer Science 120, Springer-Verlag, Berlin, 1981.

This page intentionally left blank

# Computing Integrals



Three demo programs compute integrals accurately. The program `integ` computes ordinary definite integrals, the program `mulint` computes higher dimensional integrals, and the program `impint` computes improper integrals.

## 6.1 Computing a definite integral

Suppose  $f(x)$  is an elementary function defined in  $[a, b]$ , and  $f(x)$  is infinitely differentiable in this interval except, possibly, at a finite number of points. We describe a method of accurately computing the integral  $\int_a^b f(x) dx$  using  $f(x)$  power series expansions. Methods for forming power series were described in the preceding chapter.

Assume, temporarily, that  $f(x)$  is infinitely differentiable in  $[a, b]$ . We can form a power series for  $f(x)$ , taking the  $[a, b]$  interval midpoint  $x_0 = (a + b)/2$  as the series expansion point, with the degree  $n$  of the last series term reasonably large, for example 12 or higher. This finite series, often called a Taylor polynomial, is

$$f_0 + f_1(x - x_0) + f_2(x - x_0)^2 + \cdots + f_n(x - x_0)^n$$

which may be written more explicitly as

$$f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

and more compactly as

$$\sum_{i=0}^n \frac{f^{(i)}(x_0)}{i!} (x - x_0)^i$$



We can then integrate the Taylor polynomial to obtain an approximation to  $\int_b^a f(x) dx$ .

Thus it is easy this way to approximate our integral. The more difficult task is to somehow bound the error made in replacing  $f(x)$  by the Taylor polynomial. Here we can use Taylor's formula for the remainder, given in the next theorem, and proven in most calculus texts:

**Theorem 6.1** Suppose the function  $f(x)$  is defined and continuous in  $[a, b]$  and has  $n + 1$  derivatives there. Then for any point  $x_0$  in  $(a, b)$  and any point  $x$  in  $[a, b]$  unequal to  $x_0$ , there exists a point  $z$  between  $x$  and  $x_0$  such that

$$f(x) = \sum_{i=0}^n \frac{f^{(i)}(x_0)}{i!} (x - x_0)^i + \frac{f^{(n+1)}(z)}{(n+1)!} (x - x_0)^{n+1} \quad (6.1)$$

The degree  $n + 1$  term is called the *remainder*.

Let us apply this result to our integration problem. Eq. (6.1) can be rewritten as

$$f(x) = \sum_{i=0}^{n+1} \frac{f^{(i)}(x_0)}{i!} (x - x_0)^i + \frac{f^{(n+1)}(z) - f^{(n+1)}(x_0)}{(n+1)!} (x - x_0)^{n+1}$$

Here we have added a degree  $n + 1$  term to the Taylor polynomial and then subtracted it out. The degree of the Taylor polynomial is now  $n + 1$ , but it is more convenient to have  $n$  for this degree, so we rewrite the equation replacing  $n + 1$  by  $n$ :

$$f(x) = \sum_{i=0}^n \frac{f^{(i)}(x_0)}{i!} (x - x_0)^i + \frac{f^{(n)}(z) - f^{(n)}(x_0)}{n!} (x - x_0)^n$$

We finally have a bound for the error of the Taylor polynomial:

$$\left| f(x) - \sum_{i=0}^n \frac{f^{(i)}(x_0)}{i!} (x - x_0)^i \right| = \left| \frac{f^{(n)}(z) - f^{(n)}(x_0)}{n!} (x - x_0)^n \right|$$

The point  $z$  depends on  $x$  and is in general an unknown point. Instead of trying to find  $z$ , we can find the maximum distance of  $f^{(n)}(x)$  from  $f^{(n)}(x_0)$  as  $x$  varies in the interval  $[a, b]$ . If  $W$  is this nonnegative value, then the error of the Taylor polynomial for any  $x$  in  $[a, b]$  is  $\leq \frac{W}{n!} |x - x_0|^n$ . Thus our error bound is 0 if  $x = x_0$ , and rises to its maximum value when  $x$  equals an endpoint of the  $[a, b]$  interval.

This error bound also can be expressed compactly in interval form, using the simpler power series notation:

$$f(x) = f_0 + f_1(x - x_0) + f_2(x - x_0)^2 + \cdots + f_{n-1}(x - x_0)^{n-1} + (f_n \pm w_n)(x - x_0)^n \quad (6.2)$$

The interval  $f_n \pm w_n$  has the midpoint  $f_n = f^{(n)}(x_0)/n!$  and the halfwidth  $w_n = W/n!$ . The term  $(f_n \pm w_n)(x - x_0)^n$  should be understood to mean  $f_n(x - x_0)^n \pm w_n|x - x_0|^n$ , that is, multiplication by  $(x - x_0)^n$  is treated differently for the two parts of  $f_n \pm w_n$ .

If  $h$  is taken equal to the halfwidth  $(b - a)/2$  of the interval  $[a, b]$ , then the integral of the polynomial (6.2) over the interval  $[x_0 - h, x_0 + h]$  gives an approximation to  $\int_a^b f(x) dx$ , and the approximation's error bound is obtained by integrating  $w_n|x - x_0|^n$ . Here it is convenient to take  $n$ , the polynomial's degree, as an even integer  $2q$ , because then  $|x - x_0|^{2q} = (x - x_0)^{2q}$ . We then obtain

$$\begin{aligned} \int_{x_0-h}^{x_0+h} f(x) dx &= \int_{x_0-h}^{x_0+h} \sum_{i=0}^{2q} f_i(x - x_0)^i dx \pm \int_{x_0-h}^{x_0+h} w_{2q}(x - x_0)^{2q} dx \\ &= 2 \left[ f_0 h + f_2 \frac{h^3}{3} + f_4 \frac{h^5}{5} + \cdots + f_{2q} \frac{h^{2q+1}}{2q+1} \right] \pm 2w_{2q} \frac{h^{2q+1}}{2q+1} \quad (6.3) \end{aligned}$$

The program `integ` obtains definite integrals to a prescribed number of correct decimal places, and it does this by dividing the integration interval  $[a, b]$  into subintervals small enough that their error terms become acceptably small. This requires a method of determining the bound  $w_{2q}$  for each subinterval, and we consider this problem next.

## 6.2 Formal interval arithmetic

In Chapter 2, we introduced Moore's interval arithmetic as a means of monitoring the computation error of range arithmetic. We can use interval arithmetic again in a different fashion to obtain a bound  $w_{2q}$  to be assigned to  $f_{2q}$ . The nonnegative number  $w_{2q}$  is a bound on the distance of the  $f_{2q}$  value at an arbitrary point in  $[a, b]$  from the specific  $f_{2q}$  value determined at the interval midpoint  $x_0$ .

The general idea here is to do the computation for the  $f(x)$  power series as described in Chapter 5, except that an interval  $m \pm w$  represents each series coefficient in the computation, and that interval arithmetic is used everywhere. It is true that range arithmetic itself is a form of interval arithmetic, but a ranged number has only a gross representation of halfwidth, as a means of determining an appropriate precision of computation. In this application we want a more accurate representation of interval halfwidths. So one ranged number is used for each interval midpoint  $m$ , and one ranged number is used for each interval halfwidth  $w$ . After the computation is complete, as usual we can check our ranged results to determine whether we have obtained enough correct decimal places, and if not, the whole computation is repeated at a higher precision. The series representation for  $x$  now is

$$x = (x_0 \pm w_0) + (1 \pm 0)(x - x_0)$$

with  $x_0 = (a + b)/2$  and  $w_0 = (b - a)/2$ . Note that each of the two coefficients correctly bounds the distance of the coefficient computed at an arbitrary point in  $[a, b]$  from the coefficient value computed at  $x_0$ . The series representation for a constant  $a$  is

$$a = (a \pm 0)$$

There are only two operations on a function's evaluation list that can increase the number of intermediate series, and these are the "variable to array end" and "constant to array end" operations. Both operations introduce a series with correct coefficient bounds, and so if all other operations use interval arithmetic in computing coefficients, then the final coefficients have bounds which are guaranteed not to be too small.

We repeat below the rules of interval arithmetic. The division operation has a slightly different form than previously, in that the  $\doteq$  relation is used, because now the two quantities defining an interval are both ranged numbers, not rational numbers as previously.

$$m_1 \pm w_1 + m_2 \pm w_2 = (m_1 + m_2) \pm (w_1 + w_2) \quad (6.4)$$

$$m_1 \pm w_1 - m_2 \pm w_2 = (m_1 - m_2) \pm (w_1 + w_2) \quad (6.5)$$

$$m_1 \pm w_1 \times m_2 \pm w_2 = (m_1 m_2) \pm (w_1 |m_2| + |m_1| w_2 + w_1 w_2) \quad (6.6)$$

$$m_1 \pm w_1 \div m_2 \pm w_2 = \begin{cases} \left( \frac{m_1}{m_2} \right) \pm \left( \frac{w_1 + \frac{|m_1|}{|m_2|} w_2}{|m_2| - w_2} \right) & \text{if } |m_2| > w_2 \\ \text{Division error if } |m_2| \doteq w_2 \text{ or } |m_2| < w_2 \end{cases} \quad (6.7)$$

When the steps of the  $f(x)$  evaluation list are done using the interval arithmetic relations just listed, we will call the computation "formal interval arithmetic", range arithmetic being the "informal" variety. Instead of obtaining for  $f(x)$  the series

$$f_0 + f_1(x - x_0) + f_2(x - x_0)^2 + \cdots + f_n(x - x_0)^n$$

we now obtain for  $f(x)$  the representation

$$(f_0 \pm w_0) + (f_1 \pm w_1)(x - x_0) + \cdots + (f_n \pm w_n)(x - x_0)^n$$

With  $n$  taken as the even integer  $2q$ , only the halfwidth  $w_{2q}$  is used. But to obtain it we must compute halfwidths for all coefficients of all the intermediate series.

Most unary series operations are function evaluations, and these now need to be done in formal interval arithmetic style. Generally for each standard function it is not difficult to obtain an appropriate interval arithmetic relation. As an example, consider the natural logarithm function. Because  $\ln x$  is an increasing function with a decreasing derivative, we have

$$\ln(m \pm w) = \ln m \pm (\ln m - \ln(m - w))$$

Two logarithm evaluations are required here, and each evaluation is likely to require many arithmetic operations. To speed up logarithm evaluation, one might prefer to use the less accurate relation

$$\ln(m \pm w) = \ln m \pm \frac{w}{m - w}$$

obtained by applying the Mean Value Theorem:

$$\ln m - \ln(m - w) = (\ln' z)w = \frac{1}{z} \cdot w \leq \frac{w}{m - w}$$

Thus with the standard functions, there are often several choices for the interval arithmetic evaluation equations. We list below some of the relations used by our demo programs for the standard functions.

$$e^{m \pm w} = e^m \pm e^m(e^w - 1) \quad (6.8)$$

$$\ln(m \pm w) = \begin{cases} \ln m \pm \frac{w}{m - w} & \text{if } m > w \\ \ln \text{ error} & \text{if } m < w \text{ or } m = w \end{cases} \quad (6.9)$$

$$\sin(m \pm w) = \sin m \pm \min(|\cos m| + w, 1) \cdot w \quad (6.10)$$

$$\cos(m \pm w) = \cos m \pm \min(|\sin m| + w, 1) \cdot w \quad (6.11)$$

$$\sinh(m \pm w) = \sinh m \pm \cosh m \cdot (e^w - 1) \quad (6.12)$$

$$\cosh(m \pm w) = \cosh m \pm \cosh m \cdot (e^w - 1) \quad (6.13)$$

$$\tan^{-1}(m \pm w) = \tan^{-1} m \pm \frac{w}{1 + [\max(|m| - w, 0)]^2} \quad (6.14)$$

### 6.3 The demo program `integ` for computing ordinary definite integrals

In finding an approximation with an error bound to  $\int_a^b f(x) dx$ , it is convenient to arrange the computation by means of a “task queue”. A task queue often is a convenient way of handling computations where the exact sequence of steps needed is not known in advance. A task queue finds many applications in precise computation programs. In general a task queue is a list of the tasks that need to be performed to reach the objective. Depending on the problem, the queue initially holds a single task, or a short list of tasks, which would attain the objective; but generally it is not known whether these tasks can be performed successfully. Cyclically, the task at the head of the queue is attempted, then discarded from the queue if the task is completed successfully. This process continues until the task queue is empty, because the objective has been attained. If the task at the

head of the queue cannot be done, the task is split into an appropriate set of simpler subtasks that accomplish the same end, and these subtasks replace the task at the head of the queue.

When dealing with definite integrals, it is convenient to always have  $[0, 1]$  as the integration interval. If we make the substitution  $x = a + (b - a)u$ , the definite integral  $\int_a^b f(x) dx$  can be rewritten as  $\int_0^1 F(u) du$ , where  $F(u)$  equals  $[b - a]f(a + [b - a]u)$ . If our integral is required to  $k$  correct fixed-place decimal places, the maximum error bound  $\varepsilon$  allowable can be computed in advance [see Eq. (3.1)]. We must obtain an integral approximation with an error bound no larger than  $\varepsilon$ . The task queue will hold a list of  $u$  subintervals, and the integral of  $F(u)$  must be found for each. Initially the queue holds the single  $u$  interval  $0.5 \pm 0.5$ . In general, if  $u_0 \pm h$  is the interval at the head of the queue, the queue cycle consists of calculating an integral approximation over  $u_0 \pm h$ , and checking that the error bound  $2w_{2q} \frac{h^{2q+1}}{2q+1}$  does not exceed its  $\varepsilon$  allotment, which is

$$\varepsilon \cdot \frac{\text{length } [u_0 - h, u_0 + h]}{\text{length } [0, 1]} = 2h\varepsilon$$

If this requirement is met, the integral approximation is added to a sum  $S$ , the error bound is added to another sum  $E$ , both sums initially zero, and the queue interval is discarded. Otherwise, the queue interval is bisected and the two subintervals replace it on the task queue. The error bound of an integral approximation has the multiplier  $h^{2q+1}$ , so when an interval is bisected, and  $h$  is halved, this error bound multiplier decreases by the factor  $\frac{1}{2^{2q+1}}$ , whereas the  $\varepsilon$  allotment decreases only by the factor  $1/2$ . So eventually, when  $h$  gets small enough, the error bound test is passed. When finally the task queue is empty,  $S$  holds the required approximation to  $\int_0^1 F(u) du$  and  $E$  holds the error bound, which does not exceed  $\varepsilon$ .

Whenever the integral error bound for a queue interval is more than its allotment, it is advisable to test whether the precision of computation is adequate and to increase it if not. Such a test is needed at some point in any range arithmetic program.

It may happen that the  $F(u)$  series cannot be computed for a queue interval, because the interval is too large. Such a series generation error may occur, for instance, on a division operation  $g(u)/h(u)$ , when the leading term of  $h(u)$  is an interval containing the zero point. Or a series error may occur when generating  $\ln g(u)$  if the leading term of  $g(u)$  contains the zero point in its interval. These cases are treated just as if the  $\varepsilon$  allotment were exceeded.

The procedure, as so far described, suffices when the function  $F(u)$  is analytic at every point of  $[0, 1]$ . (A function is *analytic* at a point  $u_0$  if it has derivatives of arbitrary order there.) But it is possible that  $F(u)$  is defined, but is not analytic at a few points inside  $[0, 1]$  or at an endpoint. For instance the integral might be  $\int_0^1 \sqrt{1 - u^3} du$  with the nonanalytic point  $u = 1$ , or  $\int_0^1 (u - \frac{1}{2})^{1/3} du$  with the nonanalytic point  $u = 1/2$ . It will not be possible to obtain any series coefficients

beyond the constant coefficient when the integration interval  $u_0 \pm h$  contains such a point. The  $F(u)$  series then has just the leading series term  $f_0 \pm w_0$ .

However, equation (6.3) is valid even when  $q$  is zero, and we obtain for the integral the value  $2f_0h$  and the error bound  $2w_0h$ . The halfwidth  $w_0$  decreases as  $h$  decreases, so the error  $2w_0h$  eventually becomes less than the error allotment  $2\epsilon h$  when  $h$  becomes small enough. Normally an interval  $u_0 \pm h$  containing a nonanalytic point is extremely small before it passes this test.

## 6.4 Taylor's remainder formula generalized

In the next section the demo program `multint` for computing multiple integrals is described. That program must deal with integrand functions having several variables, so it will be advantageous here to generalize the Taylor remainder formula for such functions.

For functions  $f(x)$  of one variable, we have seen that if we choose the series expansion point  $x_0$  as the midpoint of  $[a, b]$ , take the  $x$  series to be

$$\left(x_0 \pm \frac{b-a}{2}\right) + (1 \pm 0)(x - x_0)$$

and use formal interval arithmetic to obtain for  $f(x)$  the series

$$(f_0 \pm w_0) + (f_1 \pm w_1)(x - x_0) + \cdots + (f_{m-1} \pm w_{m-1})(x - x_0)^{m-1} \\ + (f_m \pm w_m)(x - x_0)^m$$

of maximum degree  $m$ , then Taylor's remainder formula implies that we are justified to use the relation

$$f(x) = f_0 + f_1(x - x_0) + \cdots + f_{m-1}(x - x_0)^{m-1} + (f_m \pm w_m)(x - x_0)^m \quad (6.15)$$

We need corresponding relations for a function  $f(x_1, \dots, x_n)$  of  $n$  variables, analytic in a region  $R$  that can be expressed as a product of intervals:

$$R = \prod_{i=1}^n [a_i, b_i]$$

Here each variable  $x_i$  is restricted to the interval  $[a_i, b_i]$ . The series expansion point for the  $f$  series is taken as  $(x_{1_0}, \dots, x_{n_0})$ , where  $x_{i_0}$  is the midpoint of  $[a_i, b_i]$ . The series for a variable  $x_i$  is then

$$x_i = \left(x_{i_0} \pm \frac{b_i - a_i}{2}\right) + (1 \pm 0)(x_i - x_{i_0})$$

We obtain for  $f$  the series

$$\sum_{d=0}^m \sum_{k_1 + \cdots + k_n = d} (f_{k_1, \dots, k_n} \pm w_{k_1, \dots, k_n})(x_1 - x_{1_0})^{k_1} \cdots (x_n - x_{n_0})^{k_n}$$

Suppose  $(x_1, \dots, x_n)$  is some point in  $R$ . We see that the expression corresponding to (6.15) is

$$\begin{aligned} f(x_1, \dots, x_n) &= \sum_{d=0}^{m-1} \sum_{k_1+\dots+k_n=d} f_{k_1, \dots, k_n} (x_1 - x_{1_0})^{k_1} \dots (x_n - x_{n_0})^{k_n} \\ &+ \sum_{k_1+\dots+k_n=m} (f_{k_1, \dots, k_n} \pm w_{k_1, \dots, k_n}) (x_1 - x_{1_0})^{k_1} \dots (x_n - x_{n_0})^{k_n} \end{aligned}$$

All the terms of degree  $m$  for  $f$ , each having an interval coefficient, arise from the single term of degree  $m$  for the function  $g(t)$  defined in Section 5.3. Now there are many series coefficients that retain their interval form, instead of only one, as was previously the case for  $f(x)$ .

## 6.5 The demo program `mulint` for higher dimensional integrals

The program `mulint`, like the program `integ`, uses a task queue to compute its integrals, but differs from `integ` in its requirements for the integrand function. With `integ`, the integrand  $f(x)$  can have a finite number of points where it is not analytic, but just defined and continuous. The program `mulint` requires its integrand function to be analytic throughout the integration domain. To see the reason for this change, consider a typical problem for `mulint`, the iterated integral  $\int_a^b \int_{g_1(x)}^{g_2(x)} f(x, y) dx dy$ . Here the region of integration defined by the limit functions  $g_1(x)$  and  $g_2(x)$  over their domain  $[a, b]$  defines a bounded region  $R$ .

The integration is converted to integration over the unit  $(u, v)$  square by making the substitutions

$$\begin{aligned} y &= g_1(x) + [g_2(x) - g_1(x)]v \\ x &= a + (b - a)u \end{aligned} \tag{6.16}$$

Here  $u$  and  $v$  both vary in  $[0, 1]$ . The integral then has the form  $\int_0^1 \int_0^1 F(u, v) du dv$  where

$$F(u, v) = (b - a)[g_2(x) - g_1(x)]f(x, y)$$

with  $x$  and  $y$  now being functions of  $u$  and  $v$ . The task queue holds a list of  $(u, v)$  subrectangles  $(u_0 \pm h_1) \times (v_0 \pm h_2)$  over which integration needs to be done.

Now we can see the reason for requiring  $f(x, y)$  to be analytic in its domain. If  $f(x, y)$  is not analytic, then in general there would be one or more curves in the  $(u, v)$  square where  $f(x, y)$  would be defined but not analytic, and this would mean that a single term series for the integrand function would be needed

over a collection of tiny rectangles covering the curves, making the integration problem computerbound for a PC. The program `mulint` requires that the limit functions  $g_1(x)$  and  $g_2(x)$  be analytic for the same reason, that is, to keep the integration problem from becoming computerbound.

Assume now that  $f(x, y)$ ,  $g_1(x)$ , and  $g_2(x)$  are analytic in their domains. As `mulint` begins the integration problems, the task queue holds just the square  $(.5 \pm .5) \times (.5 \pm .5)$ . To describe the general task queue cycle, let us take the integration rectangle at the head of the task queue to be  $(u_0 \pm h_1) \times (v_0 \pm h_2)$ . The variables  $u$  and  $v$  are set to

$$u = (u_0 \pm h_1) + (1 \pm 0)(u - u_0)$$

and

$$v = (v_0 \pm h_2) + (1 \pm 0)(v - v_0)$$

For the function  $F(u, v)$  we obtain the degree  $2q$  series

$$\sum_{d=0}^{2q} \sum_{i+j=d} (F_{i,j} \pm w_{i,j})(u - u_0)^i (v - v_0)^j$$

from which the integral approximation

$$\sum_{r=0}^q \sum_{s+t=r} \frac{F_{2s,2t} h_1^{2s+1} h_2^{2t+1}}{(2s+1)(2t+1)}$$

is obtained, as is the error bound

$$\sum_{i+j=2q} \frac{w_{i,j} h_1^{i+1} h_2^{j+1}}{(i+1)(j+1)}$$

If  $\varepsilon$  is the error bound that must be achieved by our final integral approximation in order to obtain  $k$  correct decimal places from it, we need to check that our subrectangle error bound is not greater than the allotment  $4h_1 h_2 \varepsilon$ .

When the error allotment is exceeded, the integration subrectangle is divided into two equal subrectangles, obtained by halving either the  $u$  dimension or the  $v$  dimension. The division method is chosen by recomputing the error bound with first  $h_1$  replaced by  $h_1/2$ , and then  $h_2$  replaced by  $h_2/2$ , and letting the smaller value determine the division method. If there is a series generation error, then the larger side of the subrectangle is bisected.

The integration method, described for two dimensional integrals, can be easily generalized to apply to integrals of higher dimension.



## 6.6 The demo program `impint` for computing improper integrals

The program `impint` computes various one-dimensional improper integrals. The improper integral can be of the form  $\int_a^b f(x) dx$ , where the function  $f(x)$  is not defined throughout  $[a, b]$ , but only in either  $(a, b]$ ,  $[a, b)$ , or  $(a, b)$ . Another common improper integral is  $\int_a^\infty f(x) dx$ . Here  $f(x)$  may be defined in  $[a, \infty)$  or only in  $(a, \infty)$ . A similar improper integral is  $\int_{-\infty}^b f(x) dx$ , with  $f(x)$  defined in  $(-\infty, b]$  or only in  $(-\infty, b)$ . Finally there is the improper integral  $\int_{-\infty}^\infty f(x) dx$ .

Consider first the integral  $\int_a^b f(x) dx$ , where  $f(x)$  is defined only in  $(a, b]$ . For example, the integral could be  $\int_0^\pi \frac{\cos x}{\sqrt{x}} dx$ . The program `impint` integrates the function  $f(x)$  from a point  $\alpha$ , slightly to the right of  $a$ , to the endpoint  $b$ , using the integration method described in Section 6.3. To account for the unintegrated stretch  $[a, \alpha]$ , the program `impint` requires the user to specify an error-limiting positive function  $g(x)$ , defined in some small interval  $[a, b_1]$ , such that  $|\int_a^x f(t) dt| \leq g(x)$  for  $x$  in  $(a, b_1]$ , with the limit  $\lim_{x \rightarrow a^+} g(x) = 0$  being valid. This ensures that by moving  $\alpha$  close enough to  $a$ , the error made in omitting the stretch  $[a, \alpha]$  can be made arbitrarily small.

Usually it is not difficult to find a suitable function  $g(x)$ . For the example problem, we have

$$\left| \int_0^x \frac{\cos t}{\sqrt{t}} dt \right| \leq \int_0^x \frac{|\cos t|}{\sqrt{t}} dt \leq \int_0^x \frac{dt}{\sqrt{t}} = 2\sqrt{x}$$

so  $g(x)$  can be taken to be  $2\sqrt{x}$ , and its domain can be taken to be  $(0, 1]$ .

Let  $\varepsilon$  be an error bound that, if achieved for an integral approximation, permits the display of the integral's value to the requested number of decimal places. The program `impint` computes  $g(\alpha_n)$ , where  $\alpha_n = a + 2^{-n}(b_1 - a)$ , for the cases  $n = 0, 1, 2, \dots$ , until finally it obtains a  $g$  value less than  $\varepsilon/2$ . If this occurs for the argument  $\alpha_N$ , then  $\alpha_N$  becomes  $\alpha$ , and `impint` computes the definite integral  $\int_\alpha^b f(x) dx$  to the error bound  $\varepsilon/2$  by the same method used by the program `integ`.

The program `impint` computes the improper integral  $\int_{-\infty}^b f(x) dx$  in much the same way. The user is required to specify a positive function  $g(x)$  defined in some infinite interval  $(-\infty, b_1]$ , such that for any  $x$  in this domain the inequality  $|\int_{-\infty}^x f(x) dx| \leq g(x)$  holds, with the limit  $\lim_{x \rightarrow -\infty} g(x) = 0$  being valid. By repeated trials, the program `impint` finds a point  $\alpha$  within  $(-\infty, b_1]$  such that  $g(\alpha) < \varepsilon/2$ , and then computes  $\int_\alpha^b f(x) dx$  to the error bound  $\varepsilon/2$ .

As an example, for the improper integral  $\int_{-\infty}^2 e^{-x^2} dx$ , the function  $g(x)$  can be taken as  $\frac{1}{2}e^{-x^2}$  with domain  $(-\infty, -1]$ , because for any  $x$  in this domain we have

$$\int_{-\infty}^x e^{-t^2} dt < \int_{-\infty}^x -te^{-t^2} dt = \frac{1}{2}e^{-t^2} \Big|_{-\infty}^x = \frac{1}{2}e^{-x^2} = g(x)$$

An integral  $\int_a^b f(x) dx$  with  $f(x)$  defined only in  $[a, b)$  is handled similarly. For this integral a positive function  $h(x)$  is required, defined in some interval  $[a_1, b)$ , such that for any  $x$  in this domain the inequality  $|\int_x^b f(t) dt| < h(x)$  holds, with the limit  $\lim_{x \rightarrow a^+} h(x) = 0$  being valid. The program `impint` computes the  $f(x)$  integral from  $a$  to a point  $\beta$ , slightly to the left of  $b$ , and uses the  $h(x)$  function to account for the error made in omitting the small stretch  $[\beta, b]$ . An integral  $\int_a^\infty f(x) dx$  also requires a function  $h(x)$ , defined appropriately.

An integral  $\int_a^b f(x) dx$  with  $f(x)$  defined only in  $(a, b)$  can be integrated accurately if both a function  $g(x)$  and a function  $h(x)$  are supplied, to account for unintegrated stretches near  $a$  and near  $b$ . The integral  $\int_{-\infty}^\infty f(x) dx$  also requires that two error bounding functions be supplied.

## Software Exercises D

These exercises use the three integration programs `integ`, `mulint`, and `impint`.

1. Call up `integ` and calculate to 20 decimal places  $\int_{-1}^1 \sqrt{1-x^2} dx$ . An integral is computed by summing integral approximations over small subintervals, starting at the left interval endpoint and working toward the right endpoint. Note that progress over the interval  $[-1, +1]$  is slow near the endpoints of the interval, where the derivative of the integrand gets large. At  $x = -1$  and  $x = 1$ , the integrand derivative is not defined, and a degree 0 evaluation of the integrand is needed over small subintervals containing these points.

2. Vito Lampret [4] lists the following five integrals as examples of definite integrals that cannot be integrated analytically, because antiderivatives functions for the integrands are not known.

$$I_1 = \int_0^1 \sqrt{1+x^4} dx$$

$$I_2 = 8 \int_0^{\pi/2} \sqrt{1 - \frac{3}{4} \cos^2 x} dx$$

$$I_3 = \frac{1}{\pi} \int_0^\pi \cos(\sin x) dx$$

$$I_4 = \int_0^1 \frac{\arctan x}{x} dx$$

$$I_5 = \int_0^1 e^{x^2} dx$$

Call up `integ` and evaluate any of the four integrals  $I_1$ ,  $I_2$ ,  $I_3$ , or  $I_5$  to 10 decimal places. The remaining integral  $I_4$  is improper, and is treated in exercise 5.

3. Call up `mulint` and evaluate to 5 decimal places the integral  $\int_1^2 dx \int_1^x x^2 e^{xy} dy$ . Try also to compute to 5 decimal places the integral  $\int_0^1 dx \int_0^x \sqrt{1-y^2} dy$ . This integral computation is rejected by `mulint` because of difficulty obtaining a series expansion of the integrand near the  $(x, y)$  point  $(1, 1)$ . Note that the integrand is not analytic at the point  $(1, 1)$ .

4. Call up `impint` and calculate to 5 decimal places the improper integral  $\int_0^{\frac{\pi}{2}} \frac{\sin x}{x} dx$ . The function  $g(x)$  needed here can be taken as  $x$ , because  $|\frac{\sin x}{x}| < 1$  for  $x$  in  $(0, \frac{\pi}{2}]$ . Record the computed value, and then redo the calculation with  $g(x)$  taken incorrectly as  $x/1000$ . Notice that the answer changes slightly from its previous correct value. Accurate values for improper integrals are of course not possible if the error limiting functions  $g(x)$  and  $h(x)$  are incorrect.

5. Call up `impint` to evaluate to 10 decimal places the integral  $I_4$  of exercise 2. Because  $d/dx \arctan x = \frac{1}{1+x^2} < 1 = \frac{d}{dx} x$  for  $x$  in  $(0, 1]$ , the function  $\arctan x$  does not increase as fast as the function  $x$  increases in the interval  $(0, 1]$ . Therefore  $\frac{\arctan x}{x} < 1$  for  $x$  in  $(0, 1]$ , so  $\int_0^x \frac{\arctan x}{x} dx < x$  for  $x$  in  $(0, 1]$ . Accordingly, the needed `impint` function  $g(x)$  can be taken as  $x$ .

## Notes and References

- A. The book by Davis and Rabinowitz [3] gives a comprehensive survey of other integration techniques.
- B. The papers by Corliss and Krenz [1] and Corliss and Rall [2] discuss alternate approaches to the accurate computation of definite integrals.

- [1] Corliss, G. and Krenz, G., Indefinite integration with validation, *ACM Trans. Math. Software* **15** (1989), 375–393.
- [2] Corliss, G. and Rall, L. B., Adaptive, self-validating numerical quadrature, *SIAM J. Sci. Stat. Comput.* **8** (1987), 831–847.
- [3] Davis, P. J. and Rabinowitz, P., *Methods of Numerical Integration*, 2nd Edn, Academic Press, New York, 1984.
- [4] Lampret, V., An invitation to Hermite's Integration and Summation: A comparison between Hermite's and Simpson's Rules, *SIAM Rev.* **46** (2004), 311–328.

# Finding Where a Function $f(x)$ is Zero



This chapter treats the problem of finding the points where a given elementary function is zero. The demo program `zeros` solves problems of this kind.

## 7.1 Obtaining a solvable problem

For a polynomial  $P(x)$ , any real or complex argument  $x_0$  such that  $P(x_0) = 0$  is called a *root* of the polynomial. For a general real-valued function  $f(x)$ , it is customary to use other terminology. An argument  $x_0$  such that  $f(x_0) = 0$  is called a *zero* of the function. Let us assume that some real elementary function  $f(x)$  is defined in an interval  $[a, b]$ , and that all zeros inside this interval are to be found. A zero  $x_0$  of  $f(x)$  at which the derivative  $f'(x_0)$  is nonzero is called a *simple zero*. If  $f'(x_0) = 0$ , the zero is *multiple*. A third possibility is that  $f'(x_0)$  is not defined.

In general we would want a solution program to report that there are no zeros in  $[a, b]$  when this is the case, and if there are zeros, to list them to a prescribed number of correct decimal places, and to indicate which zeros are simple. We expect a program to accomplish this by examination of  $f(x)$ , and sometimes the derivative  $f'(x)$ , at various arguments in  $[a, b]$ . The data that must be specified appears to be just the elementary function  $f(x)$ , the two numbers  $a$  and  $b$  defining the search interval, and the number  $k$  of correct decimal places wanted.

However, suppose for the search interval  $[a, b]$  the supplied function is

$$f_c(x) = x - c$$

where the parameter  $c$  is some arbitrary constant. A difficulty here is that if we find  $f_c(a) \doteq 0$  or  $f_c(b) \doteq 0$ , we are uncertain whether this means a zero

lies in  $[a, b]$ . Perhaps the zero lies just outside the interval  $[a, b]$ . Our solution program must report a zero if  $c \geq a$  and  $c \leq b$ , and no zero otherwise; but determining whether these inequalities are true is a nonsolvable problem. We eliminate this difficulty by requiring, as part of the problem specification, that the function values  $f(a)$  and  $f(b)$  both test unequal to zero.

Another difficulty is illustrated by the function

$$f_d(x) = (x - 1)^2 + |d|$$

where  $d$  is an arbitrary real constant. Assume the interval of interest is  $[0, 2]$ . Here we see that if  $d \neq 0$ , no zero must be reported; whereas if  $d = 0$ , the zero 1 must be displayed to  $k$  decimal places. This amounts to attempting the nonsolvable problem of deciding whether or not the real number  $d$  is zero. To obtain a solvable problem we must allow some sort of "escape" from complete accuracy in identifying zeros or reporting the absence of zeros.

Note that at both endpoints of the interval  $[0, 2]$ , the function  $f_d(x)$  is positive. Even if we had a function  $f(x)$  that showed opposite signs at the interval endpoints, it would sometimes be difficult to find just one zero accurately.

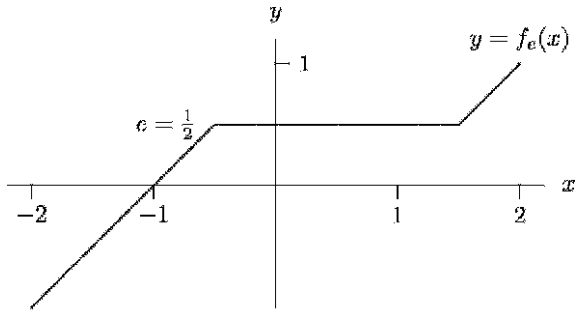
Consider the case of the function

$$f_e(x) = \max(\min(e, x + 1), x - 1)$$

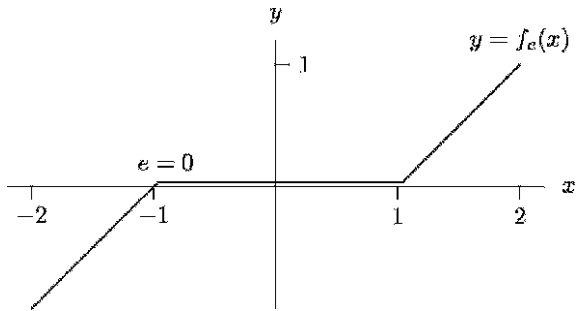
The parameter  $e$  can be any real number. For any setting of the parameter  $e$ , the function  $f_e(x)$  is defined for all  $x$ . This function has an interval of width 2 in which it equals  $e$ , and is linear with slope 1 everywhere else. The function  $f_e(x)$  is diagrammed for two  $e$  choices in Figs 7.1a and 7.1b. Suppose the search interval is  $[-2, 2]$ . For any  $e$ , we have  $f_e(-2)$  negative and  $f_e(2)$  positive. If  $e > 0$ , there is a single zero at  $x = -1$ . If  $e < 0$ , there is a single zero at  $x = 1$ . And if  $e = 0$ , all  $x$  points between  $-1$  and  $1$  are zeros. The difficulty with this function is that when  $e$  is close to zero, it becomes hard to decide whether there is a single zero at  $-1$  or at  $+1$ , or an infinite number of zeros between  $-1$  and  $+1$ .

We show next that, if for elementary functions  $f(x)$  defined on  $[a, b]$  with  $f(a)$  and  $f(b)$  of opposite signs, we could always determine just one zero to  $k$  decimal places, then we contradict Nonsolvable Problem 3.13. Given any real number  $e$ , we form the function  $f_e(x)$  and determine a zero to just 1 decimal place. If the obtained zero approximation  $\doteq 1$ , then  $e \leq 0$ , and if the approximation  $\doteq -1$ , then  $e \geq 0$ . And if the obtained approximation satisfies neither relation, then clearly  $e$  is zero, so we may choose either  $e \geq 0$  or  $e \leq 0$ . This contradiction of Problem 3.13 implies the interesting next result:

**Nonsolvable Problem 7.1** For any elementary function  $f(x)$  defined on  $[a, b]$ , with  $f(a)$  and  $f(b)$  both nonzero and having opposite signs, find to  $k$  decimal places one zero of  $f(x)$  in  $[a, b]$ .



**Fig. 7.1a** The function  $f_e(x)$  for  $e = \frac{1}{2}$ .



**Fig. 7.1b** The function  $f_e(x)$  for  $e = 0$ .

The problem of finding zeros of elementary functions becomes difficult if functions having an infinite number of zeros in the search interval  $[a, b]$  are permitted. We will require that, at most, a finite number of zeros are present in the search region, and this eliminates the possibility of  $f(x)$  being zero on a subinterval of  $[a, b]$ . Now that we have explored the difficulties in treating our problem, we can propose

**Solvable Problem 7.2** For any elementary function  $f(x)$  defined in an interval  $[a, b]$  and having at most a finite number of zeros there, and for any positive integer  $k$ , determine that  $|f(a)| < 10^{-k}$  or  $|f(b)| < 10^{-k}$  and halt. Or, if  $f(a) \neq 0$  and  $f(b) \neq 0$ , bound all the zeros of  $f(x)$  in  $[a, b]$  by:

- (1) giving, to  $k$  decimal places, points identified as simple zeros, or
- (2) giving, to  $k$  decimal places, points identified as containing within their tilde interval a subinterval where  $|f(x)| < 10^{-k}$ . The subinterval is identified as “containing at least one zero” if the  $f$  signs differ at the subinterval endpoints.

The “tilde interval” of a displayed decimal value, such as  $1.234\sim$ , is  $1.234 \pm \frac{1}{2}$ , that is, it is the interval obtained by appending  $\pm \frac{1}{2}$  to the displayed value. A case (2) answer not specified as containing at least one zero, is of course just a possible zero, and may turn out not to be a zero after all. With this problem, the results obtained depend on the size of  $k$ . It is possible that when  $k$  is increased, the number of case (2) points decreases, and the number of simple zeros increases.

The function  $f_e(x)$  can still be designated as an elementary function for this problem as long as the parameter constant  $e$  is not zero. (If  $f_e(x)$ , with  $e$  taken to be 0, is specified to the program `zeros` as the  $f(x)$  function, the program enters a loop, and must be stopped with a **control-c** input, because the requirement that only a finite number of zeros be present is violated.)

## 7.2 Using interval arithmetic for the problem

The search for  $f(x)$  zeros in  $[a, b]$  is done by evaluating  $f(x)$  using formal interval arithmetic, as described in Chapter 6 for the problem of finding definite integrals.

A task queue is needed, with the queue listing the intervals to be tested. Initially the queue has a single entry specifying the entire search interval  $[a, b]$ . We want to bound zeros in small subintervals of halfwidth less than some control parameter  $W$ , which is set initially to a small value, such as 0.1. The task queue cycle is as follows. The preliminary  $x$  series is set to reflect the leading queue interval, and then the  $f(x)$  evaluation list is executed to obtain an interval  $m_0 \pm w_0$  for  $f(x)$ .

If the computed  $f(x)$  interval is positive or negative, there are no zeros in the designated interval, and the queue interval is discarded. If the  $f(x)$  interval overlaps 0, there may be one or more zeros in the designated interval, but we cannot be certain of this, because interval arithmetic does not necessarily yield the true interval. It only gives an interval guaranteed not to be too small. If the queue interval halfwidth  $< W$ , the queue interval is transferred to a list  $L$ , which initially is empty. Otherwise the queue interval is bisected and two bisection interval entries replace the previous entry on the queue. Eventually the queue becomes empty, and then we can examine the list  $L$  of small  $[a, b]$  subintervals, for which the computed  $f(x)$  interval overlapped zero. Of course if  $L$  is empty, there are no zeros in  $[a, b]$ , and we are done.

Of all the subintervals formed in the process of subdividing  $[a, b]$ , those containing zeros must appear on the list  $L$ . But a subinterval adjoining to one of these may also end up on  $L$ , because of  $f(x)$  interval overestimation. It is also possible that a zero happens to lie on a boundary point of a subinterval, and then the two subintervals sharing this boundary point both are on  $L$ . To simplify matters, we divide the  $L$  subintervals into sets of adjoining subintervals, create for each set a container interval  $[c, d]$  equal to the union of the subintervals of

its members. Container intervals are held in a second list  $L_1$ , and each container interval has attached to it a list of its set members.

A list  $L_1$  container interval  $[c, d]$  has the property that  $f(c)$  is nonzero. If  $c$  equals  $a$ , the left endpoint of the starting search interval  $[a, b]$ , this is certainly true. If  $c$  is not  $a$ , then  $c$ , besides being a left endpoint of some subinterval that ended up on the list  $L$ , is also a right endpoint of a different subinterval that was discarded. A subinterval is discarded only if its computed  $f(x)$  interval is positive or negative, so  $f(c)$  is nonzero. Similarly,  $f(d)$  is nonzero.

Depending on the function  $f(x)$ , for a container interval  $[c, d]$  it may be possible to compute an interval  $m_1 \pm w_1$  for  $f'(x)$  over  $[c, d]$ . A favorable circumstance would be finding that the  $f'(x)$  interval was either positive or negative. In this case, because the function  $f(x)$  is either strictly increasing or strictly decreasing over  $[c, d]$ , there can be at most one simple zero in the container interval. Usually this zero can be found rapidly by Newton's method, discussed in the next two sections.

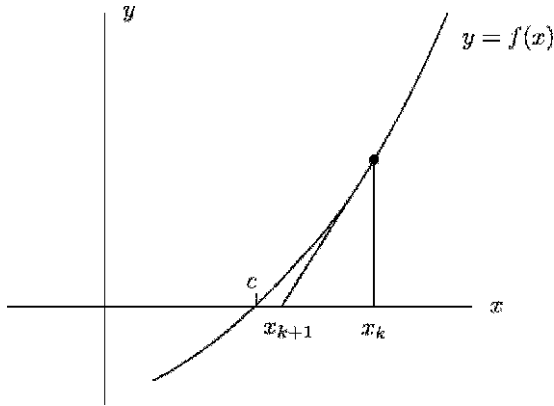
For some of the  $L_1$  container intervals, it may not be possible to compute an  $f'(x)$  interval, or if it is computed, it may fail to test positive or negative. For each such case, we move the list of member subintervals associated with the container interval back onto the task queue, and discard the container interval. We do the same for container intervals for which the Newton method attempt fails. The target halfwidth  $W$  is reduced by multiplying it by some factor, such as 0.1, and then the previously described queue procedure of  $f(x)$  interval computation and queue interval bisection is repeated, leading to a set of smaller subintervals possibly containing zeros, another list of container intervals on  $L_1$ , a repetition of  $f'(x)$  computation, and if we are not done, additional cycles of this computation.

In favorable cases of this problem, all the zeros end up being identified as simple zeros and being computed to the required number of places by Newton's method. Otherwise, the cyclic processing of each container interval  $[c, d]$  continues until the interval becomes so small that it lies within the tilde-interval of a  $k$  decimal display of its midpoint, and, simultaneously, a computed interval for  $f(x)$  over  $[c, d]$  indicates that  $|f(x)| < 10^{-k}$  for  $x$  in  $[c, d]$ . In this case the midpoint of this small interval is displayed to  $k$  decimal places along with an upper bound on  $|f(x)|$ . The presence of at least one zero in  $[c, d]$  can be reported whenever  $f(c)$  and  $f(d)$  are of opposite signs.

## 7.3 Newton's method

Suppose  $f(x)$  is a function having a derivative, and that  $x_0$  is a zero approximation for  $f(x)$ , perhaps grossly inaccurate. Newton's method is an iteration system for progressively improving a zero approximation for any function with a derivative, with the method delivering a sequence of zero approximations:  $x_1, x_2, \dots, x_k, \dots$ . The iteration equation is derived from this simple idea: if we have an approximation  $x_k$  to a zero of  $f(x)$ , then on the graph of  $f(x)$





**Fig. 7.2** Newton's Method.

(see Fig. 7.2), we can draw a tangent line at the point  $x = x_k$ , and follow this tangent line to where it crosses the  $x$  axis. This way we obtain another zero approximation  $x_{k+1}$ , that is hopefully better than  $x_k$ . The Taylor series for  $f(x)$ , with the series expansion point at  $x_k$ , is

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \dots \quad (7.1)$$

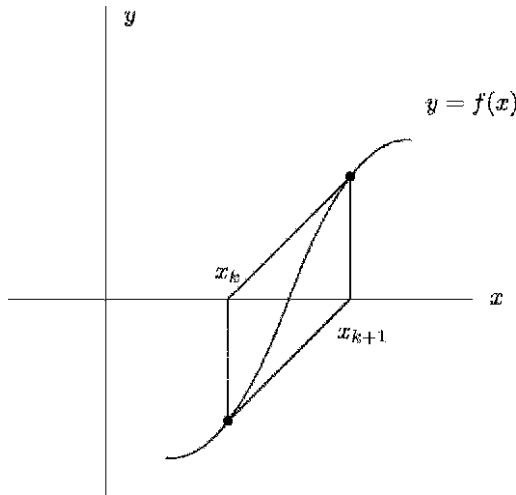
The equation of the tangent line at  $x = x_k$  is obtained by discarding all terms of the Taylor series beyond the ones shown, to get the straight line equation

$$y = f(x_k) + f'(x_k)(x - x_k)$$

If in this equation we set  $y$  equal to zero and solve for  $x$ , calling the solution  $x_{k+1}$ , then we obtain the Newton's method equation

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad (7.2)$$

The zero-finding procedure described in the preceding section bounds at most one simple zero in a container interval  $[c, d]$  if the  $f'(x)$  interval computed for  $[c, d]$  is positive or negative. Using the Newton's method iteration equation, and with the starting approximation  $x_0$  taken as the midpoint of  $[c, d]$ , the sequence of iterates generally locates the zero rapidly. However, as Fig. 7.3 shows, it is possible that the iterates cycle without approaching the zero. If this should occur, we can treat the container interval as if its  $f'(x)$  interval was not of one sign, that is, we discard  $[c, d]$  and process its associated subintervals further. Perhaps a smaller container interval will lead to success with Newton's method. In the more common case where the iterates approach the zero, we need a criterion



**Fig. 7.3** Cycling with Newton's method.

for halting the iteration cycle. After an iterate  $x_{k+1}$  is computed from  $x_k$ , the two iterates are compared. If  $x_{k+1} \neq x_k$ , then the range of  $x_{k+1}$  is set to zero in preparation for the next cycle, that is,  $x_{k+1}$  is replaced by the exact midpoint of  $x_k$ . Eventually we find  $x_{k+1} \doteq x_k$ , and then the exact midpoint of  $x_k$  becomes the final approximation to the zero in  $[c, d]$ , and we designate this final value  $x_z$ . The case of cycling iterates is detected by finding that  $|x_{k+1} - x_k|$  does not decrease as  $k$  increases.

Next we need to determine the error of  $x_z$ , which is the distance from  $x_z$  to the zero. For the  $x_z$  container interval  $[c, d]$ , an interval  $f'(x_0) \pm w_1$  was computed for the  $f$  derivative. Suppose we compute the interval  $m \pm w$  equal to  $[f(x_z) \pm 0] / [f'(x_0) \pm w_1]$ . Then, as we show, the error of  $x_z$  is bounded by  $e = |m| + w$ , so  $x_z \oplus e$  may be taken as a correctly ranged zero approximation. This result follows from the Mean Value Theorem. At the true zero  $z$ , we have  $f(z) = 0$ , so  $f(x_z) = f(x_z) - f(z) = f'(c_x)(x_z - z)$ , where the point  $c_x$  is between  $x_z$  and  $z$ . The error  $|x_z - z|$  equals  $|f(x_z)| / |f'(c_x)|$ , and because  $f'(c_x)$  lies in the interval  $f'(x_0) \pm w_1$ , the error is bounded by  $e$ . If  $x_z \oplus e$  does not give us the desired number of correct decimal places, we need to increase the precision of computation, restart the Newton's method iteration process, and repeat the error test with a better  $x_z$ .

## 7.4 Order of convergence

In the later cycles of Newton's method, when  $x_k$  gets close to a zero  $z$ , it is often noticed that the convergence accelerates, with the number of correct decimal

places of the iterates approximately doubling at each step. We can show this by using the Taylor series remainder formula, adding an additional remainder term to the terms shown in equation (7.1):

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(c_x)(x - x_k)^2$$

Here  $c_x$  is some point between  $x$  and  $x_k$ . If we set  $x$  equal to the zero  $z$ , then we have

$$0 = f(x_k) + f'(x_k)(z - x_k) + \frac{1}{2}f''(c_z)(z - x_k)^2$$

When we divide this equation by  $f'(x_k)$  and set  $x_k - f(x_k)/f'(x_k)$  equal to  $x_{k+1}$ , we obtain

$$0 = z - x_{k+1} + \frac{f''(c_z)}{2f'(x_k)}(z - x_k)^2$$

which can be rewritten as

$$z - x_{k+1} = -\frac{f''(c_z)}{2f'(x_k)}(z - x_k)^2 \quad (7.3)$$

Because the term in  $z - x_k$  is squared, if  $f'(z) \neq 0$  and  $x_k$  is close enough to  $z$ , then  $x_{k+1}$  is even closer to  $z$ , so the iterates after  $x_k$  converge to  $z$ .

The quantity  $|z - x_k|$  is the error of the approximation  $x_k$ . Taking absolute values of both sides of equation (7.3), we obtain

$$(x_{k+1} \text{ error}) = L_k(x_k \text{ error})^2$$

where  $L_k = |f''(c_z)/2f'(x_k)|$ . The coefficient  $L_k$  approaches a limiting value  $L$  because

$$\lim_{k \rightarrow \infty} L_k = \lim_{x_k \rightarrow z} \frac{|f''(c_z)|}{2|f'(x_k)|} = \frac{|f''(z)|}{2|f'(z)|} = L$$

Thus in the last stages of Newton's method, the following equation is accurate:

$$(x_{k+1} \text{ error}) = L(x_k \text{ error})^2$$

This equation implies that the number of correct decimal places approximately doubles at each step, and may be more than this or less, depending on the magnitude of  $L$ .

When a series of approximates  $x_k$  approaches the true value  $z$ , such that the error satisfies the relation

$$\lim_{k \rightarrow \infty} \frac{\text{error } x_{k+1}}{(\text{error } x_k)^s} = L > 0$$

the convergence is said to be *of order s*. Thus the convergence of the iterates  $x_k$  of Newton's method is generally of order two, or  $x_k$  converges *quadratically*. If the exponent  $s$  equals one, then  $x_k$  is said to converge *linearly*. For linear convergence, the positive constant  $L$  in the limit must be less than 1, and the nearer  $L$  is to 1, the slower the convergence.

Linear convergence is encountered with Newton's method if the derivative at  $z$  is zero. Suppose we have  $f'(z) = f''(z) = \dots = f^{(n-1)}(z) = 0$ , and  $f^{(n)}(z) \neq 0$ . If we expand  $f(x)$  in a Taylor series about the zero point  $z$ , and use the Taylor series remainder formula, at  $x = x_k$  we have the equation

$$\begin{aligned} f(x_k) &= f(z) + f'(z)(x_k - z) + \frac{1}{2!}f''(z)(x_k - z)^2 + \dots + \frac{1}{n!}f^{(n)}(c_1)(x_k - z)^n \\ &= \frac{1}{n!}f^{(n)}(c_1)(x_k - z)^n \end{aligned}$$

Here  $c_1$  is some point between  $x_k$  and  $z$ . Similarly, for  $f'(x_k)$  we have the equation

$$f'(x_k) = \frac{1}{(n-1)!}f^{(n)}(c_2)(x_k - z)^{n-1}$$

where  $c_2$  is some point between  $x_k$  and  $z$ . We have then

$$x_{k+1} - z = x_k - \frac{f(x_k)}{f'(x_k)} - z = (x_k - z) - \frac{f^{(n)}(c_1)}{nf^{(n)}(c_2)}(x_k - z)$$

If  $x_k$  is close enough to  $z$ , then  $f^{(n)}(c_1)/nf^{(n)}(c_2)$  is near to  $\frac{1}{n}$ , and convergence is certain because  $|x_{k+1} - z| < |x_k - z|$ . Then  $x_k$  approaches  $z$  in the limit, and we have

$$\lim_{k \rightarrow \infty} \frac{x_{k+1} \text{ error}}{x_k \text{ error}} = \lim_{x_k \rightarrow z} \left| 1 - \frac{f^{(n)}(c_1)}{nf^{(n)}(c_2)} \right| = \left| 1 - \frac{f^{(n)}(z)}{nf^{(n)}(z)} \right| = \frac{n-1}{n}$$

The convergence is linear, and is slower for larger  $n$ .

## Software Exercises E

These exercises demonstrate some properties of the demo program `zeros`.

1. Call up the program `zeros`. The program requests the number  $n$  of functions for which zeros are to be found. In Chapter 12, solution methods are described for finding where several functions are simultaneously zero. For now, however, take  $n$  equal to 1. Enter the function  $\sqrt[3]{x}$  by using the keyboard line `x^(1/3)`,

take the search interval to be  $[-5, 5]$ , and choose 5 decimal-place precision. The derivative of this function is undefined at  $x = 0$ , and the zero at  $x = 0$  is found not by Newton's method, but by the slower process described in the text. Note that the program makes a positive report of a zero.

2. Now that a log file is available, edit the log file to change the function's exponent from  $1/3$  to  $2/3$ , save the log file, and then call up `zeros zeros`. This time a positive report of a zero is not made, because the function does not change signs at the endpoints of the container interval.

3. Call up `zeros` and find to 10 decimal places the zeros of  $x^2 + 10^{-100}$  in  $[-1, 1]$ . Here there is no zero, but the program locates a zero possibility. Edit the log file to increase the number of decimal places requested to 50, and then call up `zeros zeros`. This time the higher precision computation leads to no zero being found.

4. Call up `zeros` and find to 5 decimal places the zeros of  $\frac{1}{x} \cdot \sin \frac{1}{x}$  in  $[0.1, 1]$ . Here there are several zeros, and the order in which they are found is determined by chance. Edit the log file to move the left endpoint of the search interval closer to the origin, to 0.01. When you call up `zeros zeros`, many more zeros are reported. To see all the zeros, use the **PNM** form to view the print file.

# Finding Roots of Polynomials



The demo program `roots` finds the roots of polynomials with real coefficients. The next five sections give needed background information for the problem of locating roots, and the last section describes the `roots` program.

## 8.1 Polynomials

Let  $n$  be any nonnegative integer. A *polynomial of degree  $n$  in the variable  $z$*  has the form

$$P(z) = c_n z^n + c_{n-1} z^{n-1} + \cdots + c_1 z + c_0 \quad (8.1)$$

where the leading coefficient  $c_n$  is required to be nonzero. (It is convenient in this section to allow a polynomial's degree to be 0, and accordingly any nonzero constant is viewed as defining a polynomial of degree 0.) A polynomial is a *real polynomial*, a *rational polynomial*, or a *complex polynomial* depending on whether all coefficients  $c_i$  are, respectively, real, rational, or complex numbers. If  $n$  is positive, the polynomial also has the representation

$$P(z) = c_n (z - z_1)(z - z_2) \cdots (z - z_n) \quad (8.2)$$

where the  $n$  factors  $z - z_i$  are unique, apart from order. Each complex number  $z_i$  is a *root* of the polynomial. Going from the representation (8.1) to the representation (8.2) is not a simple problem, but at least it is one that is solvable.

**Solvable Problem 8.1** For any real or complex polynomial of positive degree  $n$ , find the  $n$  roots to  $k$  decimal places.

We find a complex number to  $k$  decimal places if we find both its real part and its imaginary part to  $k$  places. An efficient method of locating the roots of a real polynomial is described later, in Section 8.3. We defer to Chapter 16 the description of a practical method for locating the roots of a complex polynomial.

Some of the roots of a polynomial may be identical. In equation (8.2), if we group together the linear factors of identical roots and reassign indices so that the distinct roots are  $z_1, z_2, \dots, z_q$ , we obtain the representation

$$P(z) = c_n(z - z_1)^{n_1}(z - z_2)^{n_2} \dots (z - z_q)^{n_q} \quad (8.3)$$

Here the integers  $n_1, n_2, \dots, n_q$  sum to  $n$ . The exponent  $n_i$  is the *multiplicity* of the root  $z_i$ . If  $n_i = 1$ ,  $z_i$  is a *simple* root, otherwise  $z_i$  is a *multiple* root with *multiplicity*  $n_i$ . It is not a solvable problem to find the multiplicity of the roots of a real or complex polynomial of degree more than 1, as is easily seen by considering the polynomial

$$P(z) = z^2 - (a_1 + a_2)z + a_1a_2 = (z - a_1)(z - a_2)$$

where  $a_1$  and  $a_2$  are any real numbers. If we can always determine the multiplicity of this polynomial's roots, then we have a method of determining whether any two numbers  $a_1$  and  $a_2$  are equal, contradicting Nonsolvable Problem 3.3.

**Nonsolvable Problem 8.2** For any real or complex polynomial of degree  $n > 1$ , determine the multiplicity of the roots.

Suppose for a real or complex polynomial  $P(z)$  we obtain correctly ranged approximations to all its roots, using the various methods described in the text. We can arrange the roots in sets, with two roots  $z_1 = x_1 + iy_1$  and  $z_2 = x_2 + iy_2$  belonging to the same set if we find  $x_1 \doteq x_2$  and  $y_1 \doteq y_2$ . A root that is in a set all by itself is certainly a simple root, but suppose we find  $m$  roots collected in one set. We then have a root of *apparent* multiplicity  $m$ . If we were to compute polynomial root approximations to more correct decimal places, the roots may arrange themselves differently, so that a root of apparent multiplicity  $m$  becomes a root of smaller apparent multiplicity, or even a simple root. Thus an apparent multiplicity may be larger than the true multiplicity, and only an apparent multiplicity of 1 is certain to be the correct multiplicity.

For rational polynomials this difficulty in determining multiplicity disappears.

**Solvable Problem 8.3** For any rational polynomial, find its distinct roots to  $k$  decimal places, and determine their multiplicity.

In this section, we show that a rational polynomial  $P(z)$  of positive degree can be decomposed into a set of polynomials  $N_1(z), N_2(z), \dots, N_r(z)$ , containing

the distinct roots of  $P(z)$ , such that the  $P(z)$  roots of multiplicity  $m$  are simple roots of  $N_m(z)$ . The integer  $r$  equals the highest multiplicity of any root of  $P(z)$ . If there are no multiple roots, then  $r = 1$  and  $N_1(z) = P(z)$ . The task of finding the roots of  $P(z)$  is made easier by this decomposition, and the correct multiplicity of the computed roots is obtained as a by-product.

A polynomial with a leading coefficient of 1 is called a *monic* polynomial. Any polynomial can be converted to monic form by dividing it by its leading coefficient  $c_n$ . If this is done for the polynomial (8.1), the polynomial becomes

$$P_1(z) = z^n + a_{n-1}z^{n-1} + \cdots + a_1z + a_0$$

where the coefficient  $a_i$  equals  $c_i/c_n$ . The representation for this polynomial in terms of its distinct roots is now

$$P_1(z) = (z - z_1)^{n_1} (z - z_2)^{n_2} \cdots (z - z_q)^{n_q} \quad (8.4)$$

If  $z_i$  is a root of multiplicity  $n_i$  for  $P_1(z)$ , then  $z_i$  will be a root of multiplicity  $n_i - 1$  for the derivative of  $P_1(z)$ . This is easy to show. We have

$$P_1(z) = (z - z_i)^{n_i} S(z)$$

where  $S(z)$  is some polynomial of lower degree not having  $z_i$  as a root, that is,  $S(z_i) \neq 0$ . Taking the derivative, we have

$$\begin{aligned} P_1'(z) &= n_i(z - z_i)^{n_i-1} S(z) + (z - z_i)^{n_i} S'(z) = (z - z_i)^{n_i-1} [n_i S(z) + (z - z_i) S'(z)] \\ &= (z - z_i)^{n_i-1} T(z) \end{aligned}$$

where the polynomial  $T(z)$  equals  $n_i S(z) + (z - z_i) S'(z)$ . We see that for  $P_1'(z)$  the multiplicity of  $z_i$  is at least  $n_i - 1$ , and it cannot be higher because  $T(z_i) = n_i S(z_i) \neq 0$ . Thus if the multiplicity structure of  $P_1(z)$  is as shown in equation (8.4), then  $P_1(z)$  and  $P_1'(z)$  have the common divisor polynomial

$$P_2(z) = (z - z_1)^{n_1-1} (z - z_2)^{n_2-1} \cdots (z - z_q)^{n_q-1} \quad (8.5)$$

Any two polynomials have a unique monic common divisor polynomial of highest degree, called their greatest common divisor, as is clear by considering the linear factor forms (8.3) of the two polynomials. For  $P_1(z)$  and  $P_1'(z)$ , there cannot be a higher degree divisor polynomial than  $P_2(z)$ , so  $P_2(z)$  is their greatest common divisor. When  $P_1(z)$  has rational coefficients, the greatest common divisor of  $P_1(z)$  and  $P_1'(z)$  can be found, as we now show.

Here we use a version of the Euclidean algorithm presented in Section 2.8 as a means of finding the greatest common divisor of two positive integers. The polynomial  $P_1(z)$  is divided by  $P_1'(z)$ , using the usual polynomial division procedure.



We obtain the quotient polynomial  $Q_1(z)$  and the remainder polynomial  $R_1(z)$ , as shown in the next equation:

$$P_1(z) = Q_1(z)P_1'(z) + R_1(z)$$

Any common divisor polynomial of  $P_1(z)$  and  $P_1'(z)$  is also a common divisor of  $R_1(z)$ . Similarly, any common divisor polynomial of  $P_1'(z)$  and  $R_1(z)$  is also a common divisor of  $P_1(z)$ . So the pair  $P_1(z)$  and  $P_1'(z)$  has the same greatest common divisor as the pair  $P_1'(z)$  and  $R_1(z)$ . The division process is repeated for the new pair, obtaining

$$P_1'(z) = Q_2(z)R_1(z) + R_2(z)$$

and this process is continued until a zero remainder is obtained:

$$\begin{aligned} R_1(z) &= Q_3(z)R_2(z) + R_3(z) \\ R_2(z) &= Q_4(z)R_3(z) + R_4(z) \\ &\vdots \qquad \qquad \qquad \vdots \qquad \qquad \qquad \vdots \\ R_{s-2}(z) &= Q_s(z)R_{s-1}(z) + R_s(z) \\ R_{s-1}(z) &= Q_{s+1}(z)R_s(z) \end{aligned}$$

The polynomial  $R_s(z)$ , after it is converted to monic form, will equal  $P_2(z)$  of equation (8.5). This is because  $R_s(z)$  is the greatest common divisor of the last pair, and proceeding backward step by step, also the greatest common divisor of the initial pair  $P_1(z)$  and  $P_1'(z)$ .

The procedure used on  $P_1(z)$  to obtain  $P_2(z)$  can now be applied to  $P_2(z)$  if this monic polynomial is not equal to 1, and then we get the polynomial  $P_3(z)$ . Thus the Euclidean procedure, if repeatedly applied, produces a series of monic polynomials  $P_1(z)$ ,  $P_2(z)$ ,  $P_3(z)$ ,  $\dots$ ,  $P_r(z)$ , 1, that always ends with a polynomial  $P_{r+1}(z)$  equal to 1.

When  $P_1(z)$  of line (8.4) is divided by the polynomial  $P_2(z)$  of line (8.5), we obtain the polynomial

$$M_1(z) = (z - z_1)(z - z_2) \dots (z - z_q)$$

with all the distinct roots of  $P_1(z)$  as simple roots. In general, if for the sequence  $P_1(z)$ ,  $P_2(z)$ ,  $\dots$ ,  $P_r(z)$ ,  $P_{r+1}(z) = 1$ , we define

$$M_i(z) = \frac{P_i(z)}{P_{i+1}(z)} \text{ for } i = 1, 2, \dots, r$$

we get the set of polynomials  $M_1(z)$ ,  $M_2(z)$ ,  $\dots$ ,  $M_r(z)$ , all with only simple roots, such that all roots of  $P_1(z)$  which are of multiplicity  $m$  or higher are

roots of  $M_m(z)$ . If  $M_i(z)$  and  $M_{i+1}(z)$  are two successive polynomials in the set, the degree of  $M_{i+1}(z)$  is less than or equal to the degree of  $M_i(z)$ . The last polynomial in this series,  $M_r(z)$ , has only the roots of  $P_1(z)$  with the highest multiplicity.

From these polynomials it is convenient to obtain by division another set of  $r$  polynomials  $N_1(z)$ ,  $N_2(z)$ ,  $\dots$ ,  $N_r(z)$ , again with simple roots only, defined by the equation

$$N_i(z) = \begin{cases} \frac{M_i(z)}{M_{i+1}(z)} & \text{if } i < r \\ M_r(z) & \text{if } i = r \end{cases}$$

The polynomials  $N_i(z)$  have degrees that sum to  $q$ , the number of distinct roots of  $P_1(z)$ , and each distinct root of  $P_1(z)$  is a root of just one of these polynomials. If the root is of multiplicity  $m$ , then it is a root of  $N_m(z)$ . Thus when the roots of the polynomials  $N_i(z)$  are found by the method described later, they can be displayed with their exact multiplicities.

As an example, suppose  $P_1(z) = z^7 - 3z^5 + 3z^3 - z$ . Then  $P_1'(z) = 7z^6 - 15z^4 + 9z^2 - 1$ . The Euclidean algorithm yields

$$\begin{aligned} z^7 - 3z^5 + 3z^3 - z &= \frac{1}{7}z(7z^6 - 15z^4 + 9z^2 - 1) - \frac{6}{7}z^5 + \frac{12}{7}z^3 - \frac{6}{7}z \\ 7z^6 - 15z^4 + 9z^2 - 1 &= -\frac{49}{6}z\left(-\frac{6}{7}z^5 + \frac{12}{7}z^3 - \frac{6}{7}z\right) - z^4 + 2z^2 - 1 \\ -\frac{6}{7}z^5 + \frac{12}{7}z^3 - \frac{6}{7}z &= \frac{6}{7}z\left(-z^4 + 2z^2 - 1\right) \end{aligned}$$

Making the last polynomial of the series monic by dividing by its leading coefficient, we get  $P_2(z) = z^4 - 2z^2 + 1$ . Repeating the Euclidean algorithm with  $P_2(z)$ , we obtain

$$\begin{aligned} z^4 - 2z^2 + 1 &= \frac{1}{4}z(4z^3 - 4z) - z^2 + 1 \\ 4z^3 - 4z &= -4z(-z^2 + 1) \end{aligned}$$

so  $P_3(z) = z^2 - 1$ . One more application of the Euclidean algorithm leads to

$$\begin{aligned} z^2 - 1 &= \frac{1}{2}z(2z) - 1 \\ 2z &= -2z(-1) \end{aligned}$$

so  $P_4(z)$  is 1.

The  $M_i(z)$  polynomials are

$$M_1(z) = \frac{P_1(z)}{P_2(z)} = z^3 - z$$

$$M_2(z) = \frac{P_2(z)}{P_3(z)} = z^2 - 1$$

$$M_3(z) = \frac{P_3(z)}{1} = z^2 - 1$$

and the  $N_i(z)$  polynomials are

$$N_1(z) = \frac{M_1(z)}{M_2(z)} = z$$

$$N_2(z) = \frac{M_2(z)}{M_3(z)} = 1$$

$$N_3(z) = M_3(z) = z^2 - 1$$

We see that the roots of  $P_1(z)$  are 0 with multiplicity 1, along with 1 and  $-1$ , both with multiplicity 3.

These results can also be understood by setting

$$P_1(z) = (z - z_1)^3(z - z_2)^3(z - z_3)$$

with  $z_1 = 1$ ,  $z_2 = -1$ , and  $z_3 = 0$ . We see that by repeated application of the Euclidean algorithm, we would obtain the sequence

$$P_2(z) = (z - z_1)^2(z - z_2)^2$$

$$P_3(z) = (z - z_1)(z - z_2)$$

$$P_4(z) = 1$$

leading to the sequence

$$M_1(z) = (z - z_1)(z - z_2)(z - z_3)$$

$$M_2(z) = (z - z_1)(z - z_2)$$

$$M_3(z) = (z - z_1)(z - z_2)$$

and then finally to the sequence

$$N_1(z) = (z - z_3)$$

$$N_2(z) = 1$$

$$N_3(z) = (z - z_1)(z - z_2)$$

## 8.2 A bound for the roots of a polynomial

When searching for the roots of a monic polynomial

$$P_1(z) = z^n + a_{n-1}z^{n-1} + \cdots + a_1z + a_0 \quad (8.6)$$

it is helpful to compute a *bounding radius*  $R$ , such that in the complex plane, the circle  $|z| \leq R$  contains all the roots. The next two theorems give two ways of computing a bounding radius.

**Theorem 8.1** A bounding radius for the roots of the polynomial  $P_1(z)$  is

$$R_1 = \max\{|a_{n-1}| + 1, |a_{n-2}| + 1, \dots, |a_1| + 1, |a_0|\}$$

As an example, let us form the polynomial

$$P_1(z) = (z-1)(z-2)(z-3)(z-4) = z^4 - 10z^3 + 35z^2 - 50z + 24$$

We obtain  $R_1 = \max\{11, 36, 51, 24\} = 51$ .

To prove the theorem, we show that if a complex number  $z$  is such that  $|z| > R_1$ , then

$$|a_{n-1}z^{n-1} + \cdots + a_1z + a_0| < |z^n| \quad (8.7)$$

This implies that  $z$  cannot be a root of  $P_1(z)$ , because roots satisfy the equation

$$a_{n-1}z^{n-1} + \cdots + a_1z + a_0 = -z^n$$

which leads to

$$|a_{n-1}z^{n-1} + \cdots + a_1z + a_0| = |z^n|$$

Proceeding with the proof, we assume now that  $|z| > R_1$ , and obtain

$$\begin{aligned} |a_{n-1}z^{n-1} + \cdots + a_1z + a_0| &\leq |a_{n-1}||z|^{n-1} + \cdots + |a_1||z| + |a_0| \\ &\leq (R_1 - 1)|z|^{n-1} + \cdots + (R_1 - 1)|z| + R_1 \\ &= (R_1 - 1)(|z|^{n-1} + \cdots + |z| + 1) + 1 \\ &= (R_1 - 1) \frac{|z|^n - 1}{|z| - 1} + 1 \\ &< (|z| - 1) \frac{|z|^n - 1}{|z| - 1} + 1 = |z|^n \end{aligned}$$

The next result gives a way of forming a bounding radius that often is more accurate than  $R_1$ .

**Theorem 8.2** If the monic polynomial  $P_1(z)$  of line (8.6) has at least one nonzero coefficient  $a_i$ , the polynomial

$$P_\rho(z) = z^n - |a_{n-1}|z^{n-1} - \cdots - |a_1|z - |a_0|$$

has exactly one positive root  $\rho$ , which may be taken as a bounding radius for  $P_1(z)$ .

For our example polynomial,  $z^4 - 10z^3 + 35z^2 - 50z + 24$ , the polynomial  $P_\rho(z)$  equals  $z^4 - 10z^3 - 35z^2 - 50z - 24$ , and its positive root is  $13.00\sim$ . So we get a smaller bounding radius than previously, though we need to find the positive root to obtain it. To eliminate the need of finding the positive root, we can just approximate the positive root of  $P_\rho(z)$  by choosing some convenient small positive integer  $r$ , and then forming  $P_\rho(r)$ ,  $P_\rho(r^2)$ ,  $P_\rho(r^3)$ ,  $\dots$ , stopping as soon as we have obtained  $P_\rho(r^k) > 0$ . The last  $r^k$  value formed exceeds the positive root  $\rho$ , so we can use this as our bounding radius. With our example, if we choose  $r = 2$ , we find  $P_\rho(2^3) < 0$  and  $P_\rho(2^4) > 0$ , so we get a bounding radius of 16 this way.

To prove the theorem, set  $z$  equal to the real number  $x$ , and write the polynomial  $P_\rho(x)$  as

$$P_\rho(x) = x^n \left( 1 - \frac{|a_{n-1}|}{x} - \cdots - \frac{|a_1|}{x^{n-1}} - \frac{|a_0|}{x^n} \right) \quad (8.8)$$

The factor  $x^n$  has no positive roots. The other factor, with a positive derivative for  $x > 0$ , approaches  $-\infty$  as  $x$  approaches zero from the right and approaches 1 as  $x$  approaches  $\infty$ , and thus has exactly one positive root  $\rho$ . If  $z$  is a complex number with  $|z| > \rho$ , then  $P_\rho(|z|)$  is positive. This implies

$$\begin{aligned} |z|^n - |a_{n-1}||z|^{n-1} - \cdots - |a_1||z| - |a_0| &> 0 \\ |z|^n > |a_{n-1}||z|^{n-1} + \cdots + |a_1||z| + |a_0| &\geq |a_{n-1}z^{n-1} + \cdots + a_1z + a_0| \end{aligned}$$

Again we have inequality (8.7), so  $z$  cannot be a root of  $P_1(z)$ .

### 8.3 The Bairstow method for finding roots of a real polynomial

There are many methods for finding approximations to the roots of a polynomial, and the Bairstow method is an efficient method for real polynomials.

A real polynomial  $P(z)$  of even degree can be written as a product of real quadratic (degree 2) polynomials. This follows from the polynomial's factorization (8.2), because the polynomial's complex roots come in conjugate pairs, and a conjugate pair of roots,  $x_1 + iy_1$  and  $x_1 - iy_1$ , yields the real quadratic factor

$$(z - x_1 - iy_1)(z - x_1 + iy_1) = [(z - x_1) - iy_1][(z - x_1) + iy_1] = (z - x_1)^2 - (iy_1)^2 \\ = z^2 - 2x_1z + x_1^2 + y_1^2$$

The roots that are not conjugate pairs then are real roots, and these can be grouped by pairs  $r_1$  and  $r_2$ , with each pair again yielding a real quadratic factor:

$$(z - r_1)(z - r_2) = z^2 - (r_1 + r_2)z + r_1r_2$$

A real polynomial of odd degree can be written as a product of real quadratic polynomials and one real linear (degree 1) polynomial.

The Bairstow method for real polynomials finds quadratic real factors, and in this way finds roots, a pair at a time. Suppose  $B(z) = z^2 + b_1z + b_0$  is an approximation to some quadratic factor of the real polynomial  $P(z)$  of degree  $n$ . When we divide  $P(z)$  by  $B(z)$ , we obtain the quotient  $Q(z)$  and a linear remainder  $c_1z + c_0$ :

$$P(z) = B(z)Q(z) + c_1z + c_0 \quad (8.9)$$

The coefficients  $c_1$  and  $c_0$  are functions of  $b_1$  and  $b_0$ , and we want to choose the variables  $b_1$  and  $b_0$  so both  $c_1(b_1, b_0)$  and  $c_0(b_1, b_0)$  are zero. We obtain an iteration equation by using a generalization of Newton's method. Suppose the values of  $b_0$  and  $b_1$  at the  $k$ th iteration are  $b_0^{(k)}$  and  $b_1^{(k)}$ , respectively. Then the leading terms of Taylor expansions for  $c_1$  and  $c_0$  in terms of their variables  $b_1$  and  $b_0$  are:

$$c_1(b_1, b_0) = c_1 + \frac{\partial c_1}{\partial b_1}(b_1 - b_1^{(k)}) + \frac{\partial c_1}{\partial b_0}(b_0 - b_0^{(k)}) + \dots \\ c_0(b_1, b_0) = c_0 + \frac{\partial c_0}{\partial b_1}(b_1 - b_1^{(k)}) + \frac{\partial c_0}{\partial b_0}(b_0 - b_0^{(k)}) + \dots$$

If we drop all Taylor series terms besides the ones shown, and set each series equal to zero, the result, in matrix form, is

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_0 \end{bmatrix} + \begin{bmatrix} \frac{\partial c_1}{\partial b_1} & \frac{\partial c_1}{\partial b_0} \\ \frac{\partial c_0}{\partial b_1} & \frac{\partial c_0}{\partial b_0} \end{bmatrix} \begin{bmatrix} b_1 - b_1^{(k)} \\ b_0 - b_0^{(k)} \end{bmatrix}$$

From this we easily obtain the iteration equation

$$\begin{bmatrix} b_1^{(k+1)} \\ b_0^{(k+1)} \end{bmatrix} = \begin{bmatrix} b_1^{(k)} \\ b_0^{(k)} \end{bmatrix} - \begin{bmatrix} \frac{\partial c_1}{\partial b_1} & \frac{\partial c_1}{\partial b_0} \\ \frac{\partial c_0}{\partial b_1} & \frac{\partial c_0}{\partial b_0} \end{bmatrix}^{-1} \begin{bmatrix} c_1 \\ c_0 \end{bmatrix}$$

To get expressions for the partial derivatives, we differentiate equation (8.9) with respect to  $b_1$  first, and then with respect to  $b_0$ , obtaining the equations

$$0 = zQ(z) + B(z) \frac{\partial Q(z)}{\partial b_1} + \frac{\partial c_1}{\partial b_1} z + \frac{\partial c_0}{\partial b_1} \quad (8.10)$$

$$0 = Q(z) + B(z) \frac{\partial Q(z)}{\partial b_0} + \frac{\partial c_1}{\partial b_0} z + \frac{\partial c_0}{\partial b_0} \quad (8.11)$$

It is helpful at this point to consider polynomial equations of the general form

$$0 = B(z)S(z) + \alpha z + \beta$$

where  $S(z)$  is some real polynomial, and  $\alpha$  and  $\beta$  are real numbers. The highest order coefficient of  $S(z)$ , which we may take to be the coefficient of  $z^k$ , must be zero. Otherwise, after multiplication by  $B(z)$ , a nonzero term in  $z^{k+2}$  appears on the right side of the equation. Continuing the reasoning, all the coefficients of  $S(z)$  are zero, and, consequently, so are  $\alpha$  and  $\beta$ . To make use of this observation, we divide  $Q(z)$  by  $B(z)$  to obtain, as shown in the following equation, the quotient  $Q_1(z)$  and the remainder  $d_1z + d_0$ :

$$Q(z) = B(z)Q_1(z) + d_1z + d_0$$

After we substitute this result in equation (8.11), we obtain the equation

$$0 = B(z) \left[ Q_1(z) + \frac{\partial Q(z)}{\partial b_0} \right] + \left( d_1 + \frac{\partial c_1}{\partial b_0} \right) z + \left( d_0 + \frac{\partial c_0}{\partial b_0} \right)$$

By our previous reasoning, this implies

$$\frac{\partial c_1}{\partial b_0} = -d_1$$

$$\frac{\partial c_0}{\partial b_0} = -d_0$$

After we make a similar substitution in equation (8.10), and replace the term  $d_1z^2$  by

$$d_1[B(z) - b_1z - b_0] = d_1B(z) - d_1b_1z - d_1b_0$$

we get

$$0 = B(z) \left[ zQ_1(z) + \frac{\partial Q(z)}{\partial b_1} + d_1 \right] + \left( d_0 - d_1 b_1 + \frac{\partial c_1}{\partial b_1} \right) z + \left( -d_1 b_0 + \frac{\partial c_0}{\partial b_1} \right)$$

This implies

$$\begin{aligned} \frac{\partial c_1}{\partial b_1} &= d_1 b_1 - d_0 \\ \frac{\partial c_0}{\partial b_1} &= d_1 b_0 \end{aligned}$$

We have then the matrix equation

$$\begin{bmatrix} \frac{\partial c_1}{\partial b_1} & \frac{\partial c_1}{\partial b_0} \\ \frac{\partial c_0}{\partial b_1} & \frac{\partial c_0}{\partial b_0} \end{bmatrix} = \begin{bmatrix} d_1 b_1 - d_0 & -d_1 \\ d_1 b_0 & -d_0 \end{bmatrix}$$

with a determinant  $D = d_1^2 b_0 - (d_1 b_1 - d_0) d_0$ . As may be verified by multiplication, the inverse matrix is

$$\begin{bmatrix} \frac{\partial c_1}{\partial b_1} & \frac{\partial c_1}{\partial b_0} \\ \frac{\partial c_0}{\partial b_1} & \frac{\partial c_0}{\partial b_0} \end{bmatrix}^{-1} = \frac{1}{D} \begin{bmatrix} -d_0 & d_1 \\ -d_1 b_0 & d_1 b_1 - d_0 \end{bmatrix}$$

The Bairstow method consists of choosing some initial quadratic  $z^2 + b_1^{(0)}z + b_0^{(0)}$  and then performing the following cycle. The problem polynomial  $P(z)$  is divided by  $B^{(k)}(z) = z^2 + b_1^{(k)}z + b_0^{(k)}$  to obtain the quotient polynomial  $Q(z)$  and the remainder  $c_1 z + c_0$ . Then  $Q(z)$  is divided by  $B^{(k)}(z)$  to obtain a quotient and a remainder  $d_1 z + d_0$ . The quadratic coefficients  $b_1^{(k+1)}, b_0^{(k+1)}$  are formed from  $b_1^{(k)}, b_0^{(k)}$  by adding the amounts

$$\Delta b_1 = \frac{c_1 d_0 - c_0 d_1}{D} \quad \Delta b_0 = \frac{c_1 d_1 b_0 - c_0 (d_1 b_1 - d_0)}{D}$$

If we find  $b_1^{(k+1)} \neq b_1^{(k)}$  or find  $b_0^{(k+1)} \neq b_0^{(k)}$ , then, in preparation for the next cycle, the ranges of  $b_1^{(k+1)}$  and  $b_0^{(k+1)}$  are set to zero. Otherwise, the iteration ends and our final quadratic approximation  $z^2 + b_1 z + b_0$  uses the values of the  $k$ th iterates, with again ranges set to 0. From this quadratic polynomial we obtain two roots of  $P(z)$  by using the quadratic formula  $(-b_1 \pm \sqrt{b_1^2 - 4b_0})/2$ . The quotient  $Q(z)$  becomes the new problem polynomial, and another cycle of quadratic approximations begins,



unless of course the degree of our problem polynomial has become two or one. In this case the remaining roots are found easily.

The Bairstow iteration must be monitored for steady progress toward a quadratic factor of  $P(z)$ . The value of  $|c_1| + |c_0|$  can be used as an indicator. If this indicator does not decrease with an iteration, the procedure can always be restarted by choosing new values for  $b_1^{(0)}$  and  $b_0^{(0)}$ . Because the Bairstow method is a variety of Newton's method, and we know Newton's method converges linearly toward a multiple zero, we can expect Bairstow's method to converge slowly toward a multiple quadratic factor. Thus we see the advantage of using the Euclidean procedure to obtain problem polynomials  $N_i(z)$  having no multiple roots.

## 8.4 Bounding the error of a rational polynomial's root approximations

After a rational polynomial has been dissected into polynomials  $N_1(z), \dots, N_r(z)$  and the roots of each of these polynomials found by Bairstow's method, each root being an exact complex number, the next problem is to determine an error bound for each root. In this section let  $P(z)$  denote any of the  $N_i(z)$  polynomials, and let  $n$  be its degree. We have then distinct root approximations  $w_1, w_2, \dots, w_n$  to the simple roots of  $P(z)$ . These approximations are the actual roots of a certain polynomial  $Q(z)$  of degree  $n$ , which has the factorization

$$Q(z) = (z - w_1)(z - w_2) \dots (z - w_n)$$

The quotient  $\frac{P(z)}{Q(z)}$  has the partial fraction expansion

$$\frac{P(z)}{Q(z)} = 1 + \sum_{i=1}^n \frac{h_i}{z - w_i} \quad (8.12)$$

with complex coefficients  $h_i$ . If we set  $z$  equal to any root  $z_p$  of  $P(z)$ , we get the equation

$$-1 = \sum_{i=1}^n \frac{h_i}{z_p - w_i}$$

Taking absolute values, we obtain the relations

$$1 = \left| \sum_{i=1}^n \frac{h_i}{z_p - w_i} \right| \leq \sum_{i=1}^n \frac{|h_i|}{|z_p - w_i|}$$

Suppose the sum term  $\frac{|h_i|}{|z_p - w_i|}$  is largest for  $i = i_0$ . Then we have

$$1 \leq n \frac{|h_{i_0}|}{|z_p - w_{i_0}|}$$

$$|z_p - w_{i_0}| \leq n|h_{i_0}|$$

Now suppose for each root approximation  $w_i$  we compute the quantity

$$\varepsilon_i = n|h_i| \quad (8.13)$$

Then it is certain that every root of  $P(z)$  lies in one of the disks

$$|z - w_i| \leq \varepsilon_i \quad i = 1, 2, \dots, n \quad (8.14)$$

If these disks do not intersect each other, then each disk contains exactly one root of  $P(z)$ . Moreover, if  $m$  of these disks overlap, then the composite figure contains exactly  $m$  roots of  $P(z)$ . To see this, consider the polynomial

$$P_\lambda(z) = \lambda P(z) + (1 - \lambda)Q(z)$$

where the real parameter  $\lambda$  varies in the interval  $[0, 1]$ . The polynomial  $P_\lambda(z)$  equals  $Q(z)$  when  $\lambda = 0$ , and equals  $P(z)$  when  $\lambda = 1$ . We have

$$\frac{P_\lambda(z)}{Q(z)} = \lambda \frac{P(z)}{Q(z)} + (1 - \lambda) = 1 + \sum_{i=1}^n \frac{\lambda h_i}{z - w_i}$$

and according to our previous analysis, each root of  $P_\lambda(z)$  lies in one of the disks

$$|z - w_i| \leq \lambda \varepsilon_i \quad i = 1, 2, \dots, n$$

The radius of each disk varies directly with  $\lambda$ , being zero when  $\lambda = 0$ , and equal to the radius of the  $P(z)$  disk when  $\lambda = 1$ . When  $\lambda$  is 0, the point disk at  $w_i$  has a root of  $Q(z)$ . As  $\lambda$  varies in  $[0, 1]$ , the various roots of  $P_\lambda(z)$  are continuous functions of  $\lambda$ . Because the number of roots in all the disks is always  $n$ , the number of roots of  $P_\lambda(z)$  in the disk with center  $w_i$  remains one, as long as this disk does not intersect any of the other disks. Otherwise, there would be a discontinuous jump of at least one root of  $P_\lambda(z)$ . Similarly, when  $\lambda$  is of a size that  $m$  of the  $P_\lambda(z)$  disks overlap, the total number of roots of  $P_\lambda(z)$  in the overlapping disks must equal  $m$ .

The radius  $\varepsilon_i$  obtained for the approximation  $w_i$  is used to assign appropriate ranges to  $w_i$ , changing  $w_i = x_i + iy_j$  to  $\hat{w}_i = (x_i \oplus \varepsilon_i) + i(y_j \oplus \varepsilon_i)$ . The next step

is to compare the approximations  $\hat{w}_i$  with each other to make certain that no two  $\hat{w}_i$  overlap. If some of them overlap, then the radii  $\varepsilon_i$  of the overlapping disks are summed, and twice the sum becomes the new radius assigned to approximations  $w_i$  of the overlapping set.

The quantity  $h_i$  needed for the disk radius of  $w_i$  is easily calculated. If we multiply equation (8.12) by  $(z - w_i)$  we obtain the equation

$$\frac{P(z)}{\prod_{j \neq i} (z - w_j)} = (z - w_i) + h_i + \sum_{j \neq i} \frac{h_j(z - w_i)}{z - w_j}$$

When  $z$  is set equal to  $w_i$ , we obtain

$$\frac{P(w_i)}{\prod_{j \neq i} (w_i - w_j)} = h_i$$

## 8.5 Finding accurate roots for a rational or a real polynomial

The preceding sections have described three tasks, that when done for a rational polynomial  $P(z)$ , yield accurate root approximations. First  $P(z)$  is dissected into polynomials  $N_i(z)$  having simple roots. Then the Bairstow method is used to obtain root approximations for the polynomials  $N_i(z)$ . Finally error bounds are assigned to these roots using the partial fraction method of the preceding section.

When  $P(z)$  is a rational polynomial, the polynomials  $N_i(z)$  make it easier to compute the error of the root approximations. After the root approximations of a polynomial  $N_i(z)$  are assembled, we can compute their error bounds as simple roots of  $N_i(z)$ , rather than as possibly multiple roots of the original problem polynomial  $P(z)$ . If the  $N_i(z)$  roots are not found to the desired number of correct decimal places, we need only increase the precision of computation and repeat the root finding and error bounding procedure on  $N_i(z)$  alone.

For a real polynomial  $P(z)$ , the Euclidean procedure can be used to obtain polynomials  $N_i(z)$ , but here there is ambiguity whether the computed polynomials  $N_i(z)$  are correct. Consider the range arithmetic division of the Euclidean procedure, where a polynomial  $R_j(z)$  is divided by a polynomial  $R_{j+1}(z)$  of degree  $k$ . The remainder polynomial has the general form

$$d_{k-1}z^{k-1} + d_{k-2}z^{k-2} + \cdots + d_1z + d_0$$

Suppose we find that a certain number of leading coefficients overlap zero, that is,  $0 \doteq d_{k-1} \doteq d_{k-2} \doteq \cdots \doteq d_s$ . We take this result to mean that

$0 = d_{k-1} = d_{k-2} = \dots = d_s$ , and if  $d_s$  is  $d_0$ , the Euclidean procedure has terminated; otherwise the next polynomial  $R_{j+2}(z)$  is of degree  $s - 1$ . But perhaps one or more of the coefficients presumed to be zero are actually not zero. In that case the precision of computation should have been increased, and the Euclidean procedure redone. Thus with range arithmetic, at any given precision of computation, the Euclidean procedure yields polynomials  $N_i(z)$ , but we are uncertain whether they are correct. When the root approximations for all polynomials  $N_i(z)$  are assembled, and we are ready to bound their error, we must use the original polynomial  $P(z)$  in the error bounding process. If it turns out that not enough correct decimal places have been determined, we must increase the precision of computation appropriately and repeat all steps of our procedure, and that includes a new determination of the polynomials  $N_i(z)$ .

The determination of error bounds for the root approximations is now more complex. Suppose for the monic real polynomial  $P(z)$  of degree  $n$ , we have found distinct root approximations  $w_1, w_2, \dots, w_q$  with respective apparent multiplicities  $n_1, n_2, \dots, n_q$  summing to  $n$ . These approximations are the actual roots of a certain polynomial  $Q(z)$  of degree  $n$  which has the factorization

$$Q(z) = (z - w_1)^{n_1} (z - w_2)^{n_2} \dots (z - w_q)^{n_q}$$

The function  $\frac{P(z)}{Q(z)}$  has the partial fraction expansion

$$\frac{P(z)}{Q(z)} = 1 + \sum_{i=1}^q \sum_{j=1}^{n_i} \frac{h_{ij}}{(z - w_i)^j} \tag{8.15}$$

with complex coefficients  $h_{ij}$ . If we set  $z$  equal to any root  $z_p$  of  $P(z)$ , we get the equation

$$-1 = \sum_{i=1}^q \sum_{j=1}^{n_i} \frac{h_{ij}}{(z_p - w_i)^j}$$

Taking absolute values, we obtain the relations

$$1 = \left| \sum_{i=1}^q \sum_{j=1}^{n_i} \frac{h_{ij}}{(z_p - w_i)^j} \right| \leq \sum_{i=1}^q \sum_{j=1}^{n_i} \frac{|h_{ij}|}{|z_p - w_i|^j}$$

Suppose the double sum term  $\frac{|h_{ij}|}{|z_p - w_i|^j}$  is largest for  $i = i_0$  and  $j = j_0$ . Then we have

$$\begin{aligned} 1 &\leq n \frac{|h_{i_0, j_0}|}{|z_p - w_{i_0}|^{j_0}} \\ |z_p - w_{i_0}|^{j_0} &\leq n |h_{i_0, j_0}| \\ |z_p - w_{i_0}| &\leq (n |h_{i_0, j_0}|)^{\frac{1}{j_0}} \end{aligned}$$

Now suppose for each root approximation  $w_i$  we compute the quantity

$$\varepsilon_i = \max_{j=1}^{n_i} (n_i |h_{ij}|)^{\frac{1}{j}} \quad (8.16)$$

Then it is certain that every root of  $P(z)$  lies in one of the disks

$$|z - w_i| \leq \varepsilon_i \quad i = 1, 2, \dots, q \quad (8.17)$$

The same reasoning that was used with an intermediate polynomial  $P_\lambda(z)$  in the preceding section allows us to conclude that if these disks do not intersect each other, then the disk associated with the approximation  $w_i$  of apparent multiplicity  $n_i$  contains exactly  $n_i$  roots of  $P(z)$ . Moreover, if several of these disks overlap, and the sum of the associated multiplicities of these disks is  $m$ , then the composite figure contains exactly  $m$  roots of  $P(z)$ .

The general plan for computing error bounds is as follows. Suppose we have a set of exact  $P(z)$  root approximations  $w_i$ , computed by either the Bairstow method or some other method. These root approximations have apparent multiplicities assigned by the  $N_i(z)$  decomposition of  $P(z)$ . After the various quantities  $h_{ij}$  are computed, by a method described later in this section, the radius  $\varepsilon_i$  is obtained for each approximation  $w_i = x_i + iy_j$ , and this radius is used to assign appropriate ranges to  $w_i$ , changing it to  $\hat{w}_i = (x_i \oplus \varepsilon_i) + i(y_i \oplus \varepsilon_i)$ . The next step is to compare the approximations  $\hat{w}_i$  with each other to make certain that no two overlap, that is, to make certain that  $\hat{w}_i \neq \hat{w}_j$  for  $i \neq j$ . If some of them overlap, the original approximations  $w_i$  need to be combined corresponding to the overlap, choosing arbitrarily one approximation  $w_i$  to represent the set of overlapping approximations, and increasing apparent multiplicities appropriately. Then the error bound computation is repeated. When finally there is no overlap, an approximation  $\hat{w}_i$  of apparent multiplicity  $m$  is a correctly ranged approximation to  $m$  roots of  $P(z)$ , though these  $m$  roots need not all be equal.

Suppose we are trying to compute root approximations to a certain number of correct fixed-point decimal places, and we have a root approximation  $w_i$  of apparent multiplicity  $m$ . For  $\varepsilon_i$  to be no greater than  $10^{-k}$ , according to equation (8.16) we must have  $|nh_{i,m}|$  no larger than  $10^{-km}$ . We see then that to achieve  $k$  correct decimal places, we need to carry around  $km$  decimal places in our computations for these approximations. Multiple roots require higher precision computation than simple roots, and the precision needed is proportional to the multiplicity.

We consider next the method of computing the needed quantities  $h_{i,j}$ . This essentially is a Taylor series computation. If we multiply equation (8.15) by  $(z - w_i)^{n_i}$ , we obtain

$$\frac{P(z)}{\prod_{j \neq i} (z - w_j)^{n_j}} = h_{i,n_i} + h_{i,n_i-1}(z - w_i) + \dots + h_{i,1}(z - w_i)^{n_i-1} + \dots$$

On the right side of the equals sign we have collected all the terms that contain powers of  $(z - w_i)$  with an exponent less than  $n_i$ . The other terms on the right

side can themselves be expanded in a Taylor series about the expansion point  $w_i$ , but the factor  $(z - w_i)^{n_i}$  that they have prevents any terms in  $(z - w_i)^k$  appearing with an exponent  $k$  less than  $n_i$ . Thus the terms in  $(z - w_i)$  shown are the correct leading Taylor series terms for the function on the left. Therefore if the function on the left is obtained as a power series, with the series expansion point being  $w_i$ , we obtain the needed values  $h_{ij}$  as the first  $n_i$  series coefficients. We need the two expansions

$$P(z) = b_0 + b_1(z - w_i) + b_2(z - w_i)^2 + \cdots + (z - w_i)^{n_i-1} + \cdots$$

$$\prod_{j \neq i} (z - w_j)^{n_j} = c_0 + c_1(z - w_i) + c_2(z - w_i)^2 + \cdots + (z - w_i)^{n_i-1} + \cdots$$

and then we can form the needed terms by doing a series division, using relation (5.6) of Chapter 5.

## 8.6 The demo program roots

The `roots` program obtains from the user the degree and coefficients of the problem polynomial  $P(z)$ , and also the number of correct decimal places to be obtained. The program's first task is to determine whether the specified problem polynomial is a rational polynomial. For each entered coefficient, a ranged number is obtained at the current precision when the coefficient is specified by a keyboard entry. The coefficient can be converted to a rational in either of two ways. First, if the obtained number is exact, that is, has a zero range, then the exact value is transformed into a rational number in the two-integer form  $p/q$ . Second, if the number's range is positive, then the coefficient's evaluation list is examined to determine whether standard function evaluations occur. If there are none, the number must be rational, and so the evaluation list can be used to generate a rational number in the form  $p/q$ .

If `roots` is able to obtain rational values for all coefficients, then `roots` uses the system for obtaining accurate root values described in the first four sections of this chapter. If for any coefficient, `roots` fails to obtain a rational value, then `roots` uses the more difficult procedure described in the preceding section.

## Software Exercises F

These exercises are for the demo program `roots`.

1. Call up the program `roots` and verify that the polynomial  $x^4 - 10x^3 - 35x^2 - 50x - 24$  of Section 8.2 has the single positive root  $13.00\sim$ .

2. This exercise and the next two exercises show how multiple roots are specified, which depends on whether the program `roots` is successful in obtaining a rational polynomial. Call up `roots`, specify the polynomial  $z^2 + 2z + 1$ , and obtain the roots to 5 decimal places. The polynomial  $z^2 + 2z + 1 = (z + 1)^2$ , so a single root of multiplicity 2 is displayed.
3. Modify the `roots` log file, changing the 2 coefficient to  $2 \tan(\pi/4)$  by entering `2*tan(pi/4)`, and then call up `roots roots`. Although  $2 \tan(\pi/4) = 2$ , note that this time a disclaimer is made about the multiplicity of the displayed root, because a rational polynomial was not obtained.
4. Modify the `roots` log file, changing the 2 coefficient to  $2 \cos(0)$  and then call up `roots roots`. Note that this time no multiplicity disclaimer appears. The disclaimer is absent because the ranged value obtained for the 2 coefficient has a zero range, and so the number's evaluation list is not consulted. Some standard function values are obtained as exact numbers. For instance, the keyboard entries `sin(0)`, `cos(0)`, and `exp(0)` all generate exact values.

## Notes and References

- A. The use of the Euclidean algorithm to determine the multiplicity structure of a polynomial is discussed in Uspensky's textbook [4] and in the paper by Dunaway [2].
  - B. Theorem 8.1 and 8.2 were discovered by Cauchy. Wilf's book [5] discusses these and other polynomial root bounds.
  - C. Our presentation of Bairstow's method follows that of Hamming [3].
  - D. The error bound for polynomials in Section 8.5 was given by Braess and Hadeler [1].
- 
- [1] Braess, D. and Hadeler, K. P., Simultaneous inclusion of the zeros of a polynomial, *Numer. Math.* **21** (1973), 161–165.
  - [2] Dunaway, D. K., Calculation of zeros of a real polynomial through factorization using Euclid's algorithm, *SIAM J. Numer. Anal.* **11** (1974), 1087–1104.
  - [3] Hamming, R. W., *Numerical Methods for Scientists and Engineers*, 2nd Edn, McGraw-Hill, New York, 1973.
  - [4] Uspensky, J. V., *Theory of equations*, McGraw-Hill, New York, 1948.
  - [5] Wilf, H. S., *Mathematics for the physical sciences*, John Wiley & Sons, New York, 1962.

# Solving $n$ Linear Equations in $n$ Unknowns



Three demo programs solve linear equations for the case where the number of unknowns matches the number of equations. The program `equat` is for equations with real coefficients the program `r_equat` is for equations with rational coefficients and the program `c_equat` is for equations with complex coefficients. The next section introduces matrix notation, and the succeeding section uses the ideas of Chapter 3 to define appropriate computation objectives. The succeeding sections present methods for solving linear equations and obtaining matrix determinants, and finally, in the last section, the three demo programs are described.

## 9.1 Notation

For a matrix  $A$  of  $m$  rows and  $n$  columns, or, more briefly, of size  $m \times n$ , it is customary to indicate the  $A$  elements (or  $A$  components) by using a small letter matching the matrix capital letter, and attaching row and column indices, with the row index always preceding the column index. Thus  $A$  has elements  $a_{ij}$ . The matrix  $A$  is *real* if all elements are real numbers, is *rational* if all elements are rational numbers, and is *complex* if all elements are complex numbers. The matrix is *square* when  $m = n$ , and in that case the matrix  $A$  has a *determinant*, indicated by the symbol  $\det A$ . A square matrix may also be called  $n$ -square to indicate the number  $n$  of rows and columns.

An *upper triangular* matrix is an  $n$ -square matrix of the form

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ & a_{22} & a_{23} & \cdots & a_{2n} \\ & & a_{33} & \cdots & a_{3n} \\ & & & \ddots & \vdots \\ & & & & a_{nn} \end{bmatrix}$$



An entirely blank region of a displayed matrix, as shown here, is always made up entirely of zero elements. Thus an upper triangular matrix can be characterized as a square matrix whose elements below the diagonal are all zero. The diagonal elements of any matrix, square or not, are those elements with matching row and column indices. The determinant of an upper triangular matrix equals the product of its diagonal elements. Similarly, a lower triangular matrix is an  $n$ -square matrix with zero elements above its diagonal, and its determinant also is equal to the product of its diagonal elements.

We use the symbol  $I$  to denote an  $n$ -square identity matrix, its size inferred from context, and the symbol  $O$  to denote an  $m \times n$  matrix with all elements zero, its size also inferred from context. If there is only one column to a matrix, it is customary to omit the unchanging column index for its elements, and to call the matrix a *vector*. We use lower case boldface letters to denote vectors. So if  $\mathbf{c}$  is a vector, it has the form

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

The vector  $\mathbf{c}$  may also be called an  $n$ -vector to indicate the number of its components. The length of  $\mathbf{c}$ , denoted by  $\text{len } \mathbf{c}$ , is  $\sqrt{\sum_{i=1}^n c_i^2}$  if  $\mathbf{c}$  is real, and is  $\sqrt{\sum_{i=1}^n |c_i|^2}$  if  $\mathbf{c}$  is complex.

The transpose of a matrix  $A$  of size  $m \times n$ , denoted by the symbol  $A^T$ , is a matrix  $D$  of size  $n \times m$  with  $d_{ij}$  equal to  $a_{ji}$ . If the matrix product  $AB$  is defined, then the rule  $(AB)^T = B^T A^T$  holds.

## 9.2 Computation problems

A frequently encountered problem is that of solving the system of linear equations

$$\begin{array}{cccc} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & = & b_2 \\ \vdots & & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n & = & b_n \end{array}$$

for the values of  $x_1, x_2, \dots, x_n$ . This is equivalent to solving the matrix equation

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Assigning appropriate letters to the three arrays, the problem is then to solve

$$A\mathbf{x} = \mathbf{b}$$

for the vector  $\mathbf{x}$ . The matrix  $A$  is often called the *coefficient matrix*. In a linear algebra course, it is shown that this problem has a unique solution vector  $\mathbf{x}$  if and only if  $\det A$  is nonzero. If  $\det A = 0$ , there are two possibilities: either an infinite number of solutions, or no solutions at all. This applies whether the arrays  $A$ ,  $\mathbf{b}$  are real or complex.

Suppose the computation goal for the real case is

**Nonsolvable Problem 9.1** Given a set of  $n$  linear equations in  $n$  unknowns, represented by the matrix equation  $A\mathbf{x} = \mathbf{b}$ , where  $A$  is a real  $n$ -square coefficient matrix,  $\mathbf{x}$  is the vector of unknowns, and  $\mathbf{b}$  the real vector of constants, find to  $k$  decimal places the  $\mathbf{x}$  components, or else indicate that  $\det A = 0$ .

The difficulty here is the necessity of determining whether or not  $\det A = 0$ , an instance of Nonsolvable Problem 3.1. A similar difficulty would occur for the complex case. To get a solvable problem, we give up trying to determine whether the  $A$  determinant is *exactly* zero. Rather than list two problems, one for the real case and one for the complex case, we list them together this way:

**Solvable Problem 9.2** Given a set of  $n$  linear equations in  $n$  unknowns, represented by the matrix equation  $A\mathbf{x} = \mathbf{b}$ , where  $A$  is a real (complex)  $n$ -square coefficient matrix,  $\mathbf{x}$  is the vector of unknowns, and  $\mathbf{b}$  the real (complex) vector of constants, find to  $k$  decimal places the  $\mathbf{x}$  components, or else indicate that  $|\det A| < 10^{-k}$ .

The bound  $10^{-k}$  is merely a convenient one and could be replaced by others, for instance,  $10^{-nk}$  or  $10^{-k^2}$ .

If  $A$  and  $\mathbf{b}$  are both rational, the difficulty in determining whether  $\det A = 0$  disappears. If  $\det A \neq 0$ , we obtain  $\mathbf{x}$  as a vector of rational numbers, which can always be displayed to any desired number of correct decimal places.

**Solvable Problem 9.3** Given a set of  $n$  linear equations in  $n$  unknowns, represented by the matrix equation  $A\mathbf{x} = \mathbf{b}$ , where  $A$  is a rational  $n$ -square coefficient matrix,  $\mathbf{x}$  is the  $n$ -vector of unknowns, and  $\mathbf{b}$  the rational  $n$ -vector of constants, find as rational numbers the unknowns  $x_1, \dots, x_n$ , or else indicate that  $\det A = 0$ .

Suppose for a square matrix  $A$  we wish to obtain the square inverse matrix  $A^{-1}$ , satisfying the equation  $AA^{-1} = I$ . The inverse matrix  $A^{-1}$  exists if  $\det A \neq 0$ , because if  $\mathbf{x}_i$  is the  $i$ th column of  $A^{-1}$  and  $\mathbf{b}_i$  is the  $i$ th column of  $I$ , then  $\mathbf{x}_i$  can

be determined by solving the system of linear equations  $A\mathbf{x} = \mathbf{b}$ . In a linear algebra course, it is shown that if the equation  $AA^{-1} = I$  holds, then the equation  $A^{-1}A = I$  holds also.

An appropriate computation objective for matrix inverses is

**Solvable Problem 9.4** Given the  $n$ -square real (complex) matrix  $A$ , find to  $k$  decimal places the elements of the matrix  $A^{-1}$ , or else indicate that  $|\det A| < 10^{-k}$ .

When  $A$  is rational, a stronger result is possible, as specified in

**Solvable Problem 9.5** Given the  $n$ -square rational matrix  $A$ , find the rational elements of the matrix  $A^{-1}$ , or else indicate that  $\det A = 0$ .

### 9.3 A method for solving linear equations

A system of  $n$  linear equations in  $n$  unknowns can be solved by the elimination method, whereby the equations are manipulated until they have the matrix representation  $A'\mathbf{x} = \mathbf{b}'$ , where  $A'$  is upper triangular. If we are successful in obtaining this representation for the equations, then they have the form

$$\begin{aligned} a'_{11}x_1 + a'_{12}x_2 + \cdots + a'_{1n-1}x_{n-1} + a'_{1n}x_n &= b'_1 \\ a'_{22}x_2 + \cdots + a'_{2n-1}x_{n-1} + a'_{2n}x_n &= b'_2 \\ \vdots & \\ a'_{n-1n-1}x_{n-1} + a'_{n-1n}x_n &= b'_{n-1} \\ a'_{nn}x_n &= b'_n \end{aligned}$$

and are easily solved by back substitution, namely

$$\begin{aligned} x_n &= \frac{1}{a'_{nn}}(b'_n) \\ x_{n-1} &= \frac{1}{a'_{n-1,n-1}}(b'_{n-1} - a'_{n-1,n}x_n) \\ &\vdots \\ x_2 &= \frac{1}{a'_{22}}(b'_2 - a'_{23}x_3 - \cdots - a'_{2n}x_n) \\ x_1 &= \frac{1}{a'_{11}}(b'_1 - a'_{12}x_2 - \cdots - a'_{1n}x_n) \end{aligned}$$

All these equations can be combined as

$$x_i = \frac{1}{a'_{ii}} \left( b'_i - \sum_{k=i+1}^n a'_{ik} x_k \right) \quad i = n, n-1, \dots, 2, 1$$

with the understanding that the sum is void for  $x_n$ , because here the starting index  $n+1$  is greater than the ending index  $n$ . The possibly void summation is clearer when expressed this way:

$$x_i = \frac{1}{a'_{ii}} \left( b'_i - \sum_{i < k \leq n} a'_{ik} x_k \right) \quad i = n, n-1, \dots, 2, 1 \quad (9.1)$$

The manipulation of the equations is done most conveniently by repeatedly changing the stored representations of  $A$  and  $\mathbf{b}$  until the desired upper triangular form of  $A$  is obtained. The operations performed on these matrices are of just two types:

**Exchange Row**( $i, j$ ): Exchange rows  $i$  and  $j$ .

**Add Row Multiple**( $i, j, M$ ): Add  $M$  times row  $i$  to a different row  $j$ .

Either matrix operation always is performed simultaneously on  $A$  and on  $\mathbf{b}$ . When either operation is executed, the set of linear equations represented by  $A\mathbf{x} = \mathbf{b}$  is changed by an allowable algebraic manipulation into a different set of equations. The two sets of equations are equivalent, because it is possible to undo either row operation by another operation of the same type.

The upper triangular form of  $A$  is obtained by bringing the columns of  $A$  to the proper form, one by one, starting at column 1 and working toward the right. For the sake of simplicity, we use  $a'_{ij}$  to denote an element of  $A$  during this process, and we do not try to differentiate between the successive values that the element may take. The procedure varies slightly depending on whether  $A$  and  $\mathbf{b}$  are rational. The procedure for the real or complex case is given first, with the minor changes for the rational case supplied later.

When working on column  $j$ , the procedure is the following. The elements  $a'_{jj}$ ,  $a'_{j+1j}$ ,  $\dots$ ,  $a'_{nj}$  are tested to locate the one with the “best” absolute value. The best absolute value is the largest absolute value, but because we can only decide for two ranged numbers  $c$  and  $d$  whether  $|c| > |d|$ ,  $|c| \doteq |d|$ , or  $|c| < |d|$ , we proceed cautiously. Let  $r$  be the row index of the column element that is currently largest in absolute value, and let  $L$  be the current largest absolute value. For column  $j$  the first element we test is  $a'_{jj}$ , so initially  $r = j$  and  $L = |a'_{jj}|$ . The elements with row index  $k > j$  are tested in sequence, and if  $|a'_{kj}| > L$ , then  $L$  is replaced by  $|a'_{kj}|$  and  $r$  is replaced by  $k$ . After the test of the column is complete, if we find  $L > 0$  with  $r \neq j$ , then the operation **Exchange Row**( $j, r$ ) is performed to convert element  $a'_{rj}$  into  $a'_{jj}$ . Then using the other row operation, appropriate multiples of row  $j$  are subtracted from rows below to clear the  $A$  elements in column  $j$  below  $a'_{jj}$ . Doing the clearing with the  $A$  element in column  $j$  having

the largest absolute value, rather than using any nonzero element in row  $j$  or below, has the advantage that smaller multiples of row  $j$  are subtracted from rows below, and this tends to make the  $A$  elements to the right of column  $j$  have smaller interval widths.

If we are successful in bringing  $A$  to upper triangular form, then we obtain the values of the variables  $x_i$  by using equation (9.1). We determine whether  $k$  correct decimal places have been obtained, and if not, we increase precision an appropriate amount and the entire computation is repeated.

Let us be more specific about the phrase “increase precision an appropriate amount,” which appears in the preceding paragraph and at other places in this chapter. The number of correct decimal places, in either fixed-point or scientific floating-point notation, that can be obtained from a ranged number can be determined from the number’s midpoint and interval halfwidth. By examining all the computed answers of a problem, it is possible to determine whether the target number of correct decimal places has been achieved, and, if not, what the maximum deficit in correct places is. When the computation is redone because answers are not accurate enough, precision is increased by at least the maximum deficit, and perhaps a few extra places, just to be more certain that the next cycle of computation will succeed.

It may happen when working with a certain column  $j$  of  $A$  that we encounter no element whose absolute value tests positive. In this case we find  $L \doteq 0$ , so we abandon the attempt to solve the equations, and compute  $\det A$  instead. A method for computing matrix determinants is described in the next section.

The procedure for the rational case differs, because here computation is with exact numerators and denominators, so there is no advantage in locating the element in column  $j$  with the largest absolute value. Only a nonzero element is needed in  $a'_{jj}$ , so if  $a'_{jj} \neq 0$ , the clearing operation can begin immediately. Otherwise a row exchange is needed to bring a nonzero element into the  $a'_{jj}$  position. Whenever a nonzero element in column  $j$  is not found, there is no need to compute the determinant, because it must be zero.

## 9.4 Computing determinants

The procedure for computing the determinant of a square matrix  $A$  is similar to the procedure already described for solving the matrix equation  $A\mathbf{x} = \mathbf{b}$ , except that now there is no vector  $\mathbf{b}$ . The matrix  $A$  is gradually brought to upper triangular form, column by column, starting with column 1 and working to the right, using the two matrix row operations given earlier. The procedure varies significantly, depending on whether the matrix  $A$  is rational. We consider the nonrational case first.

An operation **Add Row Multiple**( $i, j, M$ ) on  $A$  does not change the determinant of the varying matrix  $A$ . An operation **Exchange Row**( $i, j$ ) changes just

the sign of this determinant. To keep track of the sign changes, a test integer  $s$ , initially 1, is replaced by its negative each time an **Exchange Row**( $i, j$ ) operation is performed.

We again use  $a'_{ij}$  to denote the changing elements of the  $A$  matrix during the determinant computation procedure. After the matrix  $A$  is successfully brought to upper triangular form,  $\det A = \prod_{i=1}^n a'_{ij}$ , with the sign changed if the test integer  $s = -1$ .

It may happen when working with a certain column  $j$  of  $A$ , that all the quantities  $|a'_{jj}|, |a'_{j+1j}|, \dots, |a'_{nj}|$  test  $\doteq 0$ , indicating that the  $\det A$  interval contains the zero point. To obtain a narrow interval for  $\det A$ , a third operation is allowed:

**Exchange Col**( $i, j$ ): Exchange columns  $i$  and  $j$ .

This operation changes just the sign of  $\det A$ , and does not affect the width of the computed  $\det A$  interval. Our procedure now is to continue trying to bring  $A$  to upper triangular form, using, if necessary, this additional operation. Now when dealing with any column  $j$  of  $A$ , if in row  $j$  or below we can find no elements of positive absolute value, we search the columns of  $A$  further to the right for elements in row  $j$  or below with positive absolute value. If one is found in column  $k$  say, then we perform the operation **Exchange Col**( $j, k$ ) to bring this element into column  $j$ , and, if necessary, do a row exchange operation to make this element  $a'_{jj}$ . Next, we clear all elements below  $a'_{jj}$  by using **Add Row Multiple**( $i, j, M$ ) operations.

The procedure now will terminate when for some column  $q$ , no element of positive absolute value in row  $q$  or below is found in this column or in the other  $A$  columns further to the right. The final  $A$  matrix has the form

$$A' = \begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1q} & \cdots & a'_{1n} \\ & a'_{22} & \cdots & a'_{2q} & \cdots & a'_{2n} \\ & & \ddots & \vdots & \ddots & \vdots \\ & & & a'_{qq} & \cdots & a'_{qn} \\ & & & \vdots & \ddots & \vdots \\ & & & a'_{nq} & \cdots & a'_{nn} \end{bmatrix} \quad (9.2)$$

where the  $(n - q + 1)$ -square submatrix

$$\begin{bmatrix} a'_{qq} & \cdots & a'_{qn} \\ \vdots & \ddots & \vdots \\ a'_{nq} & \cdots & a'_{nn} \end{bmatrix}$$

has all elements  $\doteq 0$ . The determinant of the matrix  $A'$  is identical to the determinant of the starting matrix  $A$ , except that its sign may be wrong. The determinant

of  $A'$  equals  $\prod_{i=1}^{q-1} a_{ii}$  times the determinant of the submatrix. We can bound the magnitude of  $\det A'$  if we can bound the magnitude of the determinant of the submatrix.

In magnitude the determinant of the submatrix is not greater than the product of a magnitude bound for determinant terms times the number of terms in the determinant, which is  $(n - q + 1)!$ . The element  $a'_{ij}$  has the magnitude bound ( $|\text{midpoint } a'_{ij}| + \text{halfwidth } a'_{ij}$ ), so by taking the maximum of these for each submatrix column and multiplying these maxima together, we obtain a bound for determinant terms. After multiplying this by  $(n - q + 1)!$ , we obtain a magnitude bound  $M$  for the determinant of the submatrix. We obtain for  $\det A$  the interval value

$$\det A = 0 \pm |a'_{11} \cdot \dots \cdot a'_{q-1, q-1}| M$$

When the matrix  $A$  is rational, a rational value for  $\det A$  can be more easily determined, for two reasons. First, in the column  $j$  process, we do not search for the element of largest absolute value, but just for any nonzero element. And if no such element is found, the determinant is zero, and further computation is not needed.

## 9.5 Finding the inverse of a square matrix

As stated earlier, because  $AA^{-1} = I$ , column  $i$  of  $A^{-1}$ , denoted by the vector  $\mathbf{x}_i$ , can be found by taking the constant vector  $\mathbf{b}_i$  equal to the  $i$ th column of  $I$ , and then solving the linear equations represented by the matrix equation  $A\mathbf{x}_i = \mathbf{b}_i$ . However, it is more efficient to use the method described next.

We start with the matrix equation  $AA^{-1} = I$ , where the matrices  $A$  and  $I$  are known, and  $A^{-1}$  is unknown, and apply simultaneously to the matrices  $A$  and  $I$  a series of the operations **Exchange Row**( $i, j$ ) and **Add Row Multiple**( $i, j, M$ ), attempting with these row operations to bring  $A$  into upper triangular form. An operation **Exchange Row**( $i, j$ ) on an  $n$ -square matrix  $C$  is equivalent to premultiplying  $C$  by a matrix equal to the identity matrix  $I$ , except that rows  $i$  and  $j$  have been exchanged. Similarly an operation **Add Row Multiple**( $i, j, M$ ) on  $C$  is equivalent to premultiplying  $C$  by a matrix equal to the identity matrix  $I$ , except that  $M$  times row  $i$  has been added to row  $j$ . Thus we can justify performing either operation simultaneously on  $A$  and on  $I$  as simply the premultiplication of the equation  $AA^{-1} = I$  by another matrix to produce the equation  $A'A^{-1} = I'$ .

Following the same procedure that was described previously of gradually changing  $A$ , except that here we operate on  $A$  and  $I$  instead of  $A$  and  $\mathbf{b}$ , we may succeed in bringing  $A$  to upper triangular form. If we are unable to bring  $A$  to upper triangular form, then we abandon the goal of determining  $A^{-1}$  and

compute  $\det A$  instead. If we succeed in bringing  $A$  to upper triangular form, then we begin to use a new row operation on both  $A'$  and  $I'$ :

**Multiply Row**( $i, M$ ): Multiply row  $i$  by the nonzero constant  $M$ .

Again we can justify applying this operation to  $A'$  and  $I'$  as equivalent to the premultiplication of the equation  $A'A^{-1} = I'$  by a matrix equal to the identity matrix, except that row  $i$  has been multiplied by the constant  $M$ .

Working from column  $n$  of  $A$  backward toward column 1, in column  $j$  we apply the operation **Multiply Row**( $j, 1/a'_{jj}$ ) to both  $A'$  and  $I'$  to make  $a'_{jj}$  equal 1. The next part of the column  $j$  procedure is to apply a series of operations **Add Row Multiple**( $i, j, M$ ) to both  $A'$  and  $I'$  to clear the elements of  $A'$  above the 1 element.

After all columns of  $A'$  have been processed, the matrix  $A'$  has been converted to the identity matrix  $I$ , and the matrix equation, which initially was  $AA^{-1} = I$ , has been converted to  $IA^{-1} = I'$ . The elements of  $I'$  now equal the elements of  $A^{-1}$ . If this process leads to results that are insufficiently accurate, the entire calculation is repeated at an appropriate higher precision.

## 9.6 The demo programs `equat`, `r_equat`, and `c_equat`

All three programs have similar beginnings. The number of equations is requested from the user, and the integer obtained determines the size of the matrix  $A$  and vector  $\mathbf{b}$  arrays, which then get filled according to the user's keyboard entries.

The program `equat`, for real linear equations, continues as follows. After the number of correct decimal places of the solution vector has been specified by the user, `equat` initially follows the procedure of Section 9.3 for solving  $\mathbf{Ax} = \mathbf{b}$ . In that procedure, it sometimes happens that  $A$  cannot be converted to upper triangular form. In that case, before switching to the alternate task of computing a bound on  $\det A$ , the program `equat` attempts to construct, from the starting real coefficient matrix  $A$ , a same-sized rational matrix identical to  $A$ . The process of finding an equal rational value for an  $A$  element is similar to the process described in Section 8.6 for finding an equal rational coefficient for a real polynomial coefficient. If a rational coefficient matrix is obtained, then `equat` computes its determinant. A zero rational determinate indicates that  $A$  is singular, and no further computation is needed. A nonzero rational determinant is helpful, because the alternate objective of Solvable Problem 9.2, bounding the  $A$  determinant, now can be skipped. Instead `equat` increases precision by a fixed amount and tries the solution process again. Only when a rational matrix is not obtained, does `equat` pursue the alternate objective of computing a bound on  $\det A$ .

The program `r_equat` for rational linear equations, is somewhat simpler. The objective of the program is to obtain a solution vector of rational numbers,



and unlike the program `equat`, the program `r_equat` does not need a computation loop. The solution procedure for a rational coefficient matrix  $A$  and a rational constant vector  $\mathbf{b}$  always yields a rational solution vector or indicates that  $\det A = 0$ .

The program `c_equat` follows the solution procedure of Section 9.3, except for one difference. In the column  $j$  procedure, finding the absolute value of a complex  $A$  matrix element requires computing a square root, which is time-consuming. Instead `c_equat` computes for an element  $a + ib$  the quantity  $a^2 + b^2$ , that is, the absolute value squared, and uses the largest of these values to locate the best element to bring to the  $a'_{jj}$  position.

## Software Exercises G

These exercises show properties of the programs `r_equat` first and then `equat`. To keep the keyboard entry process short, all exercises are for two equations in two unknowns.

1. Call up `r_equat` and specify the coefficient matrix  $A = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$  and the vector  $\mathbf{b} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$ . Note that the solution vector  $\mathbf{x} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$  is displayed in rational  $p/q$  format.
2. This time call up `equat` and specify the same coefficient matrix and constant vector. The `equat` program now requests you to specify the number of correct decimal places, and let us choose 10. Note that the answers are displayed in exact form without tildes, indicating that the solution vector's components were obtained with zero ranges.
3. Now that a log file for `equat` is available, edit it by changing  $a_{11}$  from 1 to  $\sin(\pi/2)$ , which equals 1. Save the log file and call up `equat equat`. The solution vector this time appears with tildes, indicating that all its components have nonzero ranges. The changed entry affected both vector components, leading to nonexact computed values.
4. Edit the log file again, changing  $a_{12}$  from  $-1$  to 1, so that the  $A$  matrix is singular. Save the log file and call up `equat equat`. Note that this time the alternate objective of Problem 9.2 is pursued, with `equat` indicating that the  $A$  determinant is less than  $10^{-10}$  (because 10 decimal places were requested). Besides this indication, `equat` also shows the computed value of the determinant.
5. Edit the log file one last time, changing  $a_{11}$  back to 1. Save the log file and call up `equat equat`. Now all of  $A$ 's elements are convertible to rationals, and this time `equat` indicates that the  $A$  matrix is singular.

## Notes and References

- A. Two general references on linear algebra are the books by Gantmacher [2] and by Hohn [4]. Two references emphasizing computation methods of linear algebra are the books by Golub and Van Loan [3] and by Faddeev and Fadeeva [1].
- [1] Faddeev, D. K. and Faddeeva, V. V., *Computational Methods of Linear Algebra*; W. H. Freeman, San Francisco, 1963.
  - [2] Gantmacher, F. R., *Theory of Matrices, Vols 1 and 2*, Chelsea Publishing Co., New York, 1960.
  - [3] Golub, G. H. and Van Loan, C. E., *Matrix Computations, 2nd Edn*, Johns Hopkins University Press, Baltimore, 1989.
  - [4] Hohn, F. *Elementary Matrix Algebra, 3rd Edn*, Macmillan, New York, 1973.

This page intentionally left blank

# Eigenvalue and Eigenvector Problems



Three demo programs find eigenvalues and eigenvectors for an  $n$ -square matrix  $A$ . The program `eigen` is for matrices with real elements, the program `r_eigen` is for matrices with rational elements, and the program `c_eigen` is for matrices with complex elements. As a preparation for explaining how these programs work, the next section treats a linear equation problem differing somewhat from the problems treated in the preceding chapter. The following section then gives the basic terminology for eigenvalue and eigenvector problems. Following that, practical methods for obtaining eigenvalues and eigenvectors are presented, and finally, in the last section of this chapter, the three demo programs are described.

In this chapter, a concept of linear algebra is needed. The real or complex  $n$ -vectors  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$  are *linearly independent* if no vector of the set can be expressed as a linear combination of the others. An  $m \times n$  real or complex matrix  $A$  is said to have rank  $r$  if, thinking of its columns as  $m$ -vectors, the maximum number of linearly independent columns that can be found is  $r$ . We also require this linear algebra result, which is valid for both real and complex matrices:

**Theorem 10.1** Let  $B$  be any matrix of size  $m \times n$ . If  $C$  is an  $m$ -square matrix with  $\det C \neq 0$ , and  $D$  is an  $n$ -square matrix with  $\det D \neq 0$ , then  $B$ ,  $CB$ , and  $BD$  all have the same rank.



row's elements get exchanged also, and indicate the  $\mathbf{x}$  component corresponding to an  $\mathbf{x}'$  component. However, instead of an enlarged  $A$ , our description will suppose the following equivalent method is used. At the beginning of operations, a "horizontal" array  $H$  with component  $i$  equal to the integer  $i$  is constructed:

$$H = [1 \quad 2 \quad 3 \quad \dots \quad n] \text{ initially}$$

$$H = [H_1 \quad H_2 \quad H_3 \quad \dots \quad H_n] \text{ in general}$$

Then when we do an **Exchange Col**( $i, j$ ) operation on  $A$ , we also do the operation on  $H$  to switch the integers  $H_i$  and  $H_j$ . After all operations are complete, the indicator  $H$  shows how to form  $\mathbf{x}'$  from  $\mathbf{x}$ , that is,  $x'_i$  is component  $H_i$  of  $\mathbf{x}$ .

The final matrix  $A'$  that we obtain is identical to the matrix  $A'$  shown in Eq. (9.2), having a certain submatrix of elements  $\doteq 0$ , and this submatrix is shown again here:

$$\begin{bmatrix} a'_{qq} & \cdots & a'_{qn} \\ \vdots & \ddots & \vdots \\ a'_{nq} & \cdots & a'_{nn} \end{bmatrix} \tag{10.2}$$

We can solve our problem if this submatrix is 1-square. In that case, the lone submatrix element  $a'_{nn}$  must equal 0. This is because  $\det A$  is known to be 0, so  $\det A'$ , being equal to  $\det A$  or  $-\det A$ , must be 0 too. Because  $A'$  is upper-triangular,  $\det A'$  equals the product of its diagonal elements, and these elements, with the exception of  $a'_{nn}$ , are nonzero. The system of equations now has the form

$$\begin{aligned} a'_{11}x'_1 + a'_{12}x'_2 + \cdots + a'_{1n-1}x'_{n-1} + a'_{1n}x'_n &= 0 \\ a'_{22}x'_2 + \cdots + a'_{2n-1}x'_{n-1} + a'_{2n}x'_n &= 0 \\ \vdots & \\ a'_{n-1,n-1}x'_{n-1} + a'_{n-1,n}x'_n &= 0 \end{aligned}$$

To obtain a solution, we set  $x'_n$  equal to 1, and then solve for the other components of  $\mathbf{x}'$  by back substitution:

$$\begin{aligned} x'_n &= 1 \\ x'_{n-1} &= -\frac{1}{a'_{n-1,n-1}}(a'_{n-1,n}) \\ x'_{n-2} &= -\frac{1}{a'_{n-2,n-2}}(a'_{n-2,n-1}x'_{n-1} + a'_{n-2,n}) \\ \vdots & \\ x'_1 &= -\frac{1}{a'_{11}}(a'_{12}x'_2 + \cdots + a'_{1,n-1}x'_{n-1} + a'_{1,n}) \end{aligned}$$

We obtain a vector of length 1 by multiplying all components of  $\mathbf{x}'$  by  $1/\text{len } \mathbf{x}'$ , and we unscramble the effect of the column exchanges by consulting the  $H$  indicator, taking  $x'_i$  as  $\mathbf{x}_{H_i}$ , rearranging the vector's components accordingly.

If the  $A'$  submatrix of elements  $\stackrel{\neq}{=} 0$  is not 1-square, then there is a difficulty. This time we cannot be certain that all submatrix elements are actually 0. A recomputation of  $A'$  at a higher precision might result in a smaller submatrix, but then again, it might not. Let us suppose for a moment that all submatrix elements are 0. We can obtain  $n - q + 1$  linearly independent solution vectors  $\mathbf{x}_1, \dots, \mathbf{x}_{n-q+1}$ , by setting the array of variables  $(x'_q, x'_{q+1}, \dots, x'_n)$  equal successively to

$$(1, 0, \dots, 0), (0, 1, \dots, 0), \dots, (0, 0, \dots, 1)$$

each time solving for the rest of the variables  $x'_i$  by back substitution, adjusting values afterward to achieve a length of 1, and then reassigning variable values  $x'_i$  to their proper position in the  $\mathbf{x}$  vector by consulting the indicator  $H$ . This is in accordance with a theorem of linear algebra:

**Theorem 10.2** If the  $n$ -square real (complex) matrix  $A$  has rank  $r$ , less than  $n$ , then there exist  $n - r$  linearly independent solution vectors  $\mathbf{x}_1, \dots, \mathbf{x}_{n-r}$  to the matrix equation  $A\mathbf{x} = \mathbf{0}$ , and a vector  $\mathbf{x}$  is a solution vector if and only if it can be written in the form

$$\mathbf{x} = c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \dots + c_{n-r}\mathbf{x}_{n-r}$$

where  $c_1, c_2, \dots, c_{n-r}$  are real (complex) constants.

Note that if the submatrix (10.2) has all zero elements, this implies that the rank of  $A'$  is equal to  $q - 1$ , the number of nonzero  $A'$  diagonal elements. By Theorem 10.1, none of the row or column operations used on the changing matrix  $A'$  can alter its rank, because these operations can be interpreted to be either premultiplication or postmultiplication of  $A'$  by another matrix with a nonzero determinant. So the rank of  $A'$  after an operation is also the rank of  $A'$  before the operation.

When there are two or more rows to the submatrix, it is risky to assume that all submatrix elements are zero. We cannot be confident in the solution vectors  $\mathbf{x}_i$  unless the ambiguity of the submatrix elements is resolved. For the special case where  $A$  is a diagonal matrix, no computation is required, and solving our problem becomes purely a matter of determining which diagonal elements are 0. It is clear then that with this problem we cannot avoid encountering the nonsolvable problem of deciding whether two numbers are equal.

**Nonsolvable Problem 10.1** Given an  $n$ -square real (complex) matrix  $A$ , with  $n > 1$  and  $\det A = 0$ , find to  $k$  decimal places the elements of a real (complex)  $n$ -vector  $\mathbf{x}$  of length 1, such that  $A\mathbf{x} = \mathbf{0}$ .

If we know in advance what the rank of  $A$  is, then the difficulty in interpreting  $A'$  disappears. If the matrix  $A'$  we obtain is such that the row index  $q - 1$  of the last nonzero diagonal element equals the known rank  $r$ , then all submatrix elements must be zero, and we find  $n - r$  solution vectors of length 1 by the method described. And if  $q - 1$  is less than  $r$ , then  $A'$  must be recomputed at a higher precision.

**Solvable Problem 10.2** Given an  $n$ -square real (complex) matrix  $A$  of known rank  $r < n$ , find to  $k$  decimal places the elements of  $n - r$  linearly independent real (complex)  $n$ -vectors  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n-r}$ , all of length 1, such that  $A\mathbf{x}_i = \mathbf{0}$  for all  $i$ .

If we know nothing about the  $n$ -square matrix  $A$  except that the rank  $r$  is less than  $n$ , that is,  $\det A = 0$ , then we obtain a solvable problem only by reducing our requirements:

**Solvable Problem 10.3** Given an  $n$ -square real (complex) matrix  $A$  with  $n > 1$  and  $\det A = 0$ , find to  $k$  decimal places the elements of a real (complex)  $n$ -vector  $\mathbf{x}$  of length 1 such that  $A\mathbf{x} = \mathbf{0}$ , or else find to  $k$  decimal places the elements of two or more real (complex)  $n$ -vectors  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_s$ , all of length 1, such that  $\text{len}(A\mathbf{x}_i) < 10^{-k}$  for all  $i$ .

The positive bound  $10^{-k}$  is arbitrary and may be changed to some other positive bound that varies with  $k$ , such as  $10^{-nk}$ . The procedure to be followed here depends on the form of  $A'$ . If the  $A'$  submatrix turns out to be 1-square, then, as described previously, we obtain a vector  $\mathbf{x}$  of length 1 satisfying the equation  $A\mathbf{x} = \mathbf{0}$ . If there is more than one row to the submatrix, then we assume that all submatrix elements are zero to get the vectors  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n-r}$ . These vectors must be tested with the original matrix  $A$  to determine whether  $\text{len}(A\mathbf{x}_i) < 10^{-k}$  for all  $i$ . If this result is not obtained, and also if the elements of  $\mathbf{x}$  or  $\mathbf{x}_i$  are not found to  $k$  decimal places, then we must recompute  $A'$  at an appropriate higher precision.

## 10.2 Eigenvalues and eigenvectors

Let  $A$  be a square matrix with real or complex elements. In many contexts the problem arises of finding a nonzero vector  $\mathbf{x}$ , such that the matrix equation

$$A\mathbf{x} = \lambda\mathbf{x}$$

holds for some number  $\lambda$ . The values possible for  $\lambda$  are called *eigenvalues*, and the vector  $\mathbf{x}$  is called an *eigenvector*. We can rewrite the matrix equation as

$$(A - \lambda I)\mathbf{x} = \mathbf{0}$$



and in this form it is clear that there can be a nonzero vector  $\mathbf{x}$  only if  $\det(A - \lambda I) = 0$ . The matrix  $A - \lambda I$  has the form

$$\begin{bmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda \end{bmatrix}$$

and  $\det(A - \lambda I)$  is a polynomial in  $\lambda$ :

$$\det(A - \lambda I) = (a_{11} - \lambda)(a_{22} - \lambda) \cdots (a_{nn} - \lambda) + \text{other determinant terms}$$

$$\det(A - \lambda I) = (-1)^n (\lambda^n + c_{n-1} \lambda^{n-1} + \cdots + c_1 \lambda + c_0) \quad (10.3)$$

The polynomial  $\lambda^n + c_{n-1} \lambda^{n-1} + \cdots + c_1 \lambda + c_0$  is called the *characteristic polynomial* of  $A$ . There are  $n$  eigenvalues,  $\lambda_1, \lambda_2, \dots, \lambda_n$ , corresponding to the  $n$  roots of the characteristic polynomial.

It is easy to accurately compute the coefficients of the characteristic polynomial, and in the next section practical methods of doing this are presented. Solvable Problem 8.1 now implies

**Solvable Problem 10.4** Find to  $k$  decimal places the  $n$  eigenvalues of a real (complex)  $n$ -square matrix  $A$ .

Consider next the problem of determining for each eigenvalue  $\lambda_i$  an associated eigenvector  $\mathbf{x}_i$ . Solving the matrix equation

$$(A - \lambda_i I) \mathbf{x}_i = \mathbf{0}$$

for  $\mathbf{x}_i$  is equivalent to solving a system of linear homogeneous equations for the components of  $\mathbf{x}_i$ , a problem considered in the preceding section.

It is helpful here to introduce the concept of *similarity*. One  $n$ -square matrix  $B$  is similar to another  $n$ -square matrix  $A$  if there is an  $n$ -square matrix  $P$  with an inverse  $P^{-1}$  such that

$$B = P^{-1} A P \quad (10.4)$$

The matrix  $P$  is said to *transform*  $A$  into  $B$ . From equation (10.4) it follows that

$$P B P^{-1} = A$$

Thus  $P^{-1}$  transforms  $B$  into  $A$ , so the relation of similarity is symmetric. Similar matrices have the same characteristic polynomial, because

$$\begin{aligned}
 \det(B - \lambda I) &= \det(P^{-1}AP - \lambda I) = \det[P^{-1}(A - \lambda I)P] \\
 &= \det P^{-1} \det(A - \lambda I) \det P = \det P^{-1} \det P \det(A - \lambda I) \\
 &= \det(P^{-1}P) \det(A - \lambda I) = \det I \det(A - \lambda I) \\
 &= \det(A - \lambda I)
 \end{aligned}$$

Thus similar matrices  $A$  and  $B$  have identical eigenvalues. Moreover, if  $\mathbf{x}_i$  is a  $\lambda_i$  eigenvector of  $B$ , then  $\mathbf{y}_i = P\mathbf{x}_i$  is a  $\lambda_i$  eigenvector of  $A$ , because from  $B\mathbf{x}_i = \lambda_i\mathbf{x}_i$  it follows that

$$\begin{aligned}
 BP^{-1}P\mathbf{x}_i &= \lambda_i\mathbf{x}_i \\
 PBP^{-1}P\mathbf{x}_i &= P(\lambda_i\mathbf{x}_i) = \lambda_iP\mathbf{x}_i \\
 A(P\mathbf{x}_i) &= \lambda_i(P\mathbf{x}_i) \\
 A\mathbf{y}_i &= \lambda_i\mathbf{y}_i
 \end{aligned}$$

Thus if we can find a matrix similar to  $A$  for which it is easy to find eigenvectors, then we also obtain eigenvectors for  $A$  by using the transformation matrix  $P$ .

Among the matrices similar to  $A$ , there exists a certain upper triangular matrix  $J$ , called the *Jordan normal form* of  $A$ , or more simply, the *Jordan form* of  $A$ , and defined in the theorem that follows. Let  $J_r(\lambda)$  be the  $r$ -square matrix

$$J_r(\lambda) = \begin{bmatrix} \lambda & 1 & & & \\ & \lambda & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda & 1 \\ & & & & \lambda \end{bmatrix}_{r \times r} \quad (10.5)$$

The matrix  $J_r(\lambda)$  is called a *Jordan block*. Its diagonal elements equal  $\lambda$ , and its elements just above the diagonal equal 1.

**Theorem 10.3** For any  $n$ -square real (complex) matrix  $A$ , there exists an  $n$ -square complex matrix  $P$ , having an inverse  $P^{-1}$ , such that  $P^{-1}AP = J$ , where  $J$ , the Jordan form of  $A$ , is an  $n$ -square matrix composed of Jordan blocks in the configuration

$$J = \begin{bmatrix} J_{m_1}(\lambda_1) & & & \\ & J_{m_2}(\lambda_2) & & \\ & & \ddots & \\ & & & J_{m_t}(\lambda_t) \end{bmatrix} \quad (10.6)$$

The Jordan form of  $A$  is unique up to rearrangement of the Jordan blocks. These blocks are all in diagonal position, that is, their diagonal elements are also diagonal elements of  $J$ .

The block eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_l$  are eigenvalues of  $J$ , and consequently of  $A$  too, but these block eigenvalues are not necessarily all distinct. If  $\lambda_j$  is a multiplicity  $m$  eigenvalue of  $A$ , then  $\lambda_j$  must appear  $m$  times as a diagonal element of  $J$ , and may be the diagonal element of more than one Jordan block. For example, if  $A$  is a 7-square matrix with eigenvalues  $\lambda_1, \lambda_2$ , and  $\lambda_3$ , with multiplicities 2, 2, and 3, respectively, then it is possible for  $J$  to take the form

$$\begin{bmatrix} \lambda_1 & 1 & & & & & \\ & \lambda_1 & & & & & \\ & & \lambda_2 & & & & \\ & & & \lambda_2 & & & \\ & & & & \lambda_3 & 1 & \\ & & & & & \lambda_3 & 1 \\ & & & & & & \lambda_3 \end{bmatrix}$$

with diagonal Jordan blocks  $J_2(\lambda_1)$ ,  $J_1(\lambda_2)$ ,  $J_1(\lambda_2)$ , and  $J_3(\lambda_3)$ .

It is sometimes difficult to compute the Jordan form. For instance, suppose we have

$$A = \begin{bmatrix} a & b \\ 0 & a \end{bmatrix}$$

and  $b \neq 0$ . The Jordan form is either

$$\begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} a & 1 \\ 0 & a \end{bmatrix}$$

depending on whether  $b$  is equal to zero. (If  $b \neq 0$ , the matrix  $\begin{bmatrix} 1 & 0 \\ 0 & b^{-1} \end{bmatrix}$ , with inverse  $\begin{bmatrix} 1 & 0 \\ 0 & b \end{bmatrix}$ , transforms  $A$  into  $\begin{bmatrix} a & 1 \\ 0 & a \end{bmatrix}$ , which becomes  $A$ 's Jordan form.) We see that again we encounter the nonsolvable problem of determining whether a number is zero.

**Nonsolvable Problem 10.5** For a real (complex)  $n$ -square matrix  $A$ , with  $n > 1$ , determine to  $k$  decimal places the elements of the Jordan form of  $A$ .

We consider the case of a rational matrix  $A$  later in this chapter, in Section 10.6.

It is not the diagonal elements of  $J$  that are difficult to determine, because these are eigenvalues and can be found to any number of correct decimal places. It is the 1's and 0's above the diagonal that are troublesome. The Jordan form,

though sometimes difficult to compute, is helpful in understanding eigenvalue and eigenvector problems.

It is simple to determine the eigenvectors of a Jordan form  $J$ . The eigenvector equation for a Jordan block  $J_r(\lambda)$  is  $(J_r(\lambda) - \lambda I)\mathbf{x} = \mathbf{0}$ , where the vector  $\mathbf{x}$  has  $r$  components, and the matrix  $J_r(\lambda) - \lambda I$  is

$$\begin{bmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ & & & & 0 \end{bmatrix}_{r \times r} \quad (10.7)$$

By working from the next to last linear equation backward, the solution  $\mathbf{x}$  to  $[J_r(\lambda) - \lambda I]\mathbf{x} = \mathbf{0}$  is easily seen to be any multiple of

$$\mathbf{x}^{(r)} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

It is also helpful to note that for  $\lambda' \neq \lambda$ , the equation  $[J_r(\lambda) - \lambda' I]\mathbf{x} = \mathbf{0}$  has only the trivial solution, because the upper triangular matrix  $[J_r(\lambda) - \lambda' I]$  has nonzero diagonal elements, so its determinant is nonzero.

For each Jordan block  $J_r(\lambda)$  appearing in  $J$  we obtain an eigenvector  $\mathbf{x}$  by adding zero components to  $\mathbf{x}^{(r)}$  in the obvious way to create an  $n$ -vector with a single 1 component. If there are  $t$  Jordan blocks in the  $J$  matrix, we obtain in this way  $t$  linearly independent eigenvectors  $\mathbf{x}_1, \dots, \mathbf{x}_t$ , one vector associated with each block. If  $\lambda_o$  is a particular eigenvalue of  $J$ , it is easy to see that the equation  $(J - \lambda_o I)\mathbf{x} = \mathbf{0}$  implies that any eigenvector associated with  $\lambda_o$  must be a linear combination of those eigenvectors associated with the individual  $\lambda_o$  blocks. This is because the Jordan blocks that are not  $\lambda_o$  blocks all have nonzero diagonal elements, forcing corresponding components of  $\mathbf{x}$  to be 0.

When we compute eigenvalues to a certain number of correct decimal places, we can assign apparent multiplicities to them. An apparent multiplicity of 1 is always the correct multiplicity and the eigenvalue is *simple*. Suppose  $\lambda_o$  is a simple eigenvalue of  $A$ . Then  $\lambda_o$  is also a simple eigenvalue of  $J$ , and this implies that the matrix  $J - \lambda_o I$  has rank  $n - 1$ . The rank of the matrix  $A - \lambda_o I$  is also  $n - 1$ , for we have

$$A - \lambda_o I = PJP^{-1} - \lambda_o I = P(J - \lambda_o I)P^{-1}$$

and then Theorem 10.1 implies that the ranks of  $A - \lambda_o I$  and  $J - \lambda_o I$  are the same. Solvable Problem 10.2 now implies

**Solvable Problem 10.6** For a simple eigenvalue  $\lambda_o$  of a real (complex)  $n$ -square matrix  $A$ , find to  $k$  decimal places the elements of an associated eigenvector  $\mathbf{x}$  of length 1.

Suppose now that  $\lambda_o$  is an apparent multiple eigenvalue of  $A$ . Here we are uncertain whether our computed multiplicity is correct. Examining the Jordan form, we see that if there are  $q$  Jordan blocks associated with  $\lambda_o$ , then the rank of  $J - \lambda_o I$  is  $n - q$ , and there are  $q$  linearly independent eigenvectors associated with  $\lambda_o$ . The ranks of  $A - \lambda_o I$  and  $J - \lambda_o I$  are the same, and if we were certain what this rank was, according to Solvable Problem 10.2, we could determine the correct number of linearly independent  $A$  eigenvectors for  $\lambda_o$ . But we can not be certain of a computed rank unless it turns out to equal  $n - 1$ . Solvable Problem 10.3 is helpful in this situation.

**Solvable Problem 10.7** For an apparent multiple eigenvalue  $\lambda_o$  of a real (complex)  $n$ -square matrix  $A$ , find to  $k$  decimal places the elements of a single associated eigenvector  $\mathbf{x}$  of length 1, or else find to  $k$  decimal places the elements of two or more vectors  $\mathbf{x}_1, \dots, \mathbf{x}_q$ , each of length 1 and satisfying the inequality  $\text{len}(A\mathbf{x}_i - \lambda_o\mathbf{x}_i) < 10^{-k}$  for all  $i$ .

The vectors  $\mathbf{x}_1, \dots, \mathbf{x}_q$  may be called “apparent eigenvectors” if it is understood that higher precision computation may show that some of these vectors are not eigenvectors.

We now turn to practical methods for handling the solvable problems of this section.

### 10.3 Companion matrices and Vandermonde matrices

The method for finding eigenvalues and eigenvectors that is presented in the next section makes use of the two matrix types treated here. The first type answers the following question: How can one construct a square matrix that has a specific characteristic polynomial? Suppose the polynomial is

$$P(\lambda) = \lambda^n + c_{n-1}\lambda^{n-1} + \dots + c_1\lambda + c_0$$

One solution to this puzzle is the  $n$ -square *companion matrix*  $C$  having the form:

$$C = \begin{bmatrix} 0 & 0 & \dots & 0 & -c_0 \\ 1 & 0 & \dots & 0 & -c_1 \\ 0 & 1 & \dots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -c_{n-1} \end{bmatrix}$$

A companion matrix has all elements zero except for the elements just below the diagonal, which equal 1, and the elements in the last column, which equal the coefficients of the polynomial times  $-1$ .

To show that

$$\det(C - \lambda I) = (-1)^n (\lambda^n + c_{n-1} \lambda^{n-1} + \dots + c_1 \lambda + c_0) \quad (10.8)$$

compute the determinant of

$$C - \lambda I = \begin{bmatrix} -\lambda & 0 & \dots & 0 & -c_0 \\ 1 & -\lambda & \dots & 0 & -c_1 \\ 0 & 1 & \dots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -\lambda - c_{n-1} \end{bmatrix}$$

by a cofactor expansion along the last column. Recall that the cofactor of a matrix element in row  $i$  and column  $j$  is  $(-1)^{i+j}$  times the determinant of the submatrix obtained by crossing out the  $i$ th row and  $j$ th column of the matrix.

The cofactor of the element  $-c_0$  is  $(-1)^{n+1}$  times the determinant of an upper triangular matrix with all diagonal elements equal to 1, so  $-c_0$  times its cofactor is  $(-1)^n c_0$ . The cofactor of the element  $-\lambda - c_{n-1}$  is  $(-1)^{n+n} = +1$  times the determinant of an  $(n-1)$ -square lower triangular matrix with all diagonal elements equal to  $-\lambda$ , so  $-\lambda - c_{n-1}$  times its cofactor is  $(-1)^n (\lambda^n + c_{n-1} \lambda^{n-1})$ . This verifies that the coefficients shown in equation (10.8) are correct for the leading two terms of the polynomial and the constant term. The cofactor of any other element in the last column, such as  $-c_j$ , is  $(-1)^{n+j+1}$  times the determinant of a matrix with the structure

$$\begin{bmatrix} B_1 & O \\ O & B_2 \end{bmatrix}$$

The  $j$ -square matrix  $B_1$  is lower triangular with diagonal elements equal to  $-\lambda$ , and the  $(n-1-j)$ -square matrix  $B_2$  is upper triangular with diagonal elements equal to 1. This leads to a contribution to  $\det(C - \lambda I)$  of  $(-c_j)(-1)^{n+j+1}(-\lambda)^j = c_j(-1)^{n+2j+2}\lambda^j = (-1)^n c_j \lambda^j$ , in agreement with equation (10.8).

If  $C$ 's characteristic polynomial has  $n$  distinct roots  $\lambda_1, \lambda_2, \dots, \lambda_n$ , then by Theorem 10.3 the matrix  $C$  can be transformed into a diagonal matrix with these roots along the diagonal. To find the transforming matrix, consider the matrix

$$V = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_n & \lambda_n^2 & \dots & \lambda_n^{n-1} \end{bmatrix} \quad (10.9)$$

A matrix of this form is called a *Vandermonde matrix*. If we carry through the matrix multiplication  $VC$ , and use in the last column of the product the identity

$$-c_0 - c_1\lambda_i - \cdots - c_{n-1}\lambda_i^{n-1} = \lambda_i^n$$

which holds for  $i = 1, 2, \dots, n$ , we find

$$VC = \begin{bmatrix} \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^n \\ \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^n \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_n & \lambda_n^2 & \cdots & \lambda_n^n \end{bmatrix} \\ = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \begin{bmatrix} 1 & \lambda_1 & \cdots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \cdots & \lambda_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_n & \cdots & \lambda_n^{n-1} \end{bmatrix}$$

Thus if the diagonal matrix is assigned the letter  $D$ , then we have  $VC = DV$ , and if  $V$  has an inverse  $V^{-1}$ , then  $VCV^{-1} = D$ , so  $V^{-1}$  is the transforming matrix.

We show by an indirect argument that if all the roots  $\lambda_i$  are distinct, then  $\det V \neq 0$ , so  $V$  has an inverse. For an  $n$ -square matrix  $A$ , if there are no nonzero  $n$ -vectors  $\mathbf{x}$  satisfying the matrix equation  $A\mathbf{x} = \mathbf{0}$ , this implies  $\det A \neq 0$ . Suppose then that the  $n$ -vector

$$\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}$$

satisfies the matrix equation  $V\mathbf{d} = \mathbf{0}$ . We have then

$$d_1 + d_2\lambda_i + d_3\lambda_i^2 + \cdots + d_n\lambda_i^{n-1} = 0, \quad \text{for } i = 1, 2, \dots, n$$

Thus the degree  $n - 1$  polynomial  $D(\lambda)$ , given by

$$D(\lambda) = d_1 + d_2\lambda + d_3\lambda^2 + \cdots + d_n\lambda^{n-1}$$

has  $n$  distinct roots, which is impossible.

Suppose there are multiple roots in the factorization of the characteristic polynomial:

$$P(\lambda) = (\lambda - \lambda_1)^{m_1}(\lambda - \lambda_2)^{m_2} \cdots (\lambda - \lambda_r)^{m_r}$$

Now the Vandermonde matrix takes a more general form. A block of  $m_i$  rows of  $V$  are assigned to  $\lambda_i$  with the structure

$$\begin{bmatrix} 0 & 0 & 0 & 0 & \dots & 1 & \dots & \binom{n-1}{m_i-1} \lambda_i^{n-m_i} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & 1 & \binom{3}{2} \lambda_i & \dots & \binom{m_i-1}{2} \lambda_i^{m_i-3} & \dots & \binom{n-1}{2} \lambda_i^{n-3} \\ 0 & 1 & \binom{2}{1} \lambda_i & \binom{3}{1} \lambda_i^2 & \dots & \binom{m_i-1}{1} \lambda_i^{m_i-2} & \dots & \binom{n-1}{1} \lambda_i^{n-2} \\ 1 & \lambda_i & \lambda_i^2 & \lambda_i^3 & \dots & \lambda_i^{m_i-1} & \dots & \lambda_i^{n-1} \end{bmatrix}$$

If we let  $V_i$  denote this  $m_i \times n$  matrix, then we find that the matrix equation  $V_i C = J_{m_i}(\lambda_i) V_i$  holds, where  $J_{m_i}(\lambda_i)$  is an  $m_i$ -square Jordan block. To see this, first carry through the matrix multiplication  $V_i C$  to get:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & \dots & \binom{m_i}{m_i-1} \lambda_i & \dots & \binom{n}{m_i-1} \lambda_i^{n-m_i+1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 1 & \binom{3}{2} \lambda_i & \binom{4}{2} \lambda_i^2 & \dots & \binom{m_i}{2} \lambda_i^{m_i-2} & \dots & \binom{n}{2} \lambda_i^{n-2} \\ 1 & \binom{2}{1} \lambda_i & \binom{3}{1} \lambda_i^2 & \binom{4}{1} \lambda_i^3 & \dots & \binom{m_i}{1} \lambda_i^{m_i-1} & \dots & \binom{n}{1} \lambda_i^{n-1} \\ \lambda_i & \lambda_i^2 & \lambda_i^3 & \lambda_i^4 & \dots & \lambda_i^{m_i} & \dots & \lambda_i^n \end{bmatrix}$$

To obtain the last column of this matrix, we need one additional equation. We know that  $\lambda_i$  satisfies the polynomial equation

$$-(c_0 + c_1 \lambda + c_2 \lambda^2 + \dots + c_{n-1} \lambda^{n-1}) = \lambda^n$$

Because  $\lambda_i$  has multiplicity  $m_i$ , it also satisfies the equations that can be obtained from this one by taking derivatives with respect to  $\lambda$ , of any order between 1 and  $m_i - 1$ . If we take the  $j$ -th derivative and then divide by  $j!$ , we obtain our needed equation

$$-\left[ c_j + c_{j+1} \binom{j+1}{j} \lambda + \dots + c_{n-1} \binom{n-1}{j} \lambda^{n-1-j} \right] = \binom{n}{j} \lambda^{n-j}$$

A matrix identical to  $V_i C$  is obtained if we premultiply  $V_i$  by the Jordan block  $J_{m_i}(\lambda_i)$ . Here we need to make use of the binomial identity

$$\binom{s}{j} = \binom{s-1}{j-1} + \binom{s-1}{j}$$

which is valid when the integer  $s$  is larger than the positive integer  $j$ .



Thus if the characteristic equation of a companion matrix has eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_l$  with respective multiplicities  $m_1, m_2, \dots, m_l$ , and we form a Vandermonde matrix  $V$  having an appropriate sized block for each distinct eigenvalue, we have the matrix equation  $VC = JV$ , where  $J$  is the Jordan form for  $C$ . The argument used before to show  $\det V \neq 0$  can be repeated for the more general Vandermonde matrix to show that it too has a nonzero determinant. Thus  $V$  has an inverse  $V^{-1}$ , so we have  $VCV^{-1} = J$ , and  $C$  is transformed to Jordan form by  $V^{-1}$ . The Jordan form of a companion matrix is restricted, because an eigenvalue  $\lambda_i$  can have only one associated Jordan block.

## 10.4 Finding eigenvalues and eigenvectors by Danilevsky's method

Given an  $n$ -square problem matrix  $A$ , the method of Danilevsky can be used to find an  $n$ -square matrix  $Q$  that transforms  $A$  into a companion matrix  $C$ , that is,  $Q^{-1}AQ = C$ . If we are successful in finding  $Q$  and  $C$ , then the last column of  $C$  has the coefficients of the common characteristic polynomial of  $C$  and  $A$ . Knowing the characteristic polynomial,  $A$ 's eigenvalues can be obtained as the polynomial's distinct roots, with each eigenvalue assigned an apparent multiplicity. This allows us to construct the matrices  $V$  and  $V^{-1}$  and to transform  $C$  into Jordan form  $J$ . Because  $Q^{-1}AQ = C$  and  $VCV^{-1} = J$ , if we set  $P$  equal to  $QV^{-1}$ , then  $P^{-1}AP = J$ . The eigenvector of  $J$  associated with an eigenvalue  $\lambda_i$  is a vector  $\mathbf{x}_i$  with all zero components except for a single 1 component aligned with the starting row of the Jordan block associated with  $\lambda_i$ . The corresponding eigenvector  $\mathbf{y}_i$  of  $A$  is  $P\mathbf{x}_i$ , as was shown in Section 10.2. Thus the eigenvectors of  $A$  are certain column vectors of  $P$ .

The matrices  $Q$  and  $C$  are computed in stages. That is, a series of transformations are made, gradually bringing  $A$  to companion form, one column at a time. We use  $A_i$  to designate the form of  $A$  after it has been transformed so that the columns to the left of column  $i$  are in companion form. In general we have

$$A_i = \begin{bmatrix} 0 & & & & a'_{1i} & \cdots & a'_{1n} \\ 1 & & & & a'_{2i} & \cdots & a'_{2n} \\ & 1 & & & a'_{3i} & \cdots & a'_{3n} \\ & & \ddots & & \vdots & \ddots & \vdots \\ & & & 1 & a'_{ii} & \cdots & a'_{in} \\ & & & & a'_{i+1,i} & \cdots & a'_{i+1,n} \\ & & & & \vdots & \ddots & \vdots \\ & & & & a'_{ni} & \cdots & a'_{nn} \end{bmatrix}$$









the various companion submatrices either are 0 or test  $\doteq 0$ . Because some of these elements may not be exact zeros, if we find correctly ranged eigenvalue approximations to the various companion submatrices, using their easily obtained characteristic polynomials, we cannot presume these eigenvalues are correctly ranged for  $C$ . Nevertheless for each companion matrix  $C_i$ , eigenvalue approximations can be found with computed apparent multiplicities, using the methods of Chapter 8, and so a Vandermonde matrix  $V_i$  for  $C_i$  can be constructed. We form the matrix

$$V = \begin{bmatrix} V_1 & & & \\ & V_2 & & \\ & & \ddots & \\ & & & V_s \end{bmatrix}$$

to correspond to the structure of  $C$  shown in (10.12). Using this matrix and its inverse  $V^{-1}$ , we can transform  $C$  to "approximate" Jordan form  $J'$ . Thus in all cases we can construct a matrix  $P$  such that  $P^{-1}AP = J'$ .

## 10.5 Error bounds for Danilevsky's method

The Jordan form  $J'$  achieved by Danilevsky's method is approximate, in the sense that off-diagonal  $J'$  elements may not be exact 0s or exact 1s. Therefore the eigenvalues of  $J'$  and the various columns of the transforming matrix  $P$  that are taken as eigenvectors must be assigned error bounds, to be added to the automatic computational error bounds. We consider here two types of approximate Jordan forms  $J'$ , one type that is diagonal with no off-diagonal 1s, and the more general Jordan form that has off-diagonal 1s.

For diagonal Jordan forms, we use a result due to Gershgorin that bounds eigenvalue error.

**Theorem 10.4** Let  $B$  be any  $n$ -square matrix with real or complex elements. In the complex plane define the  $n$  disks

$$|z - b_{ii}| \leq r_i \quad i = 1, 2, \dots, n \quad (10.13)$$

where

$$r_i = \sum_{\substack{j=1 \\ j \neq i}}^n |b_{ij}|$$

These  $n$  disks may overlap. In general, they form  $s$  disjoint, connected sets, where  $s = n$  only if the disks do not overlap. Every eigenvalue of  $B$  lies in one of these connected sets, and each connected set consisting of  $t$  disks contains  $t$  eigenvalues.

To prove the theorem, let  $\lambda$  be any eigenvalue of  $B$ , and let  $\mathbf{x}$  be an associated eigenvector, adjusted via multiplication by a constant so that the component  $x_{i_0}$  of largest magnitude equals 1. All other components of  $\mathbf{x}$  then have magnitude  $\leq 1$ . The matrix equation  $\lambda \mathbf{x} = B\mathbf{x}$  may be converted to the following element equations:

$$(\lambda - b_{ii})x_i = \sum_{\substack{j=1 \\ j \neq i}}^n b_{ij}x_j \quad i = 1, 2, \dots, n \quad (10.14)$$

Taking absolute values, we have

$$|\lambda - b_{ii}||x_i| = \left| \sum_{\substack{j=1 \\ j \neq i}}^n b_{ij}x_j \right| \leq \sum_{\substack{j=1 \\ j \neq i}}^n |b_{ij}||x_j| \leq \sum_{\substack{j=1 \\ j \neq i}}^n |b_{ij}| \cdot 1 = r_i \quad i = 1, 2, \dots, n$$

Because  $x_{i_0} = 1$ , for equation  $i_0$  we have

$$|\lambda - b_{i_0, i_0}| \leq r_{i_0}$$

Thus  $\lambda$  lies in one of the disks, and hence in one of the  $s$  connected sets.

To prove the other part of the theorem, about the number of eigenvalues in a connected set of disks, consider the matrix

$$B(u) = (1 - u)J + uB$$

where  $J$  is a diagonal matrix with the same diagonal elements as  $B$ . As the real parameter  $u$  varies from 0 to 1, the matrix  $B(u)$  varies from  $J$  to  $B$ . When  $u = 0$ , the disks are points and all parts of the theorem are correct. As  $u$  varies toward 1, the disk radii grow, and the sets of connected disks vary in the number of component disks. The eigenvalues move continuously in the complex plane and therefore cannot jump from one set of connected disks to another. Consequently, for any value of  $u$ , the number of disks in a connected set and the number of associated eigenvalues always match, and this includes the case  $u = 1$ .

We consider next how to bound the error of a supposed eigenvector. For a diagonal matrix, the eigenvector associated with any eigenvalue may be taken as a vector with a single component equal to 1, all other components being 0. If we claim this vector as an eigenvector approximation for  $B$ , which is not necessarily diagonal, then the zero eigenvector components must be assigned some error. The matrix  $B$  may be represented as a sum of two parts, a diagonal matrix  $J$  and a discrepancy matrix  $E$ :

$$B = J + E = \begin{bmatrix} b_{11} & & & \\ & b_{22} & & \\ & & \ddots & \\ & & & b_{nn} \end{bmatrix} + \begin{bmatrix} 0 & b_{12} & \dots & b_{1n} \\ b_{21} & 0 & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & 0 \end{bmatrix}$$

The radius  $r_q$  equals the sum of the absolute values of the  $E$  matrix elements that are in row  $q$ , and bounds the error of the  $J$  diagonal element  $b_{qq}$  if the disk corresponding to this element is disjoint from the other disks. Let us assume this to be the case. Then we may set  $\lambda_q = b_{qq} \oplus r_q$  to obtain a correctly ranged eigenvalue. Let  $\mathbf{x}_q$  be the  $B$  eigenvector associated with  $\lambda_q$  with its  $q$ th component set equal to 1. The other components of  $\mathbf{x}_q$  cannot have a magnitude larger than 1. Otherwise, if  $x_{i_o}$  is the component of  $\mathbf{x}_q$  that is largest in magnitude, we can divide all the vector's components by  $x_{i_o}$  to make the  $i_o$  component 1, repeat the steps of the proof of the preceding theorem and obtain  $|\lambda_q - b_{i_o, i_o}| \leq r_{i_o}$ , so  $\lambda_q$  lies in the  $b_{i_o, i_o}$  disk, contradicting our assumption that the  $b_{qq}$  disk was disjoint.

Now we can bound the magnitude of any component  $x_i$  of  $\mathbf{x}_q$  different from  $x_q$ . Using equation (10.14) for  $i \neq q$ , after taking absolute values, we have

$$|\lambda_q - b_{ii}| |x_i| \leq \sum_{\substack{j=1 \\ j \neq i}}^n |b_{ij}| |x_k| \leq \sum_{\substack{j=1 \\ j \neq i}}^n |b_{ij}| \cdot 1 = r_i$$

Hence

$$|x_i| \leq \frac{r_i}{|\lambda_q - b_{ii}|} = \rho_i$$

and we may take  $x_i$  as  $0 \oplus \rho_i$  to obtain correctly ranged values for all the components of our eigenvector  $\mathbf{x}_q$ . The eigenvector for the problem matrix  $A$  now can be computed in the usual way as  $P\mathbf{x}_q$ .

Suppose now that the  $b_{qq}$  disk is not disjoint from the other disks, but is part of a disjoint composite formed by  $t$  disks. In this case we take  $\lambda_q$  as  $b_{qq} \oplus R$  where  $R = 2 \sum r_i$ , the sum being over all indices  $i$  such that the  $b_{ii}$  disk is part of the composite. Now  $\lambda_q$  is taken as an eigenvalue of apparent multiplicity  $t$ . In accordance with Solvable Problem 10.3,  $t$  apparent eigenvectors can be defined, using vectors  $\mathbf{x}_i$  with all zero components except for a single 1 component in positions appropriate to the set of overlapping disks. An apparent eigenvector  $\mathbf{x}_i$  is not necessarily an eigenvector, and the only claim made for  $\mathbf{x}_i$  is that  $\text{len}(B\mathbf{x}_i - \lambda_q \mathbf{x}_i)$  is within a prescribed bound. Thus  $\mathbf{x}_i$  does not have to have an additional error bound assigned, and we obtain the apparent eigenvectors for the problem matrix  $A$  in the usual way as columns of  $P$ .

Now we consider the second, more general, Jordan form, where off-diagonal 1s occur. Again we represent the matrix  $B$  as a sum of two matrices, a Jordan form matrix  $J$  and a discrepancy matrix  $E$ . An  $ij$  element of  $J$  outside the Jordan block domains is 0, and a corresponding element of  $E$  is  $b_{ij}$ . The plan for elements of  $J$  and  $E$  that are in a Jordan block domain, can be illustrated by



showing just the elements of the first Jordan block. We presume this first block is  $m$ -square and show first the  $J$  elements and then the  $E$  elements.

$$\begin{bmatrix} b_{11} & 1 & & & & & \\ & b_{11} & 1 & & & & \\ & & \ddots & \ddots & & & \\ & & & \ddots & \ddots & & \\ & & & & \ddots & & \\ & & & & & b_{11} & 1 \\ & & & & & & b_{11} \end{bmatrix} \tag{10.15}$$

$$\begin{bmatrix} 0 & b_{12} - 1 & b_{13} & \dots & b_{1,m-1} & b_{1m} \\ b_{21} & b_{22} - b_{11} & b_{23} - 1 & \dots & b_{2,m-1} & b_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ b_{m-1,1} & b_{m-1,2} & b_{m-1,3} & \dots & b_{m-1,m-1} - b_{11} & b_{m-1,m} - 1 \\ b_{m1} & b_{m2} & b_{m3} & \dots & b_{m,m-1} & b_{mm} - b_{11} \end{bmatrix}$$

As before let  $r_i$  equal the sum of the absolute values of the matrix  $E$  elements in row  $i$ . This time, to keep our error bounds simple in form, we use only the quantity  $r$  defined as the maximum  $r_j$  value over all the rows of  $B$ . Suppose  $\mathbf{x}$  is an eigenvector of  $B$  with the component of largest magnitude being one of the components  $x_1, \dots, x_m$  associated with our first block. Because our block is the first block, we designate the  $\mathbf{x}$  eigenvalue to be  $\lambda_1$ . Suppose also that our eigenvector is multiplied by a constant to make the component of largest magnitude equal 1. The element equations are

$$\begin{aligned} (\lambda_1 - b_{11})x_1 &= x_2 + \sum_{j=1}^n e_{1j}x_j \\ (\lambda_1 - b_{11})x_2 &= x_3 + \sum_{j=1}^n e_{2j}x_j \\ &\vdots \\ (\lambda_1 - b_{11})x_{m-1} &= x_m + \sum_{j=1}^n e_{m-1,j}x_j \\ (\lambda_1 - b_{11})x_m &= \sum_{j=1}^n e_{mj}x_j \end{aligned} \tag{10.16}$$

Here the quantities  $e_{ij}$  are the elements of the discrepancy matrix  $E$ . After we take absolute values, the first equation becomes

$$|\lambda_1 - b_{11}| |x_1| \leq |x_2| + \sum_{j=1}^n |e_{1j}| |x_j| \leq |x_2| + \sum_{j=1}^n |e_{1j}| \cdot 1 \leq |x_2| + r$$

and, in general, we have

$$\begin{aligned} |\lambda_1 - b_{11}| |x_j| &\leq |x_{j+1}| + r & j = 1, 2, \dots, m-1 \\ |\lambda_1 - b_{11}| |x_m| &\leq r & j = m \end{aligned} \tag{10.17}$$

Suppose the  $\mathbf{x}$  component that equals 1 is  $x_1$ . Multiply the first inequality of (10.17) by  $|\lambda_1 - b_{11}|^{m-1}$ , the second by  $|\lambda_1 - b_{11}|^{m-2}$ , and so on down to the last inequality, which is multiplied by 1. We obtain the inequalities

$$\begin{aligned} |\lambda_1 - b_{11}|^m |x_1| &\leq |\lambda_1 - b_{11}|^{m-1} |x_2| + |\lambda_1 - b_{11}|^{m-1} r \\ |\lambda_1 - b_{11}|^{m-1} |x_2| &\leq |\lambda_1 - b_{11}|^{m-2} |x_3| + |\lambda_1 - b_{11}|^{m-2} r \\ &\vdots \\ |\lambda_1 - b_{11}|^2 |x_{m-1}| &\leq |\lambda_1 - b_{11}| |x_m| + |\lambda_1 - b_{11}| r \\ |\lambda_1 - b_{11}| |x_m| &\leq r \end{aligned}$$

If we add all these inequalities and cancel any identical terms on both sides of the inequality, we obtain the relation

$$|\lambda_1 - b_{11}|^m \leq r |\lambda_1 - b_{11}|^{m-1} + \dots + r |\lambda_1 - b_{11}| + r$$

or

$$|\lambda_1 - b_{11}|^m - r |\lambda_1 - b_{11}|^{m-1} - \dots - r |\lambda_1 - b_{11}| - r \leq 0 \tag{10.18}$$

The polynomial

$$z^m - rz^{m-1} - \dots - rz - r$$

is positive if  $z$  is real and exceeds the solitary positive root  $R$  of the polynomial (see Theorem 8.2). If we take  $z = 1$ , the polynomial equals  $1 - mr$ . We assume now and in the remainder of this section, that  $r$  is sufficiently small so that  $r < \frac{1}{m}$ , making  $1 - mr$  positive. (If this should not be the case, we increase precision and repeat the Danilevsky procedure.) So the inequality (10.18) implies  $|\lambda_1 - b_{11}| \leq R$ . The positive root  $R$  of our polynomial is  $< 1$ , and so we have

$$\begin{aligned} R &= (rR^{m-1} + rR^{m-2} + \dots + rR + r)^{\frac{1}{m}} \\ &< (r1^{m-1} + r1^{m-2} + \dots + r1 + r)^{\frac{1}{m}} = (mr)^{\frac{1}{m}} \end{aligned}$$

We obtain the easily calculated bound

$$|\lambda_1 - b_{11}| < (mr)^{\frac{1}{m}} \quad (10.19)$$

If we suppose that the largest modulus component of our eigenvector is not  $x_1$  but some other component  $x_j$  associated with the first Jordan block, we obtain even sharper bounds on  $|\lambda_1 - b_{11}|$ . Instead of the inequality (10.18), we obtain by a similar method the inequality

$$|\lambda_1 - b_{11}|^{m-j+1} - r|\lambda_1 - b_{11}|^{m-j} - \dots - r|\lambda_1 - b_{11}| - r \leq 0$$

which leads to

$$|\lambda_1 - b_{11}| < [((m-j+1)r)^{\frac{1}{m-j+1}}] < (mr)^{\frac{1}{m-j+1}} < (mr)^{\frac{1}{m}}$$

Thus the inequality (10.19) always can be used, and it defines a disk in the complex plane, namely  $|z - b_{11}| \leq (mr)^{1/m}$ . Similarly, any other Jordan block of  $B$  has a bounding disk assigned. Suppose a certain number of these disks overlap, forming a connected set, and let  $t$  equal the sum of the row sizes of the Jordan blocks corresponding to these disks. In the same way as was shown in the proof of the Gershgorin theorem, this set must contain  $t$  eigenvalues of  $B$ , counting multiplicities.

The procedure to be followed when the matrix  $B$  approximates a general Jordan normal form is similar to the procedure for the diagonal case. For the first Jordan block a correctly ranged eigenvalue is  $\lambda_1 = b_{11} \oplus (mr)^{1/m}$ , and similarly for the other blocks. If a disk for a Jordan block is discrete, we try to obtain eigenvector components, and if the disk is part of a composite figure, we supply apparent eigenvectors.

We consider now the problem of determining eigenvector components for a Jordan block when the bounding disk is discrete. Again we assume the eigenvector has been multiplied by a constant so that its component of largest magnitude equals 1. To illustrate the procedure, we assume the eigenvector is associated with the eigenvalue  $\lambda_1$ , the first block eigenvalue. By similar reasoning as in the proof of the Gershgorin Theorem, the eigenvector's 1 component must be associated with the first block, that is, it must be one of the components  $x_1, x_2, \dots, x_m$ . If the bound obtained by (10.19) is  $< 1/2$ , then it cannot be the case that the 1 component of our eigenvector is not  $x_1$ , but some other component  $x_j$  associated with the first Jordan block. If that were the case, we would have the following contradiction. The element equation (10.16) for row  $j-1$  can be rewritten as

$$x_j = (\lambda - b_{11})x_{j-1} - \sum_{i=1}^n e_{j-1,i}x_i$$

After taking absolute values, setting  $|x_j|$  equal to 1, and using the inequality  $|x_i| \leq 1$  for all other components, we obtain the relation

$$1 \leq |\lambda - b_{11}| + r$$

which is impossible because both  $|\lambda - b_{11}|$  and  $r$  are less than  $1/2$ .

Thus if  $r$  is small enough, the only eigenvectors possible for the matrix  $B$  are eigenvectors whose component of largest modulus,  $x_q$ , made equal to 1, is such that  $q$  equals the starting row index of a Jordan block. As in the diagonal  $B$  case we need to obtain correctly ranged values for the other components of such a vector. Continuing with the case of the  $\lambda_1$  eigenvector, the first  $m - 1$  equations of the set (10.16) can be rewritten as

$$x_j = (\lambda_1 - b_{11})x_{j-1} - \sum_{i=1}^n e_{j-1,i}x_i \quad j = 2, 3, \dots, m$$

Taking absolute values leads to the inequalities

$$|x_j| \leq |\lambda_1 - b_{11}| + r \quad j = 2, 3, \dots, m$$

for the components other than  $x_1$  that are associated with the first block, that is, they have indices matching the row indices of the first block. All these components may be taken as  $0 \oplus q$ , where  $q = [(mr)^{\frac{1}{m}} + r]$ .

We consider next the components associated with some other block, that starts in row  $s$ . These components we will denote by  $x'_1, x'_2, \dots, x'_{m'}$ , the block being  $m'$ -square. Taking absolute values of the equations that apply to these  $m'$  components, we obtain the relations

$$\begin{aligned} |\lambda_1 - b_{ss}| |x'_1| &\leq |x'_2| + r \\ |\lambda_1 - b_{ss}| |x'_2| &\leq |x'_3| + r \\ &\vdots \qquad \qquad \qquad \vdots \\ |\lambda_1 - b_{ss}| |x'_{m'-1}| &\leq |x'_{m'}| + r \\ |\lambda_1 - b_{ss}| |x'_{m'}| &\leq r \end{aligned}$$

Working backward through these inequalities, we obtain the following bounds for  $x'_i$ :

$$\begin{aligned} |x'_{m'}| &\leq \frac{r}{|\lambda_1 - b_{ss}|} \\ |x'_{m'-1}| &\leq \frac{r}{|\lambda_1 - b_{ss}|} + \frac{r}{|\lambda_1 - b_{ss}|^2} \\ &\vdots \qquad \qquad \qquad \vdots \\ |x'_1| &\leq \frac{r}{|\lambda_1 - b_{ss}|} + \frac{r}{|\lambda_1 - b_{ss}|^2} + \dots + \frac{r}{|\lambda_1 - b_{ss}|^{m'}} \end{aligned}$$

Each component  $x'_i$  then may be assigned the value  $0 \oplus q'_i$ , with  $q'_i$  chosen according to the inequalities shown. With correctly ranged components for the  $B$  eigenvector  $\mathbf{x}$ , we can form the  $A$  eigenvector  $\mathbf{y}$  by means of the equation  $\mathbf{y} = P\mathbf{x}$ , and convert it to a vector of length 1 in the usual way. As always we need to redo the entire computation at an appropriate higher precision if the required number of correct decimal places for the eigenvector components is not obtained.

## 10.6 Rational matrices

Finding eigenvalues and eigenvectors for a rational matrix  $A$  generally requires that one leave the field of rational numbers. Nevertheless, when  $A$  is rational, this problem can be treated without difficulty.

**Solvable Problem 10.8** For a rational  $n$ -square matrix  $A$ , determine all eigenvalues to  $k$  decimal places, and determine the multiplicity of each eigenvalue. For each eigenvalue  $\lambda_0$ , find to  $k$  decimal places the elements of a complete set of linearly independent eigenvectors  $\mathbf{x}_1, \dots, \mathbf{x}_q$ , each of length 1.

More details about the computation procedure for a rational matrix are given in the next section.

The problem of finding the Jordan form  $J$  of  $A$  also poses no difficulty.

**Solvable Problem 10.9** Given an  $n$ -square rational matrix  $A$ , determine to  $k$  decimal places the elements of the Jordan form  $J$  of  $A$ , and to  $k$  decimal places the elements of a complex matrix  $P$ , such that  $P^{-1}AP = J$ .

With a rational matrix  $A$ , the Danilevsky procedure of Section 10.4 can be used to find a rational matrix  $Q$  that transforms  $A$  to companion form  $C$ , where the rational matrix  $C$  is composed of a number of companion submatrices,  $C_1, C_2, \dots, C_s$ , all in diagonal position. The Danilevsky procedure converts to rational computation without difficulty and yields a rational companion matrix  $C$  with exact elements. From the rational characteristic polynomial of one of the companion submatrices  $C_i$ , one can form eigenvalue approximations  $\lambda_1^{(i)}, \lambda_2^{(i)}, \dots, \lambda_{q_i}^{(i)}$ , with respective correct multiplicities  $m_1^{(i)}, m_2^{(i)}, \dots, m_{q_i}^{(i)}$ . Using these eigenvalues, one can construct the appropriate Vandermonde matrix  $V$ , and then obtain the matrix  $P = QV^{-1}$ , which transforms  $A$  to the Jordan form  $J$ . Thus the Jordan form obtained is the correct Jordan form, and should its elements not be obtained to the required number of decimal places, one need only increase precision appropriately, and repeat the transformation of the exact companion matrix  $C$  to Jordan form.

## 10.7 The demo programs `eigen`, `c_eigen`, and `r_eigen`

For a square matrix  $A$  with real elements, the program `eigen` finds eigenvalues, eigenvectors, and, possibly, apparent eigenvectors. After the elements of the matrix  $A$  have been entered by the user, and the number of correct decimal places specified, the program `eigen` enters a loop wherein all computations are performed.

Within the loop, first the Danilevsky procedure is used to transform  $A$  into companion matrix form  $C$ . The roots of the real polynomials defined by the companion submatrices are obtained, and used to construct a Vandermonde matrix  $V$ , allowing the companion matrix  $C$  to be transformed by  $V^{-1}$  into approximate Jordan normal form  $J'$ . The polynomial roots are obtained with their ranges set to zero, because the methods described in Section 10.5 are used to obtain from  $J'$  correctly ranged eigenvalues. Using exact root approximations to construct  $V$  allows  $V^{-1}$  to have elements with smaller ranges.

If eigenvectors are required, for each ranged eigenvalue of  $J'$  either a correctly ranged eigenvector is obtained, or a set of apparent eigenvectors is obtained. Apparent eigenvectors for  $J'$  have a single component equal to 1 and all others are 0. From a correctly ranged  $J'$  eigenvector  $\mathbf{x}$ , a correctly ranged  $A$  eigenvector  $\mathbf{y}$  is obtained using the relation  $\mathbf{y} = P\mathbf{x}$ , where  $P$  is the matrix that transforms  $A$  into  $J'$ . Apparent eigenvectors for  $A$  are formed simply by using appropriate columns of  $P$ . Eigenvectors and apparent eigenvectors are converted into vectors of length 1 by multiplying them by the reciprocal of their length. Finally, eigenvalues and eigenvector components are tested to determine whether enough correct decimal places have been obtained. If so, the loop is exited and the results displayed. Otherwise, precision is increased appropriately and another cycle through the computation loop ensues.

The program `c_eigen` is similar to `eigen` except for minor changes to accommodate a complex problem matrix  $A$  instead of a real one.

The program `r_eigen` for a rational problem matrix  $A$  has a computation loop that differs from the `eigen` loop. The companion matrix  $C$  is obtained with rational arithmetic, and the computation loop starts after  $C$  is obtained.

The Jordan blocks corresponding to a  $C$  submatrix are correctly sized, because correct multiplicities for the roots of a rational polynomial are obtained. After ranges are assigned to the leading diagonal elements of the Jordan blocks to define eigenvalues, eigenvalues associated with different  $C$  submatrices are compared with each other. If there is no overlap, then one correctly ranged eigenvector per Jordan block is obtained by the method described for `eigen`.

If there are  $k$  eigenvalues associated with different  $C$  submatrices that overlap, and if these eigenvalues truly are equal, then  $k$  linearly independent eigenvectors can be obtained. The eigenvalue multiplicity of this common eigenvalue is the sum of the individual multiplicities. An overlap can be verified by multiplying the polynomials of all the  $C$  submatrices together to obtain the  $A$  characteristic

polynomial, and determining the true multiplicity of its roots. If this multiplicity determination shows that the overlap is in error, precision is increased and the loop computations are repeated. Otherwise, the normal testing of eigenvectors ensues.

## Software Exercises H

These exercises concern the two demo programs `eigen` and `r_eigen`.

1. Call `eigen` and find to 10 decimal places the eigenvalues and eigenvectors of the following matrix.

$$\begin{bmatrix} 2 & 4 & 4 \\ 0 & 3 & 1 \\ 0 & 1 & 3 \end{bmatrix}$$

View the `eigen` print file to see the results in detail. Note the “Apparent multiplicity” label for eigenvalues. For one of the eigenvalues, two apparent eigenvectors are obtained instead of two eigenvectors. However, the problem matrix is a rational matrix, and for such a matrix, computing apparent eigenvectors is not necessary.

2. Call `r_eigen`, enter the problem matrix of the preceding exercise one more time, and view the `r_eigen` print file. Note that the multiplicity is not labeled as apparent, and three eigenvectors are obtained. Thus if all elements of a problem matrix are known to be rational, it is advantageous to use the program `r_eigen` instead of `eigen`.





with  $A$  the rational coefficient matrix of size  $m \times n$ , with  $\mathbf{x}$  an  $n$ -vector of unknowns, and with  $\mathbf{b}$  the rational  $m$ -vector of right-side constants.

A solution procedure is possible using the various row and column operations introduced in Chapter 9. First we apply the operations **Exchange Row**( $i, j$ ) and **Add Row Multiple**( $i, j, M$ ) to  $A$ , attempting to make zero all the elements of  $A$  below the diagonal. Each operation is applied to both  $A$  and  $\mathbf{b}$ , so that the new set of equations, defined by the changed  $A$  and  $\mathbf{b}$  arrays, is equivalent to the old set. As in Chapter 9, the changing elements of  $A$  and components of  $\mathbf{b}$  are denoted by the symbols  $a'_{ij}$  and  $b'_j$ , respectively, and the changed arrays are denoted by  $A'$  and  $\mathbf{b}'$ . When working on column  $j$  of  $A'$ , if we find  $a'_{jj} = 0$ , then we search column  $j$  for an element below  $a'_{jj}$  that is unequal to zero. If we find  $a'_{kj} \neq 0$ , then we perform the operation **Exchange Row**( $j, k$ ) to bring this element into the  $a'_{jj}$  position. After we have obtained a nonzero diagonal element  $a'_{jj}$ , we clear the elements in column  $j$  below  $a'_{jj}$ , with element  $a'_{kj}$  being cleared by the operation **Add Row Multiple**( $j, k, -\frac{a'_{kj}}{a'_{jj}}$ ).

If in column  $j$  the element  $a'_{jj}$  and the elements below it are zero, then we search the columns to the right of column  $j$ , one by one, attempting to find an element in row  $j$  or below that is nonzero. If we find  $a'_{pq} \neq 0$ , we perform the operation **Exchange Col**( $q, j$ ) to bring this element to the  $a'_{jj}$  position, and then we can proceed as before to bring column  $j$  to the proper form. The **Exchange Col**( $q, j$ ) operation of course cannot be performed on  $\mathbf{b}'$ . To keep the equations represented by our  $A'$  and  $\mathbf{b}'$  arrays equivalent to the initial equations, as explained in Section 10.1, we perform the **Exchange Col**( $q, j$ ) operation on an indicator array  $H$ , initially holding the integers 1 through  $n$  in natural order. The indicator  $H$  identifies  $\mathbf{x}'$ , the changing vector of unknowns. The components of  $\mathbf{x}'$  are  $x'_1, \dots, x'_n$ , with  $x'_i$  identical to  $x_{H_i}$ . After as many columns of  $A$  as possible have been brought to the desired form, the appearance of all the arrays is as follows:

$$\begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1r} & \cdots & a'_{1n} \\ & a'_{22} & \cdots & a'_{2r} & \cdots & a'_{2n} \\ & & \ddots & \vdots & & \vdots \\ & & & a'_{rr} & \cdots & a'_{rn} \\ & & & & & 0 \\ & & & & & \vdots \\ & & & & & 0 \end{bmatrix} \begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_r \\ \vdots \\ x'_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_r \\ b'_{r+1} \\ \vdots \\ b'_m \end{bmatrix} \quad (11.2)$$

If any constant  $b'_q$  in the list of constants  $b'_{r+1}, \dots, b'_m$  is unequal to zero, then there are no solutions to the equations, because we have found an equivalent set of equations with one equation being  $0 = b'_q$ .

If all the constants  $b'_{r+1}, \dots, b'_m$  are zero, we can obtain a representation of the solutions to the equations. The zero elements of  $A'$  below row  $r$  are no longer needed, and these are discarded. Similarly, all 0 components of  $\mathbf{b}'$  beyond

component  $r$  are discarded. The number of  $A'$  rows is now  $r$  and the number of  $\mathbf{b}'$  components is now  $r$ , but the vector  $\mathbf{x}'$  retains all its  $n$  components. Working from column  $r$  back toward column 1, we gradually form an identity submatrix in the first  $r$  columns of  $A'$ . When working with column  $j$ , we apply the operation **Multiply Row**( $j, 1/a'_{jj}$ ) to both  $A'$  and  $\mathbf{b}'$  to make  $a'_{jj}$  equal 1. Then we subtract appropriate multiples of row  $j$  from rows above to clear column  $j$  above the unit diagonal element. After these column operations are completed, the arrays have the form

$$[I \quad C']\mathbf{x}' = \mathbf{b}'$$

where the identity submatrix  $I$  is  $r$ -square and the submatrix  $C'$  is of size  $r \times (n - r)$ . It is convenient to split the  $n$ -vector  $\mathbf{x}'$  into two subvectors, an  $r$ -vector  $\mathbf{x}'_B$  and an  $(n - r)$ -vector  $\mathbf{x}'_N$ .

$$[I \quad C'] \begin{bmatrix} \mathbf{x}'_B \\ \mathbf{x}'_N \end{bmatrix} = \mathbf{b}' \quad (11.3)$$

The vector  $\mathbf{x}'_B$  is defined in terms of  $\mathbf{x}$  components by the first  $r$  components of the indicator  $H$ , and the vector  $\mathbf{x}'_N$  is defined by the remaining  $n - r$  components of  $H$ . Eq. (11.3) can also be expressed this way:

$$\mathbf{x}'_B + C'\mathbf{x}'_N = \mathbf{b}'$$

The solution is given by the matrix representation

$$\mathbf{x}' = \begin{bmatrix} \mathbf{x}'_B \\ \mathbf{x}'_N \end{bmatrix} = \begin{bmatrix} \mathbf{b}' - C'\mathbf{x}'_N \\ \mathbf{x}'_N \end{bmatrix} \quad (11.4)$$

where the components of the  $(n - r)$ -vector  $\mathbf{x}'_N$  can be set to any values we please.

The subscripts 'B' and 'N' stand for "basic" and "nonbasic", which is terminology from linear programming. If we set  $\mathbf{x}'_N$  to  $\mathbf{0}$ , we obtain, after undoing the scrambling of  $\mathbf{x}'$  components, the solution vector  $\mathbf{b}_0$ . Similarly, by setting  $\mathbf{x}'_N$  to various vectors with all components 0 except for a single component equal to 1, we obtain  $n - r$  linearly independent solution vectors, which, after similar unscrambling of components, can be represented as  $\mathbf{b}_0 + \mathbf{x}_1, \mathbf{b}_0 + \mathbf{x}_2, \dots, \mathbf{b}_0 + \mathbf{x}_{n-r}$ . Therefore any solution to the original set of equations has the form  $\mathbf{b}_0$  plus multiples of the vectors  $\mathbf{x}_i$ .

**Solvable Problem 11.1** Given a set of  $m$  linear equations in  $n$  unknowns, represented by the matrix equation  $A\mathbf{x} = \mathbf{b}$ , where  $A$  is a rational  $m \times n$  coefficient matrix,  $\mathbf{x}$  is the  $n$ -vector of unknowns, and  $\mathbf{b}$  is a rational  $m$ -vector of constants, decide whether the equations have a solution. If they have a solution, determine

the rank  $r$  of the coefficient matrix, and find a vector  $\mathbf{b}_0$  and  $n - r$  linearly independent rational vectors  $\mathbf{x}_1, \dots, \mathbf{x}_{n-r}$ , such that any real vector  $\mathbf{x}$  represents a solution to the equations only if it can be written in the form

$$\mathbf{x} = \mathbf{b}_0 + c_1 \mathbf{x}_1 + c_2 \mathbf{x}_2 + \dots + c_{n-r} \mathbf{x}_{n-r}$$

with real constants  $c_i$ .

## 11.2 A more efficient method for solving rational linear equations

The described method of solving rational linear equations uses row and column exchanges to bring the coefficient matrix  $A$  to the form  $[I \ C']$ . Exchanging the elements of two matrix rows or two matrix columns is time consuming, especially if the matrix is large. These exchange operations can be eliminated when the rows of the matrix  $A$ , viewed as vectors, are known to be linearly independent. In this case the solution procedure described in the preceding section cannot generate a contradictory equation of the form  $0 = b'_q$ . The equations are consistent and can be solved, because a zero  $A'$  row, as shown in Eq. (11.2), implies that the rows of  $A$  are linearly dependent.

To eliminate row and column exchanges of the  $m \times n$  coefficient matrix  $A$ , with only the two operations **Multiply Row**( $i, M$ ) and **Add Row Multiple**( $i, j, M$ ) now allowed on  $A$  and on the vector  $\mathbf{b}$  of right-hand constants, we need two indicator arrays,  $B$  and  $N$ . Both arrays have integer components.

The array  $B$  ("basic variable indicator") has  $m$  components that are used to locate the columns of the changing matrix  $A'$  that have been converted to columns of  $I$ , that is, columns having a single 1 element with all other elements being 0. Specifically,  $B_i = k$  if column  $k$  of  $A'$  is identical to the  $i$ th column of an  $m$ -square identity matrix  $I$ . The array  $B$  will also define the basic variables, because if  $B_i = k$ , then  $x_k$  is the  $i$ th basic variable. Now there is no longer a need for the indicator  $H$  defining  $\mathbf{x}'$ . The array  $N$  ("nonbasic variable indicator"), with  $n - m$  components, gives the positions of the other  $A'$  columns, and defines the nonbasic variables.

To solve  $A\mathbf{x} = \mathbf{b}$  when the rank of the  $m \times n$  coefficient matrix  $A$  is known to equal  $m$ , we can proceed as follows. We examine the rows of  $A$ , starting with the first row and proceeding in order to the last row. In each row we test the elements in sequential order, stopping at the first nonzero element. When examining row  $i$ , if we find  $a'_{ik} \neq 0$ , we convert the element to 1 by using the operation **Multiply Row**( $i, 1/a'_{ik}$ ) on  $A'$  and  $\mathbf{b}'$ . Next we use appropriate **Add Row Multiple**( $i, j, M$ ) operations on  $A'$  and  $\mathbf{b}'$  to clear the other elements of column  $k$ . Now with a column matching the  $i$ th identity matrix column, we set  $B_i = k$ . The  $I$  columns already found by processing the rows before row  $i$  are unaffected by the row  $i$  procedure. After all rows have been processed, the



In analogy with the quadrant concept of the cartesian plane, the points satisfying (11.8) are said to lie in the *first orthant* of the  $n$ -space. If in the inequalities (11.8) we prefix some of the variables with a minus sign, we define another orthant of the  $n$ -space, which has a total of  $2^n$  orthants.

We will assume that all the initial constants  $a_{ij}$ ,  $b_i$ , and  $f_i$  of a linear programming problem are rational constants, because the solution method presented, the *simplex method*, at various steps requires deciding whether certain quantities are positive, negative, or zero, and there is no difficulty making such decisions if we use rational arithmetic.

The first step in the solution procedure is to rewrite each inequality of the set (11.7) as an equation, introducing one extra variable to represent the difference between the two sides of the inequality:

$$\begin{array}{rcl}
 a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + x_{n+1} & = & b_1 \\
 a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + & x_{n+2} & = b_2 \\
 \vdots & \ddots & \vdots \\
 a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + & x_{n+m} & = b_m
 \end{array} \tag{11.9}$$

The introduced variables  $x_{n+1}, x_{n+2}, \dots, x_{n+m}$  are called *slack* variables, and like the original variables, they must be nonnegative, so we still have  $x_i \geq 0$  for all  $i$ . All these relations may be represented concisely as

$$[A \ I]\mathbf{x} = \mathbf{b} \quad \text{and} \quad \mathbf{x} \geq \mathbf{0} \tag{11.10}$$

where  $\mathbf{x}$  is an  $(n+m)$ -vector and  $\mathbf{b}$  is an  $m$ -vector.

The general solution to the matrix equation of line (11.10) can be found by the method described in the preceding section, but there is an easier route. Because of the presence of the  $m$ -square identity submatrix within the enlarged coefficient matrix  $[A \ I]$ , we need only set  $B$  to indicate the columns of this submatrix, and set  $N$  to indicate the other columns, and we are done. From now on we call this particular solution the “simple solution”. Each possible way of writing the solution using different specifications of the basic and nonbasic variables determines the same set of solution points within the  $(n+m)$ -dimensional space  $S$ . Because the nonbasic indicator  $N$  has  $(n+m) - m = n$  components, the set of solution points defines an  $n$ -dimensional subspace of  $S$ . Any point  $\mathbf{x}_0$  satisfying both relations of line (11.10), called a *feasible point*, lies in the intersection of the subspace with the first orthant of  $S$ .

The set of feasible points  $FP$ , if it is not empty or infinite in extent, may be thought of as a kind of generalized  $n$ -dimensional polyhedron lying in  $S$ . The set  $FP$  is convex, because, as we show next, if  $\mathbf{x}_1$  and  $\mathbf{x}_2$  are vectors defining two feasible points, all points on the line segment joining the two points are also feasible. The line segment is defined by  $\mathbf{x} = (1-t)\mathbf{x}_1 + t\mathbf{x}_2$ ,

$0 \leq t \leq 1$ . As  $t$  varies from 0 to 1, the point  $\mathbf{x}$  moves from  $\mathbf{x}_1$  to  $\mathbf{x}_2$ . We have  $[A \ I]\mathbf{x}_1 = \mathbf{b}$  and  $[A \ I]\mathbf{x}_2 = \mathbf{b}$ , along with  $\mathbf{x}_1 \geq \mathbf{0}$  and  $\mathbf{x}_2 \geq \mathbf{0}$ . Then we have also  $[A \ I](1-t)\mathbf{x}_1 = (1-t)\mathbf{b}$  and  $[A \ I]t\mathbf{x}_2 = t\mathbf{b}$ , along with  $(1-t)\mathbf{x}_1 \geq \mathbf{0}$  and  $t\mathbf{x}_2 \geq \mathbf{0}$ . Adding, we obtain  $[A \ I]\mathbf{x} = \mathbf{b}$  along with  $\mathbf{x} \geq \mathbf{0}$ , so all points on the line segment are feasible.

The gradient vector of the objective function  $f$ , pointing in the direction of maximum increase for  $f$ , is a constant vector. If in  $FP$  we move in a straight line making an acute angle with the gradient, the objective function increases, whereas if we move perpendicular to the gradient, it does not change. If we wish to find the objective function maximum in  $FP$ , by moving along a sequence of straight line segments, each segment should make an acute angle with the gradient, or at worst be perpendicular to it. The maximum will usually be found at a "corner point", that is, a vertex of  $FP$ . Occasionally there is no maximum because we can move as far as we please in  $FP$  along some line making an acute angle with the gradient. Here the set  $FP$  is infinite in extent and allows unrestricted motion along this line. Accordingly, except for the case where there is no finite maximum, the corner points of  $FP$  are the only points we need to test to find the maximum of the objective function. It is possible that an entire boundary line segment or higher dimensional face of  $FP$  consists of maximum points because the gradient is perpendicular to it, but in this case we also can find a corner point where the maximum is attained. The situation is similar when searching for the minimum of the objective function.

As shown in the preceding section, the  $m$  equations in  $n+m$  unknowns given by (11.9) have the simple solution, and there are other solutions obtainable by exchanging nonbasic and basic variables. The nonbasic components of  $\mathbf{x}$  may be set to any value, and the basic components of  $\mathbf{x}$  can then be determined. The components  $x_{B_1}, x_{B_2}, \dots, x_{B_m}$  of  $\mathbf{x}$  are called *basic* variables, and the components  $x_{N_1}, x_{N_2}, \dots, x_{N_n}$  are *nonbasic* variables. According to Eq. (11.5), for any solution representation, when the nonbasic components are set to 0, for all  $i$  we have  $x_{B_i} = b'_i$ , where  $b'_i$  is the current  $i$ th component of  $\mathbf{b}'$ . We will call the  $\mathbf{x}$  point determined this way the *null point* of the solution representation. If all components  $b'_i$  are nonnegative, then the null point is a corner point of the set of feasible points  $FP$ . This point satisfies all the constraints of (11.10), and is a corner point because if  $\mathbf{x}$  is to continue to define a feasible point, the nonbasic components, being zero, can change in only one direction, that is, become positive.

The simplex method of solving a linear programming problem starts with any solution representation whose null point is a corner point of  $FP$ , and then attempts to find another solution representation with its null point again a corner point, but with the objective function  $f$  showing an improved value at the new corner point. This process of  $f$  improvement continues until finally a corner point is found at which  $f$  is optimum.

If the components of the starting  $\mathbf{b}$  vector of (11.10) are all nonnegative, the null point of the simple solution representation is a corner point. When this

fortunate situation does not occur, the problem of locating a starting corner point is more difficult, and we deal with it later.

We assume now our linear programming problem requires us to maximize the objective function  $f$ . (One can find an  $f$  minimum by using the maximizing procedure on  $-f$ .) When we are ready to improve the objective function, it must be expressed entirely in terms of the nonbasic variables. Note that if our starting corner point was obtained by the simple solution representation, this is automatically the case. To make the process of eliminating basic variables from  $f$  easy, it is convenient before starting the procedure to rewrite  $f$  of line (11.6) as

$$f + f'_1x_1 + f'_2x_2 + \cdots + f'_nx_n = f_0 \quad (f'_i = -f_i)$$

and to carry the coefficients  $f'_i$  as an extra last row of the starting  $[A \ I]$  array. The coefficient  $f_0$  then becomes an extra last component of the  $\mathbf{b}$  vector. The enlarged  $[A \ I]$  array and the enlarged  $\mathbf{b}$  vector thus have this form:

$$D' = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{m,n} & 0 & 0 & \cdots & 1 \\ f'_1 & f'_2 & \cdots & f'_n & 0 & 0 & \cdots & 0 \end{bmatrix} \quad \mathbf{b}' = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \\ f_0 \end{bmatrix}$$

Suppose now the null point of the simple solution is a feasible point. With the  $f$  coefficients in the last row of the enlarged  $[A \ I]$  array, whenever a nonbasic and a basic component of  $\mathbf{x}$  are exchanged by the process of converting a column to an identity matrix column by clearing elements, the objective function element in the last row is cleared too. So the coefficients  $f'_j$  associated with nonbasic variables may be any value, but all coefficients  $f'_j$  associated with basic variables are zero. The value of the objective function at the null point equals the current value of  $f_0$  in the  $\mathbf{b}'$  vector.

We begin now the description of the simplex cycle at a general solution representation having the matrix form  $D'\mathbf{x} = \mathbf{b}'$ . The coefficients of the objective function that correspond to nonbasic variables, that is, the coefficients  $f'_{N_1}, f'_{N_2}, \dots, f'_{N_n}$ , are examined one by one in the order shown, and the examination stops at the first negative coefficient. Suppose  $f'_{N_q}$  is negative. The nonbasic variable  $x_{N_q}$  will be increased from its zero value to make  $f$  larger. As  $x_{N_q}$  increases, the basic variables also change. A basic variable  $x_{B_i}$  satisfies the following equation:

$$x_{B_i} + d'_{i,N_1}x_{N_1} + d'_{i,N_2}x_{N_2} + \cdots + d'_{i,N_n}x_{N_n} = b'_i$$

The basic variable  $x_{B_i}$  increases, decreases, or does not change, depending on whether  $d'_{i,N_q}$  is negative, positive, or zero, respectively. If  $d'_{i,N_q}$  is positive and  $x_{B_i}$  decreases, it must not decrease below zero; otherwise, the null point

of the new solution representation will be nonfeasible. The basic variable  $x_{B_i}$  becomes zero when  $x_{N_q}$  equals  $b'_i/d'_{i,N_q}$ . Thus the nonbasic variable  $x_{N_q}$  can be allowed to increase only to the minimum  $b'_i/d'_{i,N_q}$  value determined for those positive coefficients  $d'_{i,N_q}$  in column  $N_q$ . If there happens to be no positive coefficients in this column, then  $x_{N_q}$  can be allowed to increase indefinitely, and the linear programming problem is revealed as a problem without a finite maximum. Suppose a minimum of the  $b'_i/d'_{i,N_q}$  values occurs for  $i$  equal to  $r$ . The nonbasic variable  $x_{N_q}$  is increased to this value and becomes basic, and the basic variable  $x_{B_r}$  is decreased to zero and becomes nonbasic. We make these changes in our  $D'$  and  $\mathbf{b}'$  arrays in the following way. The operation **Multiply Row**( $r, \frac{1}{d'_{r,N_q}}$ ) on  $D'$  and  $\mathbf{b}'$  changes the  $d'_{r,N_q}$  element to 1, and afterwards a series of **Add Row Multiple** operations on  $D'$  and  $\mathbf{b}'$  clears column  $N_q$  of  $D'$  (and also clears  $f'_{N_q}$ ). Finally the exchange of roles of  $x_{B_r}$  and  $x_{N_q}$  is accomplished by exchanging the  $B_r$  and the  $N_q$  integers. We are now ready for the next simplex cycle.

When we can not find any negative  $f'_i$  coefficients associated with nonbasic variables for a solution representation, we have located a corner point where  $f$  is maximum, and the simplex process terminates. The maximum of  $f$  is the current  $f_0$  value, the last component of the enlarged  $\mathbf{b}'$  vector, and the point at which this maximum occurs is the null point for this final solution representation. The point is determined when the current nonbasic variables are set to 0, making the current basic variables equal the corresponding  $\mathbf{b}'$  components. If all the  $f'_i$  components associated with nonbasic variables are positive, no other feasible point shares this maximum  $f$  value, and the located corner point is identified as a unique maximum point.

## 11.4 Making the simplex process foolproof

In the simplex cycle, if  $b'_r$  is positive when the roles of  $x_{B_r}$  and  $x_{N_q}$  are exchanged, the new  $f_0$  coefficient will be larger than the preceding one, because the coefficient  $f'_{N_q}$  is negative, and a positive multiple of  $b'_r$  is added to  $f_0$  when  $f'_{N_q}$  is cleared. Because  $f_0$  is larger than any previous  $f_0$  value, the corner point determined by the new solution representation is different from any previous corner point. If  $b'_r$  is zero, the objective function's  $f_0$  value is unchanged. In this case, it is possible the new corner point is identical to a previous one, and there is a danger that we are in a simplex cycle loop. A number of ways have been proposed to eliminate the possibility of loops, and our demo programs use the method proposed by Dantzig, Orden, and Wolfe [3]. With this method we measure improvement of the objective function in a more general way, using vector comparison involving additional  $f$  coefficients instead of just  $f_0$  in the enlarged  $\mathbf{b}'$  vector. At the start of the simplex process, when the first corner point has been located, the  $D'$  columns associated with the  $m$  basic variables



are recorded. Let  $[1], [2], \dots, [m]$  be these columns. ( $[k]$  equals the initial  $B_k$  integer.) The vector  $\mathbf{f}'$  to be used in comparisons has  $m + 1$  components, which are:

$$f'_0, f'_{[1]}, \dots, f'_{[m]}$$

Here  $f'_{[i]}$  denotes the coefficient associated with column  $[i]$ . A similar vector  $\mathbf{d}'_i$  is defined for any row  $i$  of the  $D'$  array by taking its components to be:

$$b'_i, d'_{i,[1]}, \dots, d'_{i,[m]}$$

Now let  $\mathbf{u}$  and  $\mathbf{v}$  be two such vectors, and let  $\mathbf{0}$  be such a vector with all components zero. We count  $\mathbf{u} > \mathbf{v}$  if, comparing the components in order and ignoring components that are equal, the first larger component belongs to  $\mathbf{u}$ . If  $\mathbf{u} > \mathbf{v}$ , we will say that the vector  $\mathbf{u}$  is "lexicographically" greater than the vector  $\mathbf{v}$ . Thus  $(2, 2, 1, \dots, 3) > (2, 1, 3, \dots, 5)$ . Many rules for inequalities with real numbers have analogous rules for this vector relation. From  $\mathbf{u} > \mathbf{v}$ , it follows that  $\mathbf{u} - \mathbf{v} > \mathbf{0}$ . It also follows that  $a\mathbf{u} > a\mathbf{v}$ , where  $a$  is any positive number, and that  $\mathbf{u} + \mathbf{w} > \mathbf{v} + \mathbf{w}$  where  $\mathbf{w}$  is any vector. Thus if  $\mathbf{u} > \mathbf{0}$  and  $a > 0$ , then if  $\mathbf{w}$  is any vector, it follows that  $\mathbf{w} + a\mathbf{u} > \mathbf{w}$ . A final needed relation is that if  $\mathbf{u}_1 > \mathbf{v}_1$  and  $\mathbf{u}_2 > \mathbf{v}_2$ , it follows that  $\mathbf{u}_1 + \mathbf{u}_2 > \mathbf{v}_1 + \mathbf{v}_2$ .

In a simplex cycle, previously when  $f'_{N_q}$  was negative, we chose a row  $r$  such that  $b'_r/d'_{r,N_q}$  was the minimum of the quantities  $b'_i/d'_{i,N_q}$  having  $d'_{i,N_q}$  positive. However, there could be a tie for minimum, and we revise our selection system to make the choice unique. Now we choose row  $r$  if the vector  $\frac{1}{d'_{r,N_q}}\mathbf{d}'_r$  is the lexicographic minimum of all vectors  $\frac{1}{d'_{i,N_q}}\mathbf{d}'_i$  having  $d'_{i,N_q}$  positive.

At the beginning of the first simplex cycle, the last  $m$  components of the vectors  $\mathbf{d}'_i$  are associated with the initial basic variables, and accordingly are all 0 except for a single 1 component. Therefore subvectors composed of the last  $m$  components of the vectors  $\mathbf{d}'_i$  are linearly independent, implying that a comparison tie among the beginning vectors  $\frac{1}{d'_{i,N_q}}\mathbf{d}'_i$  is impossible. Also, we have the relation  $\mathbf{d}'_i > \mathbf{0}$  for all  $i$ . We show next that if these two conditions exist at the start of a simplex cycle, they exist at the end of the cycle, and so always hold. During a simplex cycle, the last  $m$  components of the new  $\mathbf{d}'_i$  vectors get generated from the corresponding components of the preceding  $\mathbf{d}'_i$  vectors, by a number of **Multiply Row and Add Row Multiple** operations. These operations do not affect the linear independence of the last  $m$  components, and so the new  $\mathbf{d}'_i$  vectors have the same property. Note also that when column  $N_q$  is cleared by adding multiples of row  $r$  to other rows, if  $d'_{i,N_q}$  is negative, a positive multiple of row  $r$  is added to row  $i$ . Because  $\mathbf{d}'_r > \mathbf{0}$ , for the new vector  $\mathbf{d}'_i$  we will have  $\mathbf{d}'_i > \mathbf{0}$  if this relation was true for the old vector  $\mathbf{d}'_i$ . If  $d'_{i,N_q}$  is positive, the multiple  $d'_{i,N_q}/d'_{r,N_q}$  of row  $r$  is subtracted from row  $i$ , but because  $\frac{1}{d'_{i,N_q}}\mathbf{d}'_i > \frac{1}{d'_{r,N_q}}\mathbf{d}'_r$ , implying

$\mathbf{d}'_i > \frac{d'_{i,N_q}}{d'_{r,N_q}} \mathbf{d}'_r$ , we again obtain  $\mathbf{d}'_i > \mathbf{0}$  for the new vector  $\mathbf{d}'_i$ . The multiplication of row  $r$  by the positive constant  $1/d'_{r,N_q}$  maintains the relation  $\mathbf{d}'_r > \mathbf{0}$ . Thus  $\mathbf{d}'_i > \mathbf{0}$  for all  $i$  at the conclusion of the simplex cycle.

Because  $\mathbf{d}'_r > \mathbf{0}$ , when a positive multiple of  $\mathbf{d}'_r$  is added to  $\mathbf{f}'$ , the new  $\mathbf{f}'$  vector is lexicographically greater than the old one. This implies that the new solution representation is different from any preceding solution representation, and simplex cycle loops now are impossible.

Now we address the problem of locating an initial corner point when the simple solution representation does not determine one. Assume that we have *any* solution representation, possibly the simple one, and that some of the components  $b'_i$  are negative, so our representation's null point is not feasible. The basic variable  $x_{B_i}$  is negative or nonnegative, according to whether the current  $b'_i$  component is negative or nonnegative, respectively. We collect the indices of all negative basic variables in a list  $L$ . So  $i$  is on the list  $L$  if  $x_{B_i}$  is negative. The general idea is to adapt the simplex procedure to the problem of finding the largest value of the linear function  $f$  equal to the sum of basic variables on the list  $L$ . The list  $L$  varies as we proceed, with  $i$  being dropped from  $L$  as soon as  $x_{B_i}$  becomes nonnegative. When the list is empty, our solution representation determines a feasible null point.

The varying linear function  $f$ , defined as the sum of all basic variables with indices currently on the list  $L$ , has, corresponding to a nonbasic variable  $x_{N_j}$ , a coefficient  $f'_{N_j}$  that equals  $\sum_{i \in L} d'_{i,N_j}$ . We want to increase  $f$ , so we test these coefficients to find a negative one, say  $f'_{N_q}$ . As before, we wish to make the nonbasic variable  $x_{N_q}$  positive to increase our temporary objective function. Here the increase is limited to the minimum of the quantities  $b'_i/d'_{i,N_q}$  for  $i$  not on  $L$  with  $d'_{i,N_q}$  positive, because we do not want any nonnegative basic variables to become negative. However, there may be no indices  $i$  satisfying the conditions given. In this case we increase  $x_{N_q}$  to the maximum of the group of *positive* quantities  $b'_i/d'_{i,N_q}$  having  $i$  on  $L$  and having  $d'_{i,N_q}$  negative, because then every corresponding negative basic variable in this group will become nonnegative, and nonnegative basic variables stay nonnegative. Once the bound on  $x_{N_q}$  is determined, the process of altering the  $D'$  and  $\mathbf{b}'$  arrays and exchanging the variable  $x_{N_q}$  with a basic variable is the same as before. After we have a new solution representation, the entire process is repeated, and it continues until the list  $L$  is empty and a feasible null point has been determined. If it ever happens that none of the  $f'_{N_j}$  coefficients are negative, the linear programming problem is revealed as one that has no feasible points.

The lexicographical vector comparison system for avoiding simplex cycle loops can be used when finding a starting feasible solution. Here, however, every time the objective function gets changed, the  $D'$  columns  $[1], \dots, [m]$  need to be changed to the current  $B_i$  values, to ensure that  $\mathbf{d}'_i > \mathbf{0}$  for all nonnegative basic variables.



The methods of linear programming can be used to obtain accurate bounds. First, we show some needed notation. The interval elements of  $A$  and  $\mathbf{b}$  can be expressed in either the endpoint notation or the midpoint-halfwidth notation. If for an interval element  $a_{ij}$  we use the notation  $a_{ij}^{(m)}$  for the element's midpoint and  $a_{ij}^{(w)}$  for its halfwidth, and similar notation for the interval element  $b_i$ , then for all  $i, j$

$$a_{ij} = \left\{ a_{ij}^{(m)} \pm a_{ij}^{(w)}, [a_{ij}^{(m)} - a_{ij}^{(w)}, a_{ij}^{(m)} + a_{ij}^{(w)}] \right\} \quad b_i = \left\{ b_i^{(m)} \pm b_i^{(w)}, [b_i^{(m)} - b_i^{(w)}, b_i^{(m)} + b_i^{(w)}] \right\}$$

We use  $A^{(m)}$  for the matrix of midpoints and  $A^{(w)}$  for the matrix of halfwidths, and similarly  $\mathbf{b}^{(m)}$  and  $\mathbf{b}^{(w)}$  for the midpoint and halfwidth vectors. Also, if  $C$  is a real, rational, or complex matrix, then  $|C|$  denotes the matrix of corresponding size with elements equal to  $|c_{ij}|$ . Similarly, if  $\mathbf{d}$  is a real, rational, or complex  $n$ -vector, then  $|\mathbf{d}|$  denotes the  $n$ -vector with components  $|d_i|$ . The following basic result, due to Oettli and Prager [8], allows the problem to be transformed into a problem of linear programming.

**Theorem 11.1** Let the real  $n$ -square matrix  $A$  and the real  $n$ -vector  $\mathbf{b}$  be defined by intervals. A necessary and sufficient condition for a real  $n$ -vector  $\mathbf{x}$  to specify a solution point to the matrix equation  $A\mathbf{x} = \mathbf{b}$  is

$$|A^{(m)}\mathbf{x} - \mathbf{b}^{(m)}| \leq A^{(w)}|\mathbf{x}| + \mathbf{b}^{(w)} \quad (11.13)$$

or, equivalently,

$$-A^{(w)}|\mathbf{x}| - \mathbf{b}^{(w)} \leq A^{(m)}\mathbf{x} - \mathbf{b}^{(m)} \leq A^{(w)}|\mathbf{x}| + \mathbf{b}^{(w)} \quad (11.14)$$

We show the necessity of the condition first. If the  $n$ -vector  $\mathbf{x}$  specifies a solution point, there is an  $n$ -square matrix  $\widehat{A}$  and an  $n$ -vector  $\widehat{\mathbf{b}}$  such that  $\widehat{A}\mathbf{x} = \widehat{\mathbf{b}}$ , with  $\widehat{A}$  and  $\widehat{\mathbf{b}}$  satisfying the relations

$$A^{(m)} - A^{(w)} \leq \widehat{A} \leq A^{(m)} + A^{(w)} \quad \text{and} \quad \mathbf{b}^{(m)} - \mathbf{b}^{(w)} \leq \widehat{\mathbf{b}} \leq \mathbf{b}^{(m)} + \mathbf{b}^{(w)}$$

which can be rewritten as

$$-A^{(w)} \leq \widehat{A} - A^{(m)} \leq A^{(w)} \quad \text{and} \quad -\mathbf{b}^{(w)} \leq \widehat{\mathbf{b}} - \mathbf{b}^{(m)} \leq \mathbf{b}^{(w)}$$

which is equivalent to

$$|\widehat{A} - A^{(m)}| \leq A^{(w)} \quad \text{and} \quad |\widehat{\mathbf{b}} - \mathbf{b}^{(m)}| \leq \mathbf{b}^{(w)}$$

We have

$$\begin{aligned} |A^{(m)}\mathbf{x} - \mathbf{b}^{(m)}| &= |A^{(m)}\mathbf{x} - \mathbf{b}^{(m)} - (\widehat{A}\mathbf{x} - \widehat{\mathbf{b}})| = |(A^{(m)} - \widehat{A})\mathbf{x} - (\mathbf{b}^{(m)} - \widehat{\mathbf{b}})| \\ &\leq |A^{(m)} - \widehat{A}||\mathbf{x}| + |\mathbf{b}^{(m)} - \widehat{\mathbf{b}}| \leq A^{(w)}|\mathbf{x}| + \mathbf{b}^{(w)} \end{aligned}$$

To show the sufficiency of the condition, suppose the matrix relation (11.13) holds for a certain  $n$ -vector  $\mathbf{x}$ . We must show that there is an acceptable matrix  $\widehat{A}$  and an acceptable vector  $\widehat{\mathbf{b}}$  such that  $\widehat{A}\mathbf{x} = \widehat{\mathbf{b}}$ . Take  $\widehat{A}$  equal to  $A^{(m)} + A^{(\delta)}$  with  $A^{(\delta)}$  to be determined, and take  $\widehat{\mathbf{b}}$  equal to  $\mathbf{b}^{(m)} + \mathbf{b}^{(\delta)}$ , with  $\mathbf{b}^{(\delta)}$  likewise to be determined. The matrix  $A^{(\delta)}$  and the vector  $\mathbf{b}^{(\delta)}$  must satisfy

$$|A^{(\delta)}| \leq A^{(w)} \quad \text{and} \quad |\mathbf{b}^{(\delta)}| \leq \mathbf{b}^{(w)} \quad (11.15)$$

The matrix equation  $\widehat{A}\mathbf{x} = \widehat{\mathbf{b}}$  implies

$$\begin{aligned} (A^{(m)} + A^{(\delta)})\mathbf{x} &= \mathbf{b}^{(m)} + \mathbf{b}^{(\delta)} \\ A^{(\delta)}\mathbf{x} - \mathbf{b}^{(\delta)} &= -(\mathbf{b}^{(m)} - A^{(m)}\mathbf{x}) \\ &= -\mathbf{y} \end{aligned}$$

Here we have taken the vector  $\mathbf{y}$  equal to the determinable vector  $\mathbf{b}^{(m)} - A^{(m)}\mathbf{x}$ . The  $i$ th component of  $\mathbf{y}$  then satisfies the equation

$$\sum_{j=1}^n a_{ij}^{(\delta)} x_j - b_i^{(\delta)} = -y_i \quad (11.16)$$

The elements of  $A^{(\delta)}$  and  $\mathbf{b}^{(\delta)}$  which appear for component  $i$  do not appear for any other component, so we are free to set them as we please to satisfy the  $-y_i$  equation. We take  $a_{ij}^{(\delta)} = t_i \cdot a_{ij}^{(w)} \operatorname{sgn} x_j$  and take  $b_i^{(\delta)} = -t_i \cdot b_i^{(w)}$ , where the parameter  $t_i$ , one component of the parameter vector  $\mathbf{t}$ , is to be determined. We must choose each component  $t_i$  so that the inequality  $|t_i| \leq 1$  holds, because the relations (11.15) must be satisfied. Equation (11.16) now has the form

$$t_i \left( \sum_{j=1}^n a_{ij}^{(w)} |x_j| + b_i^{(w)} \right) = -y_i$$

If  $t_i$  is set to 1, the left side of this equation is a number  $\rho_i$ , that is,

$$\sum_{j=1}^n a_{ij}^{(w)} |x_j| + b_i^{(w)} = \rho_i$$

where  $\rho_i$  is nonnegative. Because  $\mathbf{y} = \mathbf{b}^{(m)} - A^{(m)}\mathbf{x}$ , the inequality (11.14) implies  $-\rho_i \leq y_i \leq \rho_i$ , so  $|y_i| \leq \rho_i$ . Thus if  $\rho_i = 0$ , then  $y_i = 0$  too, and  $t_i$  is set equal

to 0. If  $\rho_i$  is nonzero,  $t_i$  is taken equal to  $-y_i/\rho_i$  and the needed condition  $|t_i| \leq 1$  holds. Thus an appropriate matrix  $\widehat{A}$  and vector  $\widehat{\mathbf{b}}$  must exist such that relation (11.13) holds, and the theorem is proved.

We can use this result to obtain a method of treating the preceding solvable problem. Given a specific problem  $\mathbf{Ax} = \mathbf{b}$ , one can always obtain one solution point by solving the set of equations defined by the matrix equation  $A^{(m)}\mathbf{x} = \mathbf{b}^{(m)}$ . Suppose that when we do this, we obtain a point  $\mathbf{x}$  that lies in a particular orthant  $Q$ . To find the relations that apply in  $Q$ , let  $D_Q$  be an  $n$ -square diagonal matrix with its  $i$ th diagonal element chosen to be +1 or -1 depending on whether  $x_i$  is positive or negative, respectively, in the orthant. Then we have both equations  $D_Q\mathbf{x} = |\mathbf{x}|$  and  $\mathbf{x} = D_Q|\mathbf{x}|$ . The relations (11.14) now may be written as

$$-A^{(w)}|\mathbf{x}| - \mathbf{b}^{(w)} \leq A^{(m)}D_Q|\mathbf{x}| - \mathbf{b}^{(m)} \leq A^{(w)}|\mathbf{x}| + \mathbf{b}^{(w)}$$

or, equivalently, as

$$\begin{aligned} (A^{(m)}D_Q - A^{(w)})|\mathbf{x}| &\leq \mathbf{b}^{(m)} + \mathbf{b}^{(w)} \\ (-A^{(m)}D_Q - A^{(w)})|\mathbf{x}| &\leq -\mathbf{b}^{(m)} + \mathbf{b}^{(w)} \end{aligned} \tag{11.17}$$

To find the largest value of a component  $|x_i|$  of  $|\mathbf{x}|$  for all solution vectors  $\mathbf{x}$  in the orthant  $Q$ , we can solve the linear programming problem with the  $2n$  linear constraints represented by the two vector inequalities just given, and the nonnegativity constraints

$$|x_k| \geq 0, \quad k = 1, 2, \dots, n$$

for the maximum of the objective function  $f$  equal to  $|x_i|$ . The smallest value for  $|x_i|$  is obtained by solving the identical linear programming problem, except that we find the minimum of the objective function.

Suppose, for instance, that the problem to be solved is

$$\begin{aligned} [5, 6]x_1 + [4, 5]x_2 &= [2, 3] \\ [3, 4]x_1 + [0, 1]x_2 &= [1, 2] \end{aligned} \tag{11.18}$$

Here the various matrices and vectors of the theorem are:

$$A^{(m)} = \begin{bmatrix} \frac{11}{2} & \frac{9}{2} \\ 7 & \frac{1}{2} \end{bmatrix}, \quad A^{(w)} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}, \quad \mathbf{b}^{(m)} = \begin{bmatrix} \frac{5}{2} \\ 3 \end{bmatrix}, \quad \mathbf{b}^{(w)} = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}$$

The solution to  $A^{(m)}\mathbf{x} = \mathbf{b}^{(m)}$  yields a point in the first orthant, or rather first quadrant, the dimension of  $\mathbf{x}$  being 2. To find the maximum and minimum of

the variables  $x_1$  and  $x_2$  in the first quadrant, we set  $D_Q$  to  $I$ . The two vector relations (11.17), when written out for each component, are:

$$5|x_1| + 4|x_2| \leq 3$$

$$3|x_1| + 0|x_2| \leq 2$$

$$-6|x_1| - 5|x_2| \leq -2$$

$$-4|x_1| - 1|x_2| \leq -1$$

We can use the program `linpro` to find the maximum and minimum of  $|x_1|$  and  $|x_2|$  in quadrant 1. Here we use this program four times, always entering the inequalities just displayed but replacing  $|x_1|$  by  $x_1$  and  $|x_2|$  by  $x_2$ , finding the maximum and the minimum of the objective function  $x_1$ , and doing the same for the objective function  $x_2$ . The interval obtained this way for  $x_1$  is  $[\frac{1}{11}, \frac{3}{5}]$ , and the interval obtained for  $x_2$  is  $[0, \frac{7}{11}]$ .

Because the minimum of  $x_2$  is 0, the boundary of quadrant 1 was reached, so we must repeat the entire process for the adjoining quadrant 4, that is, the quadrant obtained by changing the second diagonal element of  $D_Q$  from  $+1$  to  $-1$ . In this quadrant the vector inequalities (11.17), when written out, are:

$$5|x_1| + 5|x_2| \leq 3$$

$$3|x_1| + 1|x_2| \leq 2$$

$$-6|x_1| - 4|x_2| \leq -2$$

$$-4|x_1| - 0|x_2| \leq -1$$

As before we can use the program `linpro` to find the extremes for the variables in quadrant 4, but this time when we enter the displayed inequalities,  $|x_1|$  is replaced by  $x_1$  and  $|x_2|$  is replaced by  $-x_2$ , with the objective function being varied just as before. The interval obtained for  $x_1$  is  $[\frac{1}{3}, 1]$  and the interval obtained for  $x_2$  is  $[0, 1]$ , which, taking into account the effect of  $D_Q$ , is actually the  $x_2$  interval  $[-1, 0]$ . Combining results from the two quadrants, the interval for  $x_1$  is  $[\frac{1}{11}, 1]$ , and the interval for  $x_2$  is  $[-1, \frac{7}{11}]$ .

## 11.6 Solving linear interval equations via linear programming

The method used to solve the simple interval problem of the preceding section can be made more efficient and applicable to an arbitrary problem of this type. It is necessary to assume that solution points to a set of linear interval equations lie in just one orthant to make the problem conform to the requirements of linear

programming. The points  $\mathbf{x}$  that satisfy such equations frequently do lie in just one orthant, but, as shown by the preceding example, it is also possible that solution points stretch over several orthants. In each orthant the set of solution points is convex, because a solution point is also a feasible point for a set of inequalities, and as shown in Section 11.3, the feasible points form a convex set. When there are solution points in more than one orthant, these convex orthant parts join together to form a solution set that may not be convex.

To simplify our notation, we now replace  $|x_i|$  everywhere by  $\hat{x}_i$ . We assign one additional slack variable to each individual component inequality of the set (11.17) to convert it into an equation. So the number of variables increases from  $n$  to  $3n$ , and the constraint equations then have this matrix representation:

$$\begin{bmatrix} A^{(m)}D_Q - A^{(w)} & I & O \\ -A^{(m)}D_Q - A^{(w)} & O & I \end{bmatrix} \hat{\mathbf{x}} = \begin{bmatrix} \mathbf{b}^{(m)} + \mathbf{b}^{(w)} \\ -\mathbf{b}^{(m)} + \mathbf{b}^{(w)} \end{bmatrix} \quad (11.19)$$

Here  $\hat{\mathbf{x}}$  is a  $3n$ -vector with components  $\hat{x}_i$ .

It is necessary to maintain a current upper and a current lower bound for each of the  $n$  variables of  $\mathbf{x}$ . These bounds are updated as various orthants of the  $n$ -space are investigated.

A task queue listing the orthants to be investigated is also needed, with each entry having the diagonal elements of the matrix  $D_Q$  defining the orthant. Initially the queue has a single entry for the starting orthant, with the list entry identifying the orthant as the starting one. This starting orthant is found by solving the equation  $A^{(m)}\mathbf{x} = \mathbf{b}^{(m)}$  for  $\mathbf{x}$ , and using the vector to determine the orthant  $Q$  and the corresponding matrix  $D_Q$ . If a component  $x_j$  of this initial solution vector is zero, the corresponding  $j$ th diagonal element of  $D_Q$  is arbitrarily taken as  $+1$ .

Because (11.19) has  $2n$  equations, the number of basic variables is  $2n$ , and the number of nonbasic variables is  $n$ . The first step with a task queue entry is to generate the corresponding coefficient matrix of size  $2n \times 3n$ . For the starting orthant, an initial feasible solution representation is found by the computation described in Section 11.4. For any other orthant, a list of the  $2n$  basic variables that permit a feasible solution appears on the queue entry, and this information is used to generate the initial solution representation. It will become clear shortly how this information has been obtained.

For a nonstarting orthant  $Q$ , it is easy to generate the starting feasible solution representation, using the provided list of basic variables, which identifies the coefficient matrix columns that must be converted into identity submatrix columns. After these preliminary operations are done, a feasible solution representation is available for the simplex process, and the indicators  $B$  and  $N$  are set accordingly.

We now employ notation used before in Section 11.3 to describe the simplex procedure. The changing coefficient matrix is  $D'$ , with elements  $d'_{i,j}$ , but now  $D'$  does not have an added row to hold objective function coefficients. The changing



right-hand vector  $\mathbf{b}'$ , with components  $b'_i$ , also does not have an added component for the objective function.

The objective functions  $f = \widehat{x}_i$ ,  $i = 1, 2, \dots, n$  are maximized in sequence, and then these same objective functions are minimized in sequence. Each objective function problem begins operations with the final solution representation and associated corner point obtained by the preceding problem. This is not true, of course, for the maximization of the first objective function, which begins with the specified solution representation, or in the case of the starting orthant, the computed solution representation.

We consider the maximization problems first. The objective function  $f = \widehat{x}_i$  is defined by a  $D'$  row and a  $\mathbf{b}'$  component whenever  $\widehat{x}_i$  is a basic variable. Suppose  $\widehat{x}_i$  is basic variable  $\widehat{x}_{B_j}$ . The row  $j$  elements of  $D'$  and component  $j$  of  $\mathbf{b}'$  imply then the equation

$$\widehat{x}_{B_j} + d'_{j,N_1}\widehat{x}_{N_1} + d'_{j,N_2}\widehat{x}_{N_2} + \dots + d'_{j,N_n}\widehat{x}_{N_n} = b'_j$$

Just as with an objective function, we look in row  $j$  of  $D'$  for an element  $d'_{j,N_k}$  that is negative. If none is found, we are done and the maximum equals  $b'_j$ . If a negative coefficient  $d'_{j,N_q}$  is found, we search column  $N_q$  of  $D'$  for positive elements  $d'_{s,N_q}$ , using lexicographic comparison to determine which basic variable will be replaced. (The basic variable replaced cannot be  $\widehat{x}_{B_j}$  because  $d'_{j,N_q}$  is negative.) Here the unboundedness of the objective function  $\widehat{x}_i$  is detected if no positive elements are found in column  $N_q$  of  $D'$ . This indicates that the interval matrix  $A$  is singular, and computation ceases.

If  $\widehat{x}_i$  is a nonbasic variable, such as  $\widehat{x}_{N_k}$ , the first step is to convert it to a basic variable. Here we proceed as if column  $N_k$  had been determined as the column to change for an objective function, making the variable basic by the process just described. Now with the variable basic, the procedure is the same as that just described.

The procedure for finding the minimum of a variable  $\widehat{x}_i$  is similar if  $\widehat{x}_i$  is the basic variable  $\widehat{x}_{B_j}$ , except that we look for positive coefficients  $d'_{j,N_k}$  instead of negative ones. And if  $\widehat{x}_i$  is nonbasic, we are done, because the minimum is identified as zero. Whenever a zero minimum value is obtained for a variable  $\widehat{x}_i$ , this signals that the orthant  $Q_N$ , identical to the current orthant  $Q$  except for the opposite  $x_i$  sign, must be investigated. The task queue list  $L$  is searched to see if  $Q_N$  is already on the list. If it is, the minimum problem for  $\widehat{x}_i$  in the  $Q_N$  entry is set to be skipped. If it is not on the list, then a  $Q_N$  entry is added to the end of the task queue with the  $\widehat{x}_i$  minimum problem set to be skipped. In this case the starting basic variables for  $Q_N$  are assigned  $Q$ 's basic variables which show a zero value for  $\widehat{x}_i$  because it is nonbasic. (We may assume  $\widehat{x}_i$  is nonbasic because it is made nonbasic should it happen not to be so.) Note that  $D_Q$  and  $D_{Q_N}$  differ only in the  $i$ th diagonal element, so the coefficient matrix for orthant  $Q_N$  and the coefficient matrix for  $Q$  differ only in column  $i$ . Suppose the set of equations (11.19) for  $Q$  were supplied and the current basic variables were designated.

By going through the procedure described earlier for obtaining a starting feasible solution, we would obtain the current solution representation without column  $i$  being invoked in the process of generating the identity submatrix  $I$  of  $D'$ , and we would obtain a feasible null point. If these same basic variables are specified for the equations corresponding to  $Q_N$ , a solution representation is obtained which has the same null point, and so this solution is feasible also.

After the maxima and minima of all variables  $\hat{x}_i$  have been determined for the current orthant  $Q$ , these are converted to  $x_i$  values by attaching the signs of orthant  $Q$ , and these values now update the current upper and current lower bounds for the components of  $\mathbf{x}$ . After the last orthant has been investigated, the final values of the current upper and current lower bounds give the desired  $\mathbf{x}$  interval bounds.

## 11.7 The program `linpro` for linear programming problems

The program `linpro` for solving linear programming problems follows the procedures described in Sections 11.3 and 11.4, but has some features that need explanation. The program allows the entry of either linear inequalities or linear equations. And an entered inequality can take either the form

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i$$

or the form

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i$$

Our treatment of linear programming problems presumes that any relation imposed is always an inequality of the first displayed form. How are the other cases handled?

After inequality and equation constants have been entered by a user, the program `linpro` uses a rational matrix with enough rows present so that two rows are assigned to each equation, one row is assigned to each inequality, and one row is assigned to the objective function. An inequality of the second displayed form is converted to an inequality of the first displayed form by multiplying its coefficients and right side constant by  $-1$ , and the inequality is stored this way in the rational matrix.

An equation is changed into two separate inequalities of the two displayed forms, and these two inequalities are stored in two adjoining rows of the matrix, with a second form inequality being converted to the first form. Slack variables are taken into account by setting appropriate matrix elements to 1.

### 11.8 The program `i_equat` for interval linear equations

The program `i_equat` follows the procedure described in Section 11.6, and obtains rational bounds for each  $x$  component. These bounds determine a “box” in the  $n$ -space, with sides perpendicular to the coordinate axes, within which all solution points to the interval equations must lie. In general, not all points within the box are solution points, but the size of the box cannot be reduced. If only a single orthant is visited, the true solution space is convex, but this may not be the case when several orthants are visited.

### Software Exercises I

These exercises are mainly with the programs `linpro` and `i_equat`. The program `r_equat` is used to verify a text statement.

1. A simple linear programming problem is the following. A small factory produces two items from a raw material. Each item  $a$  requires 5 hours of manpower to assemble, takes up 4 units of raw material, and yields a profit of 3 units. Each item  $b$  requires 2 hours of manpower, takes up 2 units of raw material, and yields a profit of 2 units. If during a week, 120 manpower hours and 100 units of raw materials are available, how many of each item should be produced to maximize profit.

Let  $x_1$  denote the number of  $a$  items produced in a week, and  $x_2$  the number of  $b$  items. The constraint equations are

$$5x_1 + 2x_2 \leq 120$$

$$4x_1 + 2x_2 \leq 100$$

and the objective function is

$$f = 3x_1 + 2x_2$$

Call up `linpro` and solve this linear programming problem, obtaining the result in both rational form and 5 correct decimal places form. The result shows that it is best to concentrate on item  $b$ , and produce no items of  $a$ .

2. Suppose for some reason it is essential that every week all the manpower hours be used productively, and all raw materials be consumed. In this case the constraint inequalities become constraint equations. Edit the `linpro` log file appropriately to fit this situation, and call up `linpro linpro`. Now more units of item  $a$  are produced than item  $b$ .

3. In the discussion of the interval equation example (11.18), it is stated that the solution to  $A^{(m)}\mathbf{x} = \mathbf{b}^{(m)}$  is a point in the first quadrant. Verify this assertion by calling up `r_equat` and entering the example's  $A^{(m)}$  and  $\mathbf{b}^{(m)}$ . The rational solution vector obtained has components  $x_1 = 11/26$  and  $x_2 = 1/26$ .
4. Call up `i_equat` and verify that equations (11.18) have the solution obtained in the text.
5. The program `i_equat` displays the orthant being investigated, but normally the solution is obtained so quickly that this temporary display goes unnoticed. If the linear interval equations supplied to `i_equat` have  $\mathbf{x} = \mathbf{0}$  as a solution point, then every orthant of the  $n$ -space is investigated. To see the orthants displayed as `i_equat` carries out a search through all orthants, specify 10 equations, take  $A^{(m)}$  equal to an identity matrix and  $\mathbf{b}^{(m)}$  equal to the zero vector, and take all elements of  $A^{(w)}$  and  $\mathbf{b}^{(w)}$  equal to 0.01.

## Notes and References

- A. Two texts on linear programming are the books by Brickman [2] and by Ignizio and Cavalier [5].
- B. The linear programming method for solving linear interval equations [1] that is described in Section 11.6 is a generalization of a method proposed by Oettli [7] for the case where the  $\mathbf{x}$  solution points lie in a single orthant. Other methods for handling this single orthant case are given in Neumaier's book [6]. A general method for treating linear interval equations, with no restrictions placed on the  $\mathbf{x}$  solution, was found by Rohn [9] and is described in Neumaier's book.
- C. An example of linear interval equations with a nonconvex solution set contained in several orthants was given by Hansen [4].

- [1] Aberth, O., The solution of linear interval equations by a linear programming method, *Linear Algebra Appl.* **259** (1997), 271–279.
- [2] Brickman, L., *Mathematical Introduction to Linear Programming and Game Theory*, Springer-Verlag (Series: Undergraduate Texts in Mathematics), New York, 1989.
- [3] Dantzig, G. B., Orden, A., and Wolfe, P., The generalized simplex method for minimizing a linear form under linear inequality restraints, *Pacific J. Math.* **5** (1955), 183–195.
- [4] Hansen, E., On the solution of linear algebraic equations with interval coefficients, *Linear Algebra Appl.* **2** (1969), 153–165.
- [5] Ignizio, J. P. and Cavalier, T. M., *Linear Programming*, Prentice Hall (International Series in Industrial and Systems Engineering), Englewood Cliffs, NJ, 1994.
- [6] Neumaier, A., *Interval Methods of Systems of Equations, Encyclopedia of Mathematics and its Applications*, Cambridge University Press, Cambridge, 1990.

- [7] Oettli, W., On the solution set of a linear system with inaccurate coefficients, *SIAM J. Numer. Anal.* **2** (1965), 115–118.
- [8] Oettli, W. and Prager, W., Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides, *SIAM J. Numer. Anal.* **2** (1965), 291–299.
- [9] Rohn, J., Systems of linear interval equations, *Linear Algebra Appl.* **126** (1989), 39–78.

# Finding Where Several Functions are Zero



In Chapter 7 we considered the problem of finding where a function  $f(x)$  is zero. This chapter treats the more general problem of finding the points where each of several specified functions is zero. The demo program `zeros`, which solves the simpler problem, also solves the more general problem.

## 12.1 The general problem for real elementary functions

Suppose we must find where  $n$  real elementary functions of  $n$  variables are simultaneously zero. The equations that must be satisfied are

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ f_2(x_1, x_2, \dots, x_n) &= 0 \\ \vdots & \qquad \qquad \qquad \vdots \\ f_n(x_1, x_2, \dots, x_n) &= 0 \end{aligned} \tag{12.1}$$

We use the simpler problem's terminology and call an argument point  $(c_1, c_2, \dots, c_n)$  at which all these equations hold a *zero* of the functions. In problems of this kind there often is some specific region of the argument space in which zeros are sought. Initially we assume the region of interest is defined by restricting each variable to a finite interval:

$$a_i \leq x_i \leq b_i \quad \text{for } i = 1, 2, \dots, n$$

Later in this chapter we consider other ways to specify the region of interest.

## 12.2 Finding a suitable solvable problem

For the general problem we employ vector notation. We use  $\mathbf{x}$  to denote a vector with components  $x_1, x_2, \dots, x_n$ . Such a vector is determined “to  $k$  decimal places” by giving every component to  $k$  decimal places. We have now a vector-valued function  $\mathbf{f}(\mathbf{x})$  with components

$$f_i(\mathbf{x}) = f_i(x_1, x_2, \dots, x_n) \quad i = 1, 2, \dots, n$$

The function  $\mathbf{f}(\mathbf{x})$  is called elementary if all component functions are elementary. The search region is defined by the relations  $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$ , where the constant vector  $\mathbf{a}$  has components equal to the left endpoints of the  $n$  intervals defining the region, and the constant vector  $\mathbf{b}$  has components equal to the right endpoints. Here the vector relation  $\mathbf{x} \leq \mathbf{y}$  denotes the  $n$  component relations  $x_i \leq y_i$ . When  $n = 3$ , the search region is a *box* in the argument space. In general, for any  $n$ , we will use the convenient term “box” for this search region defined by two vector constants. Our problem now is that of finding in a specified box  $B$  the arguments  $\mathbf{x}$  where  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ , with  $\mathbf{0}$  of course denoting a vector with all components zero.

The method described in Chapter 7 for the case  $n = 1$  can be generalized to apply when  $n > 1$ . However, there are parts of the  $f(x)$  procedure that need reinterpretation. If the  $f'(x)$  interval is of one sign in a subinterval  $[c, d]$ , this indicates that there can be at most one zero in  $[c, d]$ . To obtain a similar inference for  $\mathbf{f}(\mathbf{x})$  over a subbox  $B^{(S)}$ , we must compute a Jacobian interval over  $B^{(S)}$ . The Jacobian of  $\mathbf{f}$  at a point  $\mathbf{x}$  is the determinant of an  $n$ -square matrix  $J(\mathbf{f}(\mathbf{x}))$  defined by the equation

$$J(\mathbf{f}(\mathbf{x})) = \begin{vmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \frac{\partial f_n(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{vmatrix}$$

Thus the Jacobian is  $\det J(\mathbf{f}(\mathbf{x}))$ . When  $n = 1$  and  $\mathbf{f}(\mathbf{x})$  becomes  $f(x)$ , the Jacobian is the derivative  $df(x)/dx$ . A commonly used symbol for the Jacobian is  $\partial(f_1, f_2, \dots, f_n)/\partial(x_1, x_2, \dots, x_n)$ , which we employ later, in Chapter 16. If  $\mathbf{f}(\mathbf{x})$  is such that we can compute intervals over  $B^{(S)}$  for all the elements of  $J(\mathbf{f}(\mathbf{x}))$ , then we also can compute an interval for the Jacobian. If we find that this interval is positive or negative, then there can be at most one zero in  $B^{(S)}$ . This becomes clear after we prove the following well-known generalization of the Mean Value Theorem.

**Theorem 12.1** Let the box  $B$  be defined by  $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$  where  $\mathbf{a}$  and  $\mathbf{b}$  are two constant vectors. If  $\mathbf{f}(\mathbf{x})$  is defined in  $B$  and has all first partial derivatives there, then if  $\mathbf{y}$  and  $\mathbf{z}$  are any two points in  $B$ , there are points  $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_n$  along the line segment joining  $\mathbf{y}$  and  $\mathbf{z}$  such that

$$f_i(\mathbf{z}) - f_i(\mathbf{y}) = \sum_{j=1}^n \frac{\partial f_i(\mathbf{c}_i)}{\partial x_j} (z_j - y_j) \quad \text{for } i = 1, 2, \dots, n$$

If we set  $\mathbf{x} = (1 - t)\mathbf{y} + t\mathbf{z}$ , where the parameter  $t$  varies from 0 to 1, then  $\mathbf{x}$  varies along the line segment from  $\mathbf{y}$  to  $\mathbf{z}$ . For each  $i$ , let  $F_i(t) = f_i([1 - t]\mathbf{y} + t\mathbf{z})$ . Making use of the Mean Value Theorem, for each  $i$  there is a point  $t_i$  in  $(0, 1)$  such that

$$\begin{aligned} F_i(1) - F_i(0) &= f_i(\mathbf{z}) - f_i(\mathbf{y}) = \frac{dF_i(t_i)}{dt} (1 - 0) = \sum_{j=1}^n \frac{\partial f_i([1 - t_i]\mathbf{y} + t_i\mathbf{z})}{\partial x_j} \frac{\partial x_j}{\partial t} \\ &= \sum_{j=1}^n \frac{\partial f_i([1 - t_i]\mathbf{y} + t_i\mathbf{z})}{\partial x_j} (z_j - y_j) \end{aligned}$$

The point  $\mathbf{c}_i$  is  $[1 - t_i]\mathbf{y} + t_i\mathbf{z}$  and the theorem is proved.

**Corollary.** If the computed Jacobian interval over a box  $B$  is positive or negative, then  $\mathbf{f}(\mathbf{x})$  is one-to-one over  $B$ , and as a consequence, there can be at most one zero in  $B$ .

If  $\mathbf{y}$  and  $\mathbf{z}$  were two distinct arguments in  $B$  with  $\mathbf{f}(\mathbf{y}) = \mathbf{f}(\mathbf{z})$ , it would follow from the theorem that there were points  $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_n$  inside  $B$  such that

$$\begin{bmatrix} \frac{\partial f_1(\mathbf{c}_1)}{\partial x_1} & \frac{\partial f_1(\mathbf{c}_1)}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{c}_1)}{\partial x_n} \\ \frac{\partial f_2(\mathbf{c}_2)}{\partial x_1} & \frac{\partial f_2(\mathbf{c}_2)}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{c}_2)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(\mathbf{c}_n)}{\partial x_1} & \frac{\partial f_n(\mathbf{c}_n)}{\partial x_2} & \cdots & \frac{\partial f_n(\mathbf{c}_n)}{\partial x_n} \end{bmatrix} \begin{bmatrix} z_1 - y_1 \\ z_2 - y_2 \\ \vdots \\ z_n - y_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The determinant of the displayed matrix must be zero, because the product of the matrix with the nonzero vector  $\mathbf{z} - \mathbf{y}$  is  $\mathbf{0}$ . The elements of each matrix row are evaluated at different points, unlike a Jacobian matrix, where all elements are evaluated at the same point. Nevertheless, each element of this matrix is within



the interval used for the element in the Jacobian evaluation over the box  $B$ . If a positive or negative Jacobian interval is obtained, it is not possible for the determinant of this matrix to be zero. So  $\mathbf{f}(\mathbf{y}) = \mathbf{f}(\mathbf{z})$  is impossible, and the  $\mathbf{f}(\mathbf{x})$  mapping over  $B$  is one-to-one.

The zero-finding procedure for dimension  $n = 1$  has another part that requires reinterpretation for dimension  $n > 1$ . For a container subinterval  $[c, d]$ , there is at least one  $f(x)$  zero in  $[c, d]$  if the signs of  $f(c)$  and  $f(d)$  are different. For a container subbox  $B^{(S)}$ , is there an analogous test?

If  $\mathbf{f}(\mathbf{x}) \neq \mathbf{0}$  on the boundary of  $B^{(S)}$ , there are two ways known to generalize the simple sign test used when  $n = 1$ . One way is to compute a nonnegative integer called the *crossing parity*, which can have only two values, 0 or 1, with a 1 value indicating the presence of at least one zero. This integer test is described in Section 12.4.

A second way of generalizing the simple sign test is to compute an integer called the *crossing number*, which can be positive, negative, or zero. A nonzero crossing number indicates the presence of at least one zero. This integer is related to the crossing parity, and may be considered an improvement over that test. This second integer test is described in Section 12.5. The computation for the crossing number is only slightly more complicated than the computation for the crossing parity, and, accordingly, the program `zeros` uses the crossing number test.

Now we can generalize the  $f(x)$  solvable problem to the case of  $\mathbf{f}(\mathbf{x})$  over a box  $B$ . First we present some needed terminology for the general case. A zero  $\mathbf{x}_0$  is *simple* if the Jacobian at  $\mathbf{x}_0$  is nonzero. The *tilde-box* of a point  $\mathbf{x}$  is the box determined by the tilde-intervals of the components  $x_i$ . And the halfwidth of a box  $B$  is the largest halfwidth of its component intervals.

**Solvable Problem 12.1** For any elementary function  $\mathbf{f}(\mathbf{x})$  defined on a box  $B$  and having at most a finite number of zeros there, and for any positive integers  $k_1$  and  $k_2$ , locate to  $k_1$  decimal places a point  $\mathbf{x}_0$  on the boundary of  $B$  at which every component of  $\mathbf{f}(\mathbf{x}_0)$  is less in magnitude than  $10^{-k_1}$ , and halt. Or, if  $\mathbf{f}(\mathbf{x}) \neq \mathbf{0}$  on the boundary of  $B$ , bound all the zeros of  $\mathbf{f}$  in  $B$  by:

- (1) giving, to  $k_2$  decimal places, points identified as simple zeros, or
- (2) giving, to  $k_2$  decimal places, points identified as containing within their tilde-box, a subbox where  $\max_i |f_i(\mathbf{x})| < 10^{-k_2}$ ; the subbox is certain to contain at least one zero if the  $\mathbf{f}(\mathbf{x})$  crossing number over the subbox is nonzero.

This is the first time a solvable problem allows two different values for decimal places,  $k_1$  and  $k_2$ . The integer  $k_1$  is for unexpected outcomes and the integer  $k_2$  is for expected outcomes. We do not distinguish between the two integers in our description of practical solution methods. We always use  $k$  for the number of decimal places wanted.

## 12.3 Extending the $\mathbf{f}(\mathbf{x})$ solution method to the general problem

We describe now the general computation scheme for solving this problem. In the computation it is helpful to use a reference box  $B^{(u)}$ , with all dimension intervals being  $[0, 1]$ , the  $i$ th interval with variable  $u_i$  being mapped linearly into the corresponding interval  $[a_i, b_i]$  of  $x_i$ , according to the rule

$$x_i = (1 - u_i)a_i + u_i b_i$$

Any required subdivision of  $B$  is done by making the appropriate subdivision of the reference box  $B^{(u)}$ , with each interval endpoint of a  $B^{(u)}$  subbox maintained as an exact number, with a range of zero. The interval endpoints for the corresponding  $B$  subbox are obtained by using the linear relations given. Controlling the subdivision of  $B$  this way has two advantages. If the precision of computation is increased, the endpoints of the intervals defining a subbox  $B$  are obtained to higher precision too. And it is always possible to determine when two subboxes of  $B$  are adjoining, by comparing the exact interval endpoints of their  $B^{(u)}$  reference subboxes. In our description of the zero-finding procedure, each subbox of  $B$  is to be understood as defined by a corresponding subbox of  $B^{(u)}$ .

The first step of the general procedure is to test  $\mathbf{f}(\mathbf{x})$  to make certain it is defined over  $B$ . We need here a task queue holding boxes to be checked. Initially the queue holds just the starting box  $B$ . The task queue cycle with the first queue box  $B^{(Q)}$  is the following. We set all series primitives  $x_j$  to their box interval values, and then generate intervals for all component functions  $f_j$ . If there are no series errors, the subbox is discarded. If there is an error in the evaluation of a component, we construct all possible subboxes that can be formed by bisecting all  $B^{(Q)}$  dimension intervals, and these  $2^n$  subboxes replace  $B^{(Q)}$  on the queue. Eventually, either the queue becomes empty, in which case  $\mathbf{f}(\mathbf{x})$  passes its test, or else the leading queue box becomes small enough to fit inside the tilde-box of its centerpoint, if this centerpoint were displayed to  $k$  decimal places. The box centerpoint can now be displayed to  $k$  decimal places to indicate a point in the proposed search box  $B$  where some component of  $\mathbf{f}$  cannot be computed.

Next we test  $\mathbf{f}$  over the boundary of  $B$  to make certain that the components of  $\mathbf{f}(\mathbf{x})$  are nonzero there. Again a task queue is needed, but for this examination we must allow the queue to hold domains that define a side of a box or a part of a side. A queue domain now has  $n$  fields  $s_1, s_2, \dots, s_n$ , and each field  $s_i$  is flagged either as a "point" holding a single number  $c_i$ , or as an "interval" holding a number pair  $c_i, d_i$ , with  $c_i < d_i$ . We distinguish the two cases by writing  $s_i = c_i$

or  $s_i = [c_i, d_i]$ . The queue initially has  $2n$  domains defining the boundary of  $B$ . For example, the side of  $B$  that has  $x_j$  fixed at  $a_j$  is obtained as a queue domain by setting

$$s_i = \begin{cases} a_j & \text{for } i = j \\ [a_i, b_i] & \text{for } i \neq j \end{cases}$$

The task queue cycle with the leading queue domain is as follows. We use the  $s_i$  fields to set the  $x_i$  primitives to their interval or point values, and then obtain intervals for all components  $f_i$ . If any  $f_i$  interval is positive or negative, the domain is discarded. On the other hand, if for each component  $f_i$  we obtain an interval overlapping 0, then we construct all possible subdomains that can be formed by bisecting interval fields and copying the point field, and these  $2^{n-1}$  subdomains replace the leading queue domain. Eventually, either the queue becomes empty, in which case the test is passed, or the leading queue domain fails within the tilde-box of its centerpoint if it were displayed to  $k$  decimal places, and simultaneously, the relations  $|f_i(\mathbf{x})| < 10^{-k}$  hold for all  $i$ . When this second case occurs, the centerpoint is displayed to  $k$  decimal places as a point on the boundary of  $B$  where  $\max_i |f_i(\mathbf{x})| < 10^{-k}$ .

Once these two tests are passed, the next phase is to get a list of small  $B$  subboxes where a zero is likely. The target halfwidth  $W$  is set to its initial value, such as 0.1. Our first goal will be to find  $B$  subboxes where a zero is likely, with the subbox halfwidth  $< W$ . Again we need a task queue holding subboxes, and initially the queue holds just the box  $B$ . The task queue cycle is as follows. We set the primitives  $x_j$  to the component intervals of the first queue box  $B^{(Q)}$ , and then attempt to generate intervals for the functions  $f_i$ . If we obtain a series evaluation error, then  $B^{(Q)}$  is divided into two subboxes by bisecting its largest component interval, and these two subboxes replace  $B^{(Q)}$  on the queue. If we obtain intervals for all functions  $f_i$ , and one of these intervals is positive or negative, then  $B^{(Q)}$  is discarded. If all  $f_i$  intervals overlap 0, the halfwidth of  $B^{(Q)}$  is compared with  $W$ , and if it is smaller, then  $B^{(Q)}$  is moved to another initially empty list  $L$ , for later processing. Otherwise  $B^{(Q)}$  is divided into two subboxes by bisecting its largest component interval, and these two subboxes replace  $B^{(Q)}$  on the queue. Eventually the queue becomes empty. If the list  $L$  is also empty we are done, since there are no zeros in  $B$ .

The next part of the procedure is to arrange the  $L$  subboxes into sets of adjoining boxes. (Two boxes of dimension  $n$  adjoin if the intersection of their boundaries is a box of dimension  $n - 1$ .) Any subbox belonging to one of these sets is connected to any other subbox of the set either directly or via other adjoining subboxes of the set. For each set we make up a container box  $B^{(C)}$  just large enough in each dimension, to contain all of the subboxes of a set. For any member subbox of a container box  $B^{(C)}$ , it is certain that

$\mathbf{f}$  is not  $\mathbf{0}$  on any boundary it shares with the container box, because any such point  $\mathbf{x}$  is either a boundary point of  $B$ , implying  $\mathbf{f}(\mathbf{x}) \neq \mathbf{0}$ , or is also a boundary point of some adjoining subbox that got discarded in the testing process, again implying  $\mathbf{f}(\mathbf{x}) \neq \mathbf{0}$ .

Next an interval for the Jacobian is computed for each container box  $B^{(C)}$ . If the Jacobian interval is positive or negative, according to the Theorem 12.1 Corollary, there is at most one simple zero inside  $B^{(C)}$ , and the zero, if present, usually can be found easily to  $k$  decimal places by the generalized Newton's method, discussed in Section 12.8.

For any container box  $B^{(C)}$  with a Jacobian interval that overlaps 0, or with a Jacobian interval computation attempt that fails for some reason, the box is checked to determine whether it satisfies the requirements of outcome (2) of Problem 12.1. If it does, then after the crossing number is computed for the box, and the box centerpoint is displayed, the container box and its associated subboxes are discarded.

Any container boxes failing these tests are also discarded, after first moving their member subboxes back to the task queue. The parameter  $W$  is reduced by some factor, such as 0.1, and another cycle of subbox processing ensues.

## 12.4 The crossing parity

Let  $B$  be a box with  $\mathbf{f}(\mathbf{x}) \neq \mathbf{0}$  on the the boundary of the box. We use the notation  $\partial B$  to denote the boundary of  $B$ . For dimension  $n > 1$ , the  $\mathbf{f}$  image of  $\partial B$  is an  $(n - 1)$ -dimensional "surface". The general idea of the crossing parity test is that if this surface encloses the origin, then a straight line ray from the origin should pierce the surface at an odd number of points and then be free of it. And if the surface does not enclose the origin, then a ray from the origin either does not pierce the surface or pierces it at an even number of points. The ray used must not be tangent to the surface at any point, for then the touched tangential point is not a piercing point. So a suitable ray for this test is what we term a "nontangential" ray.

The crossing parity is defined as the number of times, modulus 2, that a nontangential ray from the origin pierces the  $\mathbf{f}$  image of  $\partial B$ . In modulus 2 addition of integers, indicated by the abbreviation "mod 2", multiples of 2 are discarded from the sum, so the result is always the 0 or 1 remainder that is left. Accordingly, the crossing parity has only two possible values, 0 or 1, and if it is 1, this indicates that  $B$  contains at least one  $\mathbf{f}(\mathbf{x})$  zero. If we imagine the ray from the origin being varied slightly, the parity value does not change unless the ray leaves some part of  $\mathbf{f}(\partial B)$ , and as it does, an entering piercing point combines with an exiting piercing point at a tangential point, so again the parity does not change. Thus the crossing parity does not depend on which nontangential ray from the origin we use.

For dimension  $n = 1$ , the box  $B$  defines a single interval  $[a_1, b_1]$ , and the boundary  $\partial B$  is a nonconnected set, made up of two distinct points. This is unlike the situation with higher dimensions, where  $\partial B$  is a connected set. Nevertheless, a crossing parity is defined for  $n = 1$ , where a ray from the origin is either the positive  $x_1$  axis or the negative  $x_1$  axis. If  $f_1(a_1)$  and  $f_1(b_1)$  are both positive or both negative, the crossing parity is easily seen to be 0, using either of the two rays to make the crossing parity test. The crossing parity is 1 only if  $f_1(a_1)$  and  $f_1(b_1)$  have opposite signs. Thus for  $n = 1$ , the crossing parity is equivalent to the test of the  $f_1(a_1)$  and  $f_1(b_1)$  signs.

For dimension  $n = 2$ , it is possible for the  $\mathbf{f}$  image of  $\partial B$  to loop around the origin any number of times. For instance, suppose that the two intervals defining the box  $B$  are both  $[-1, +1]$  and that  $\mathbf{f}$  is defined by the equations

$$\begin{aligned} f_1(x_1, x_2) &= x_1^2 - x_2^2 \\ f_2(x_1, x_2) &= 2x_1x_2 \end{aligned} \tag{12.2}$$

Here we have used the real and imaginary parts of the complex function

$$z^2 = (x_1 + ix_2)^2 = (x_1^2 - x_2^2) + i(2x_1x_2)$$

to specify  $f_1$  and  $f_2$ . If we imagine the square domain  $B$  situated in the complex plane, with the  $x_1$  axis the real axis, and the  $x_2$  axis the imaginary axis, then the  $\mathbf{f}$  image of any domain point  $P$  is  $P^2$ , using complex arithmetic. The  $\mathbf{f}$  image of  $\partial B$  loops twice around the origin.

Similarly, using the same  $B$  intervals, one can create an  $\mathbf{f}$  function such that the  $\mathbf{f}$  image of  $\partial B$  loops around the origin  $k$  times, where  $k$  is any positive integer. Take as  $f_1$  and  $f_2$  the real and imaginary parts of the complex function  $z^k = (x_1 + ix_2)^k$ .

These examples illustrate a shortcoming of the crossing parity test. Here we obtain a crossing parity of 1 only if  $k$ , the number of loops, is odd, even though for every  $k$  the corresponding example function has a zero in  $B$ . The test described next does not have this shortcoming.

## 12.5 The crossing number and the topological degree

The general idea of the crossing number is as follows. For dimension  $n > 1$ , the set  $\partial B$  has an “inside” and an “outside”. If  $\mathbf{f}$  has a nonzero Jacobian, the  $\mathbf{f}$  image of  $\partial B$  is also a set with an inside and outside. If we can distinguish somehow between the two sides of  $\mathbf{f}(\partial B)$ , then we can determine whether a nontangential ray from the origin is entering or leaving  $\mathbf{f}(\partial B)$ , instead of merely determining the ray is piercing  $\mathbf{f}(\partial B)$ .

Suppose that at each point of  $\mathbf{f}(\partial B)$  we are able to define a normal vector that points in a consistent direction. That is, the normal vector either points outward

everywhere or it points inward everywhere. With such a normal vector, the crossing parity test is improved, because now, instead of counting the number (mod 2) of  $\mathbf{f}(\partial B)$  points pierced by a nontangential ray, we assign to each piercing point a count of +1 if the ray is in the general direction of the normal, and assign a count of -1 if the ray is in the general direction of a vector opposite to the normal. The sum of the ray's assigned counts at all the  $\mathbf{f}(\partial B)$  piercing points can be positive, negative, or 0, and this sum is the ray's *crossing number*. A nonzero crossing number indicates the presence of at least one zero within  $B$ .

For elementary functions, we describe a method of defining a normal vector for dimension  $n = 3$ , and then generalize the method. In dimension  $n = 3$ , the cross product of the vector  $\mathbf{a} = a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$  with the vector  $\mathbf{b} = b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}$  is the vector  $\mathbf{a} \times \mathbf{b}$  defined by the equation

$$\mathbf{a} \times \mathbf{b} = \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix}$$

This vector is perpendicular to both  $\mathbf{a}$  and  $\mathbf{b}$ , because the dot product of it with either vector yields a determinant with two identical rows:

$$\begin{aligned} (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{a} &= \det \begin{bmatrix} a_1 & a_2 & a_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} = 0 \\ (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{b} &= \det \begin{bmatrix} b_1 & b_2 & b_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} = 0 \end{aligned}$$

In dimension  $n = 3$ , a box  $B$  is defined by three intervals, and  $\partial B$  consists of the 6 sides of the box. At any point of a side of  $\partial B$ , one of the three  $x_i$  variables is constant, but the other two variables can vary to generate the particular side of  $\partial B$  containing the point. Thus, for each side of  $\partial B$ , of the three vector partial derivatives  $\partial\mathbf{f}/\partial x_1$ ,  $\partial\mathbf{f}/\partial x_2$ ,  $\partial\mathbf{f}/\partial x_3$ , two are defined, and both vectors are tangent to the  $\mathbf{f}$  image of the  $\partial B$  side. We take the normal vector at an image point to be the cross product of the two defined vector partial derivatives times a numerical factor that can equal only +1 or -1. The numerical factor is chosen to obtain an outward-pointing normal vector for the simple case where  $\mathbf{f}$  is the identity function.

To generalize the method to apply to any dimension  $n > 1$ , let  $\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_n$  be unit basis vectors associated with the  $n$  dimensional coordinate system  $x_1, x_2, \dots, x_n$ . That is,  $\mathbf{x} = \sum_i x_i \mathbf{e}_i$ . At any  $\mathbf{f}$  image point, the normal vector  $\mathbf{n}$  is assigned according to the equation

$$\mathbf{n} = (\pm 1) \cdot \det \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \\ \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} = \sum_{i=1}^n y_i \mathbf{e}_i \quad (12.3)$$

Here  $u_1, u_2, \dots, u_{n-1}$  are the  $n - 1$  variables that are “free” on the side of  $\partial B$  that generates the  $\mathbf{f}$  image point. Take  $u_i$  to be the  $i$ th free variable from the list of variables  $x_1, x_2, \dots, x_n$ . Each vector  $\partial \mathbf{f} / \partial u_i$ , whose components form one of the rows of the determinant, is tangent to some line in the  $\mathbf{f}$  image of the side. The vector  $\mathbf{n}$ , being perpendicular to all these vectors, is then perpendicular to the image of the side. The factor  $\pm 1$  is set to either  $+1$  or  $-1$  so that an outward pointing normal vector is obtained when  $\mathbf{f}$  is the identity function.

Suppose  $\mathbf{n}$  is evaluated at the image of some point  $P$  on a side of  $\partial B$ . The free variables are given by the list  $u_1, u_2, \dots, u_{n-1}$ , which, if replaced by their  $x_i$  values, would become the list  $x_1, x_2, \dots, x_n$ , except for one missing, fixed,  $x_i$  variable. Let  $u_n$  be this single fixed variable on this particular side of  $\partial B$ . For one of the intervals defining the box  $B$ ,  $u_n$  is either the interval’s right or left endpoint. The vector  $\frac{\partial \mathbf{f}}{\partial u_n} = \sum_i \frac{\partial f_i}{\partial u_n} \mathbf{e}_i$  can be evaluated at  $P$  if we have access to  $B$  rather than just  $\partial B$ . This vector, if nonzero and imagined situated at  $\mathbf{f}(P)$ , points outward from  $\mathbf{f}(\partial B)$ , but not necessarily perpendicularly, if  $u_n$  is at the right endpoint of its interval. This vector points inward, but not necessarily perpendicularly, if  $u_n$  is at the left endpoint of its interval.

The dot product of  $\mathbf{n}$  with this vector satisfies the relation

$$\mathbf{n} \cdot \frac{\partial \mathbf{f}}{\partial u_n} = (\pm 1) \cdot \det \begin{bmatrix} \frac{\partial f_1}{\partial u_n} & \frac{\partial f_2}{\partial u_n} & \cdots & \frac{\partial f_n}{\partial u_n} \\ \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \cdots & \frac{\partial f_1}{\partial u_n} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \cdots & \frac{\partial f_2}{\partial u_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \frac{\partial f_n}{\partial u_2} & \cdots & \frac{\partial f_n}{\partial u_n} \end{bmatrix}$$

If  $u_n$  is identical to  $x_k$ , then by  $k - 1$  successive row exchanges of the first row with the rows below, this determinant would become the Jacobian for  $\mathbf{f}$  at the point  $P$ . That is, we have

$$\mathbf{n} \cdot \frac{\partial \mathbf{f}}{\partial u_n} = (\pm 1) \cdot (-1)^{k-1} \det J(\mathbf{f}(P)) \tag{12.4}$$

This equation allows us to determine the  $\pm 1$  factor for  $\mathbf{n}$ . When  $\mathbf{f}$  is the identity function, the normal vector  $\mathbf{n}$  points outward from  $\mathbf{f}(\partial B)$ . We know the direction that  $\partial \mathbf{f} / \partial u_n$  points, so the dot product  $\mathbf{n} \cdot \frac{\partial \mathbf{f}}{\partial u_n}$  is positive when  $u_n$  is at its right endpoint, and is negative when  $u_n$  is at its left endpoint. Also, the Jacobian is now the determinant of the identity matrix, and is 1. So at any point  $P$  on  $\partial B$ , the  $\pm 1$  factor is  $(-1)^{k-1}$  for a right endpoint and is  $(-1)^k$  for a left endpoint, where  $k$  is the index of the fixed variable on the side of  $\partial B$  containing  $P$ .

Now that the  $\pm 1$  factor is determined, equation (12.4) becomes

$$\mathbf{n} \cdot \frac{\partial \mathbf{f}}{\partial u_n} = \begin{cases} J(\mathbf{f}(P)) & \text{if } u_n \text{ is at its right endpoint} \\ -J(\mathbf{f}(P)) & \text{if } u_n \text{ is at its left endpoint} \end{cases}$$

Suppose now that  $\mathbf{f}$  is a function such that everywhere in  $B$  its Jacobian is of one sign. The last equation implies that if the Jacobian is positive (as it is for the identity function), the  $\mathbf{f}$  normal vector  $\mathbf{n}$  points outward. and if the Jacobian is negative, the  $\mathbf{f}$  normal vector  $\mathbf{n}$  points inward.

For dimension  $n = 1$ , the box  $B$  consists of a single interval  $[a_1, b_1]$ , and to complete the specification of normal vectors, we take  $\mathbf{n}$  to be  $\mathbf{e}_1$  at the  $b_1$  image point, and take  $\mathbf{n}$  to be  $-\mathbf{e}_1$  at the  $a_1$  image point. This defines a consistent crossing number for the two possible rays, and the crossing number can equal only  $+1$ ,  $-1$ , or  $0$ . A crossing number of  $+1$  or  $-1$  is equivalent to having opposite signs for  $f_1(a_1)$  and  $f_1(b_1)$ .

For an elementary function  $\mathbf{f}$  defined over a box  $B$ , having a nonzero Jacobian everywhere except for a “few” points, does every nontangential ray from the origin yield a crossing number value? Of course there is no difficulty when the dimension  $n$  is 1, but for higher dimensions, there are three ways that nontangential rays can fail to give a crossing number. An elementary function may fail to have the required partial derivatives at some points of  $\partial B$ , and at the  $\mathbf{f}$  image of such points, the normal vector fails to be defined. Also, if any vector  $\partial \mathbf{f} / \partial u_i$  is a zero vector, again the normal vector is not defined. Finally, for any point  $P$  lying on the intersection of two sides of  $\partial B$ , the  $P$  image point gets differing normal vectors according to the side of  $\partial B$  used to define the normal, and so a ray through  $P$ 's image point must be considered as having an undefined crossing number. So a nontangential ray from the origin must be considered “exceptional” if one of these three conditions is encountered. From now on we call a nontangential, nonexceptional ray a “satisfactory” ray.



To summarize, for an elementary function having a nonzero Jacobian everywhere except for a few points, all rays are satisfactory if the dimension  $n$  is 1. For  $n = 2$  there can be a finite number of nonsatisfactory rays, and for  $n = 3$ , there can be curvilinear lines such that rays from the origin intersecting these lines are not satisfactory. For  $n > 3$  the number of nonsatisfactory rays can be even larger, but always the number of such rays is always infinitesimal in comparison with the number of satisfactory rays. A ray that is not satisfactory has neighboring rays that are.

The mathematician Kronecker devised a way of obtaining the crossing number by integration over  $\partial B$ . His integration formula makes the various exceptional or nontangential rays insignificant for an elementary function. The integer that Kronecker's integration formula delivers has been given the name "topological degree". Thus the topological degree may be taken to mean the crossing number obtained for a satisfactory ray from the origin.

## 12.6 Properties of the crossing number

Suppose the elementary function  $\mathbf{f}$  is defined over a box  $B$ . We can divide the box  $B$  into two subboxes by choosing one of the intervals defining  $B$ , dividing it into two subintervals, and then reassigning the subintervals and the remaining  $B$  intervals in the obvious two ways. The two subboxes created this way have a side in common, and we assume that no  $\mathbf{f}(\mathbf{x})$  zero lies on this side. Let the two subboxes created this way from  $B$  be labeled  $B_1$  and  $B_2$ . The crossing number for  $B$  equals the crossing number for  $B_1$  plus the crossing number for  $B_2$ . This is because  $\mathbf{f}(\partial B)$  consists of  $\mathbf{f}(\partial B_1)$  and  $\mathbf{f}(\partial B_2)$  together, except for the presence of the image of the dividing side in  $f(\partial B_1)$  and in  $f(\partial B_2)$ . Consider computing the crossing numbers for  $B_1$ ,  $B_2$ , and  $B$ , by following a satisfactory ray from the origin. The crossing number count for  $B$  equals the crossing number count for  $B_1$  plus the crossing number count for  $B_2$ , except for the counts obtained for  $B_1$  and  $B_2$  where the ray intersects the image of the common dividing side. But at any such intersection point, the normal vector for  $B_1$  is opposite to the normal vector for  $B_2$ , so this intersection point can be ignored when the counts for  $B_1$  and  $B_2$  are added.

If the box  $B$  is divided into several subboxes, with no dividing side intersecting a zero, the crossing number of  $B$  is the sum of the crossing numbers for all the subboxes.

When it is known that all the zeros of  $B$  are simple, a relation useful for crossing number computation can be obtained. We divide  $B$  several times so that each  $B$  zero is isolated within a subbox. The Jacobian at each zero is nonzero, and we may assume the subbox enclosing the zero is small enough that a Jacobian interval computed for the subbox is nonzero. In this case, according to the Theorem 12.1 Corollary, the  $\mathbf{f}$  image of the subbox is one-to-one, so the crossing number for such a subbox is  $+1$  if the Jacobian is positive, and  $-1$

if the Jacobian is negative. The crossing number for any subbox not containing a zero is 0.

Thus, when all  $B$  zeros are simple, the crossing number for  $B$  is  $p - n$ , where  $p$  is the number of zeros with positive Jacobians, and  $n$  is the number of zeros with negative Jacobians. Even for dimension  $n = 1$ , this result holds, but here  $p - n$  can be only  $+1$ ,  $-1$ , or  $0$ .

## 12.7 Computation of the crossing number

The crossing number of an elementary function  $\mathbf{f}$  over a box  $B$  can be calculated using interval arithmetic. The method may be viewed as an attempt to find the crossing number by choosing a ray  $H$  from the origin and finding all the points where this ray crosses  $\mathbf{f}(\partial B)$ . If there are  $N_+$  crossing points where the ray is in the general direction of the normal vector  $\mathbf{n}$ , and  $N_-$  crossing points where the ray is in the general direction of  $-\mathbf{n}$ , then  $N_+ - N_-$  is the crossing number. The interval arithmetic character of the computation allows us to assume that  $H$  is a satisfactory ray, because even if  $H$  were tangential or exceptional, there would be a satisfactory ray within an arbitrarily small arc rotation of  $H$ .

To carry out the ray test procedure, we use a task queue holding domains comprising the boundary of  $B$ , with each queue domain defined by fields  $s_i$ ,  $i = 1, \dots, n$ , previously mentioned in Section 12.3 as fields needed to test  $\partial B$  to make certain  $\mathbf{f}(\mathbf{x}) \neq \mathbf{0}$  there. Recall that a field is flagged as a “point” holding a single number  $a$ , or as an “interval” holding a number pair  $a$  and  $b$ , with  $a < b$ , and we distinguish the two cases by writing  $s_i = a$  or  $s_i = [a, b]$ . Initially the queue is loaded with  $2n$  domains designating the different sides of the box  $B$ .

A queue domain has one additional field, “orientation”, which equals the  $\pm 1$  factor appearing in the expression (12.3) for the normal vector  $\mathbf{n}$ . Recall that this factor equals  $(-1)^k$  or  $(-1)^{k-1}$ , depending on whether the side’s fixed variable  $x_k$  is set to its lefthand interval endpoint or its righthand interval endpoint, respectively.

Initially we take the ray  $H$  to be pointing in the direction of  $\mathbf{e}_1$ , that is, the basis vector pointing in the positive direction of the  $x_1$  coordinate. We search for regions of  $\partial B$  where the ray  $H$  would meet  $\mathbf{f}(\partial B)$ , the image of  $\partial B$ . We do this with the following task queue cycle. Taking the first queue domain, we set the primitives  $x_i$  equal to the corresponding  $s_i$  point or interval values. That is, one variable is set to a fixed value, and the others are set to intervals. Then using formal interval arithmetic, we generate interval values for the component functions  $f_2, f_3, \dots, f_n$ . If any of these intervals do not contain 0, the ray  $H$  cannot intersect the image of this part of  $\partial B$ , and the queue domain is discarded. If all these function intervals contain 0, then we form an interval for  $f_1$ . If the  $f_1$  interval is negative, again the ray  $H$  cannot meet the image of this part of  $\partial B$ , and the queue domain is discarded. If the  $f_1$  interval is positive, it is likely that

the ray  $H$  meets the image of this part of  $\partial B$ , and the queue domain is moved to another initially empty list  $L$ , for later attention. In the remaining case, where the  $f_1$  interval contains 0, it is uncertain whether the ray  $H$  meets the image of this part of the boundary of  $B$ , and we proceed as follows. From the queue domain we construct all possible queue domains that can be formed by replacing interval  $s_i$  fields by either their right half or their left half subintervals, copying point fields  $s_i$ , and copying the orientation value. These  $2^{n-1}$  queue domains replace the first queue domain.

Because the point  $\mathbf{0}$  does not lie in the image of  $\partial B$ , eventually the task queue becomes empty, although it may be necessary to increase the precision of computation to obtain this. (Every time a queue domain is subdivided, this is an appropriate time to check the precision of computation.) When the task queue is empty, the list  $L$  is examined. In general  $L$  contains domains which together define a subset  $B^{(1)}$  of  $\partial B$ . The subset  $B^{(1)}$  will in general have a number of connected parts. On the boundary of each connected part the functions  $f_2, f_3, \dots, f_n$  can never be simultaneously zero. This is because each point on the boundary of the connected part is also a point on the boundary of some queue domain that was discarded. This discarded queue domain had one of its functions  $f_2, \dots, f_n$  positive or negative, or had  $f_1$  negative, but  $f_1$  negative did not occur, because  $f_1$  was positive on the adjoining connected part.

Consider any domain on  $L$  with orientation  $\sigma$  defining a subdomain  $S$  of  $B^{(1)}$ . The ray  $H$  may meet the  $\mathbf{f}$  image of  $S$  at one or more points where  $\mathbf{f}(\mathbf{x})$  is a positive multiple of  $\mathbf{e}_1$ , and for each of these points there will be a point  $P$  of  $S$  where the functions  $f_2, f_3, \dots, f_n$  are simultaneously zero. According to equation (12.3) the normal vector has a positive or negative  $\mathbf{e}_1$  component depending on whether

$$\sigma \det \begin{bmatrix} \frac{\partial f_2}{\partial u_1} & \frac{\partial f_3}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_1} \\ \frac{\partial f_2}{\partial u_2} & \frac{\partial f_3}{\partial u_2} & \cdots & \frac{\partial f_n}{\partial u_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_2}{\partial u_{n-1}} & \frac{\partial f_3}{\partial u_{n-1}} & \cdots & \frac{\partial f_n}{\partial u_{n-1}} \end{bmatrix}$$

is positive or negative. Note that the determinant may be interpreted as the Jacobian for the functions  $f_2, f_3, \dots, f_n$  at the point where all these functions are zero. By the relation obtained in the last paragraph of the preceding section, the contribution made by all such points to the sum  $N_+ - N_-$  is  $\sigma$  times the crossing number of the mapping defined by  $f_2, f_3, \dots, f_n$  on the  $(n-1)$ -dimensional region  $S$ . We need to sum these contributions for each domain on the list  $L$  to obtain  $N_+ - N_-$ . Thus we have replaced our original task of calculating a crossing number for a function with  $n$  components, by the

task of calculating a set of signed crossing numbers for a function with  $n - 1$  components.

To do this second task, we proceed much as we did before, this time loading our task queue with boundary domains for each element on the list  $L$ . For each domain of  $L$  with orientation  $\sigma$ , we form  $2^{n-1}$  boundary domains for the task queue by copying all fields  $s_i$  except for one interval field which is converted to a point field set to an endpoint of the interval. If the interval field converted was the  $j$ th interval field, counting from  $s_1$  toward  $s_n$ , then the orientation assigned to this new domain is  $\sigma$  times  $(-1)^j$  or times  $(-1)^{j-1}$ , depending on whether the left or the right endpoint is assigned. If two  $L$  domains have a common side, then this side gets specified twice on the queue. Such pairs are discarded, since the orientation values assigned to them are always opposite, and so their net contribution is zero. It is also possible for one  $L$  domain's side to be contained within another  $L$  domain's side. In this case the task queue is corrected to specify only the unrepeated part of the larger side. After such duplications are removed, what is left on the queue defines the boundary of  $B^{(1)}$ . The formation of the second task queue is similar to that of the initial task queue, except for the need to remove duplications. If duplications were not removed, the correct crossing number still would be obtained, because of the opposing signs on duplicated sides, but the computation would take longer.

The second task queue computation is similar to the first, except that the number of  $f_i$  functions evaluated is reduced by one. This time we take the ray  $H$  to be pointing in the positive direction of the second coordinate  $x_2$ . Eventually the task queue is empty, and there is another list  $L$  specifying a subdomain  $B^{(2)}$  of  $\partial B^{(1)}$  that requires further examination.

Each time the process is carried out, the dimension of the point sets examined and the number of functions treated is reduced by one. After the next-to-last cycle, the list  $L$  defines a set  $B^{(n-1)}$  that contains a number of line segments, each with an orientation  $\sigma$  assigned. The sum of contributions  $\sigma$  times crossing number for all these segments must be found to obtain  $N_+ - N_-$ . Now we are down to just one component  $f_n(\mathbf{x})$ , so the crossing number is easy to obtain. For each line segment  $S$  with orientation  $\sigma$  on the list  $L$ , we construct two point domain elements for the task queue, where the orientation value assigned is  $\sigma \cdot (-1)$  for the point domain having the left endpoint of the single remaining  $S$  interval field, and  $\sigma \cdot (+1)$  for the right endpoint. The ray  $H$  now points in the positive direction of  $x_n$ , and the final queue cycle is to move to the list  $L$  only those point domains  $\mathbf{x}$  for which  $f_n(\mathbf{x})$  is positive. The others are discarded. After this final task queue is empty, the sum of the orientations of the point domains on  $L$  yields  $N_+ - N_-$ , and we finally obtain the crossing number of  $\mathbf{f}(\mathbf{x})$  over  $B$ .

An example may clarify the general process. Consider the mapping  $\mathbf{f}$  from  $R^3$  into  $R^3$  defined by

$$f_1 = x_1^2 + x_2^2 - x_3, \quad f_2 = x_2^2 + x_3^2 - x_1, \quad f_3 = x_3^2 + x_1^2 - x_2$$

These functions have a zero at  $(0, 0, 0)$  and at  $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ . The Jacobian at  $(0, 0, 0)$  is

$$\det \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} = -1$$

and the Jacobian at  $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$  is

$$\det \begin{bmatrix} 1 & 1 & -1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} = 4$$

Thus the crossing number equals  $-1$  if  $B$  is a small box containing just the zero  $(0, 0, 0)$ , it equals  $+1$  if  $B$  is a small box containing just the zero  $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ , and equals zero if  $B$  is a box large enough to contain both zeros. We will compute the crossing number for a box  $B$  containing just the origin, namely the box  $B$  with the interval  $[-\frac{1}{4}, \frac{1}{4}]$  for each variable  $x_i$ .

There are initially six domains defining sides of  $B$  on the first task queue, and only the domain given below is moved to the list  $L$  when it is tested.

$$s_1 = \left[-\frac{1}{4}, \frac{1}{4}\right], \quad s_2 = \left[-\frac{1}{4}, \frac{1}{4}\right], \quad s_3 = -\frac{1}{4}; \quad \sigma = -1 \quad (12.5)$$

For this domain, the interval for both  $f_2$  and  $f_3$  is  $[-\frac{3}{16}, \frac{3}{8}]$  and contains 0 as required, and the interval for  $f_1$  is  $[\frac{1}{4}, \frac{3}{8}]$  and is positive as required. The other five queue domains are discarded either because their  $f_2$  interval does not contain 0, their  $f_3$  interval does not contain 0, or their  $f_1$  interval is negative.

The region  $B^{(1)}$  is defined by the single domain (12.5), and so the queue on the second iteration has four domains defining its sides. Of these only the one given below is transferred to  $L$  when it is tested.

$$s_1 = -\frac{1}{4}, \quad s_2 = \left[-\frac{1}{4}, \frac{1}{4}\right], \quad s_3 = -\frac{1}{4}; \quad \sigma = 1 \quad (12.6)$$

For this domain, the interval for  $f_3$  is  $[-\frac{1}{8}, \frac{3}{8}]$  and contains 0 as required, and the interval for  $f_2$  is  $[\frac{5}{16}, \frac{3}{8}]$  and is positive as required. The other three



(12.7), and with the starting approximation  $\mathbf{x}^{(0)}$  taken as the box centerpoint, the sequence of iterates generally rapidly locates the zero. After an iterate  $\mathbf{x}^{(k+1)}$  is computed, if  $x_i^{(k+1)} \neq x_i^{(k)}$ , then, in preparation for the next cycle,  $\mathbf{x}^{(k+1)}$  is replaced by its midpoint, that is, the vector whose  $i$ th component is  $x_i^{(k+1)}$  with its range set to 0. Eventually we find  $x_i^{(k+1)} = x_i^{(k)}$  for all  $i$ , and then the midpoint of  $\mathbf{x}^{(k)}$  defines the final zero approximation, which we denote  $\mathbf{x}^{(z)}$ .

An error bound for  $\mathbf{x}^{(z)}$  can be obtained by generalizing the error bounding method used in Section 7.3. If  $\mathbf{z}$  is the zero inside  $B^{(C)}$ , then  $\mathbf{f}(\mathbf{z}) = \mathbf{0}$ , and using the Mean Value Theorem once more, we have for  $i = 1, \dots, n$ ,

$$f_i(\mathbf{x}^{(z)}) = f_i(\mathbf{x}^{(z)}) - f_i(\mathbf{z}) = \sum_{j=1}^n \frac{\partial f_i(\mathbf{c}_{ij})}{\partial x_j} (x_j^{(z)} - z_j)$$

where the points  $\mathbf{c}_{ij}$  lie on the line segment joining  $\mathbf{x}^{(z)}$  and  $\mathbf{z}$ . If  $\widehat{J}$  is the  $n$ -square matrix with elements  $\frac{\partial f_i(\mathbf{c}_{ij})}{\partial x_j}$ , then we have

$$\mathbf{f}(\mathbf{x}^{(z)}) = \widehat{J}(\mathbf{x}^{(z)} - \mathbf{z})$$

which implies

$$\mathbf{x}^{(z)} - \mathbf{z} = \widehat{J}^{-1} \mathbf{f}(\mathbf{x}^{(z)})$$

If a matrix  $J^{(C)}$  of interval elements is formed, using the intervals computed in testing the container box, each element of this matrix contains the corresponding element of  $\widehat{J}$ . So if an interval inverse matrix  $[J^{(C)}]^{-1}$  is formed, each component of  $\mathbf{x}^{(z)} - \mathbf{z}$  will be contained within the corresponding interval element of the vector

$$\mathbf{y} = [J^{(C)}]^{-1} \mathbf{f}(\mathbf{x}^{(z)}) \quad (12.8)$$

Thus the error of  $x_i^{(z)}$  is bounded by  $e_i = |\text{midpoint } y_i| + \text{halfwidth } y_i$ . So  $x_i^{(z)} \oplus e_i$  is a correctly ranged zero approximation component. If our error bounding process fails to yield a zero approximation with sufficient correct decimal places, then it is necessary to increase the precision of computation, restart the Newton's method iteration, and repeat this test with a better  $\mathbf{x}^{(z)}$  approximation.

## 12.9 Searching a more general region for zeros

Suppose  $\mathbf{f}$  has two components, and we are interested in finding all the zeros of  $\mathbf{f}$  that lie inside the unit circle  $x_1^2 + x_2^2 \leq 1$ . Solvable Problem 12.1 requires the

search domain to be a rectangle. We could enclose the unit circle in a square and search within the square, but there is a difficulty that could arise. It is possible that a component function  $f_i$  is defined inside the unit circle but not in the square.

It is worthwhile to consider now whether Solvable Problem 12.1 can be changed to allow more general search domains than just boxes. An iterated integral for  $n$  dimensional volume of the form

$$\int_a^b \int_{g_1(x_1)}^{h_1(x_1)} \dots \int_{g_{n-1}(x_1, x_2, \dots, x_{n-1})}^{h_{n-1}(x_1, x_2, \dots, x_{n-1})} dx_1 dx_2 \dots dx_n \quad (12.9)$$

has an integration domain that can serve also as a zero search domain. For instance, for our example problem of finding zeros within the unit circle, we could use the dimension 2 domain of

$$\int_{-1}^1 \int_{-\sqrt{1-x_1^2}}^{\sqrt{1-x_1^2}} dx_1 dx_2$$

To solve Problem 12.1 over boxes, we employed a mapping from a reference unit box  $B^{(u)}$  into the problem box  $B$ . For the circle example we can map from the unit square into the circle with a function  $\mathbf{x}(\mathbf{u})$  defined by

$$\begin{aligned} x_1 &= -1 \cdot (1 - u_1) + 1 \cdot u_1 \\ x_2 &= -\sqrt{1 - x_1^2} \cdot (1 - u_2) + \sqrt{1 - x_1^2} \cdot u_2 \end{aligned}$$

Here  $u_1$  and  $u_2$  vary in  $[0, 1]$ . This mapping allows us to convert a function  $\mathbf{f}(\mathbf{x})$  defined in the circle into a function  $\mathbf{f}(\mathbf{x}(\mathbf{u}))$  defined in the unit square. The mapping from the boundary of the unit square into the boundary of the circle is not one-to-one, since the entire left and right edges of the unit square map onto points, but the mapping from the interior of the unit square into the interior of the circle is one-to-one. Using the composite  $\mathbf{f}$ , we can search the interior of  $B^{(u)}$  for zeros, mapping each zero in  $B^{(u)}$  into a zero in the circle.

**Definition 12.1** An elementary region is a region of  $n$ -space that can be defined as the integration domain of the integral (12.9), where all functions  $h_i$  and  $g_i$  are elementary functions.

An elementary region of dimension 1 is a closed interval  $[a, b]$ . An elementary region  $R$  of dimension 2 is defined by two elementary functions  $h_1(x_1)$  and  $g_1(x_1)$  over an interval  $[a, b]$ , and its interior consist of those points  $(x, y)$  for which the inequalities  $a < x < b$  and  $g_1(x) < y < h_1(x)$  hold.

The method described for solving Problem 12.1 also serves for solving the more general problem given next.



**Solvable Problem 12.2** For any elementary function  $\mathbf{f}(\mathbf{x})$  defined on an elementary region  $R$  and having at most a finite number of zeros there, and for any positive integers  $k_1$  and  $k_2$ , locate to  $k_1$  decimal places a point on the boundary of  $R$  where all components of  $\mathbf{f}(\mathbf{x})$  are less in magnitude than  $10^{-k_1}$  and halt. Or, if  $\mathbf{f}(\mathbf{x}) \neq \mathbf{0}$  on the boundary of  $R$ , bound all the zeros of  $\mathbf{f}$  in  $R$  by:

- (1) giving, to  $k_2$  decimal places, points identified as simple zeros, or
- (2) giving, to  $k_2$  decimal places, points identified as containing within their tilde-box a region where  $\max_i |f_i(\mathbf{x})| < 10^{-k}$ ; the region is certain to contain at least one zero if the  $\mathbf{f}(\mathbf{x})$  crossing number computed over a box defining the region is nonzero.

With an elementary region  $R$ , we map from a unit box  $B^{(u)}$  into  $R$ , using a mapping  $\mathbf{x} = \mathbf{x}(\mathbf{u})$  derived from the functions defining  $R$ . Using the notation of the integral (12.9), we have  $x_1 = (1 - u_1)a + u_1b$ , and for  $i > 1$  we have  $x_i = (1 - u_i)g_{i-1} + u_i h_{i-1}$ . The mapping from the boundary of  $B^{(u)}$  into the boundary of  $R$  may not be one-to-one, but the mapping from the interior of  $B^{(u)}$  to the interior of  $R$  is one-to-one, so we can search for zeros of  $\mathbf{f}$  in the interior of  $R$  by searching for zeros of  $\mathbf{f}(\mathbf{x}(\mathbf{u}))$  in the interior of  $B^{(u)}$ .

The zero-finding process that we used previously will also work for the more general region  $R$ . The  $B^{(u)}$  box is subdivided just as before, but the determination of a degree 0 or degree 1 series expansion for  $\mathbf{f}$  now is done in two steps, reflecting  $\mathbf{f}$ 's composite nature. For any  $\mathbf{u}$  subbox  $B^{(s)}$  of  $B^{(u)}$ , we obtain an  $\mathbf{x}$  subbox  $B^{(x)}$  by finding interval values for the various  $x_i$  expressions given in the preceding paragraph. Then after  $x_i$  primitives are set accordingly, we obtain the  $f_i$  expansions.

As before, we test  $\mathbf{f}$  to make certain it is nonzero over the boundary of  $B^{(u)}$ . The search for zeros is done with a task queue listing subboxes of  $B^{(u)}$  in which a zero is possible.

## Software Exercises J

These exercises, like Software Exercises E, are with the demo program `zeros`.

1. Call up `zeros` and find to 10 decimal places the zero of the Section 12.4 example used to illustrate a deficiency of the crossing parity:

$$f_1(x_1, x_2) = x_1^2 - x_2^2$$

$$f_2(x_1, x_2) = 2x_1x_2$$

Use search intervals  $[-10, 10]$  for both  $x_1$  and  $x_2$ .

Note that the phrase "at least one zero" is used for the single zero possibility found. If the crossing parity had been used by the demo program instead of the

crossing number, the “at least one zero” phrase would not appear because a zero crossing parity would be obtained.

2. Call up `zeros` and find to 10 decimal places the zero of

$$f_1(x_1, x_2) = x_1$$

$$f_2(x_1, x_2) = |x_2|$$

Again use search intervals  $[-10, 10]$  for both  $x_1$  and  $x_2$ .

A zero possibility is indicated near  $x_1 = x_2 = 0$ , but the phrase “at least one zero” does not appear. This function’s Jacobian is not defined along the line  $x_2 = 0$ , is positive for  $x_2 > 0$ , and is negative for  $x_2 < 0$ . The crossing number test is reliable for an elementary function if the function’s Jacobian has one sign throughout the containment box, except, possibly, at a “few” points. This example shows that the crossing number test, though more reliable than the crossing parity test, still can fail to indicate the presence of an existing zero.

3. Call up `zeros` and find to 10 decimal places the zeros of

$$f_1(x_1, x_2) = x_1^2 - x_2$$

$$f_2(x_1, x_2) = x_2^2 - x_1$$

Use the search interval  $[-10, 10]$  for both  $x_1$  and  $x_2$ . Obtain in this way the two simple zeros  $x_1 = x_2 = 0$  and  $x_1 = x_2 = 1$ . Note that one zero is displayed as an exact number, that is, without the usual tilde symbol. Whenever this occurs, it indicates that the equation (12.8) vector  $\mathbf{f}(\mathbf{x}^{(z)})$  is  $\mathbf{0}$ .

4. Edit the log file to change both search intervals from  $[-10, 10]$  to  $[0, 10]$ , and then call up `zeros zeros`. Note the program’s rejection of the problem because the functions no longer test nonzero on the boundary of the search region.

5. Call up `zeros` and reenter the preceding problem but use a circle centered at the origin, with a radius of 2, as the search region. Note that the two simple zeros are again obtained. If you edit the log file to change the radius to  $\sqrt{2}$ , and then call up `zeros zeros`, the problem is rejected because the boundary check fails.

6. Call up `zeros` and find to 10 decimal places the zeros of the three component function used as an example in Section 11.7:

$$f_1 = x_1^2 + x_2^2 - x_3, \quad f_2 = x_2^2 + x_3^2 - x_1, \quad f_3 = x_3^2 + x_1^2 - x_2$$

Take the search interval for  $x_1$ ,  $x_2$ , and  $x_3$  to be  $[-10, 10]$ .

7. Call up `zeros` and find to 1 decimal place the single zero of the two functions

$$f_1 = x_1^2 + x_2^2 - 2, \quad f_2 = x_1 + x_2 - 2$$

Take the search interval for  $x_1$  and  $x_2$  to be  $[-10, 10]$ . The straight line  $x_1 + x_2 = 2$  is tangent to the circle  $x_1^2 + x_2^2 = 2$  at the point  $(1, 1)$ , and the  $\mathbf{f}$  Jacobian is zero there, so this zero is found by the slow method of reducing the size of the container region. Whenever the `zeros` program fails to display any results after a reasonable waiting period, it becomes advisable to interrupt the program with a `control-c`, edit the log file to reduce the number of decimal places to 1 or 2, and then repeat the search by calling up `zeros zeros`.

## Notes and References

- A. The Kronecker integral for topological degree is given in the text by Alexandroff and Hopf [2, pp. 465–467].
  - B. The method given in Section 12.7 for computing crossing number was initially presented as a method for computing the topological degree using interval arithmetic [1]. The method is a version of a method proposed earlier by Kearfott [3].
- [1] Aberth, O., Computation of topological degree using interval arithmetic, and applications, *Math. of Comp.* **62** (1994), 171–178.
- [2] Alexandroff P. and Hopf, H., *Topologie*, Chelsea, New York, 1935.
- [3] Kearfott, R. B., *A summary of recent experiments to compute the topological degree*, Proceedings of an International Conference on Applied Nonlinear Analysis, University of Texas at Arlington, April 20–22, 1978 (V. Laskshmikantham, ed.), Academic Press, New York, 1979, pp. 627–633.

# Optimization Problems



Two programs locate the extreme values of a real function within a specified bounded domain. The program `maxmin` finds the absolute maximum or minimum, and the program `extrema` locates the relative maximum or minimum.

## 13.1 Finding a function's extreme values

A common problem is finding the maximum or minimum of an elementary function  $f(x_1, x_2, \dots, x_n)$  in a specified bounded region of  $n$ -space. Box search regions, defined in Section 12.2, are convenient bounded regions, and we take our computation problem to be locating the maximum or the minimum of an elementary function  $f$  in a specified box  $B$ . Finding the maximum  $M$  of  $f$  is equivalent to finding the minimum  $m$  of  $-f$ , because if the inequality

$$m \leq -f(x_1, x_2, \dots, x_n)$$

is true for all arguments in  $B$ , with equality for at least one argument point, then after we multiply by  $-1$ , the inequality

$$-m \geq f(x_1, x_2, \dots, x_n)$$

is true for all arguments in  $B$ , with equality for at least one argument point. So the  $f$  maximum  $M$  is  $-m$ . Accordingly, we can limit our problem now to locating the minimum value of  $f(x_1, x_2, \dots, x_n)$  in a specified box  $B$ .

Usually one is interested not only in the value of the minimum, but also in the argument or arguments where the minimum is attained. However, locating to  $k$  decimal places even a single argument point where the minimum occurs

is a nonsolvable problem, as may be seen by considering the case of functions  $f_c(x) = cx$ , having the arbitrary constant parameter  $c$  and with the interval  $[-1, 1]$  as the search region. If the parameter  $c$  is zero, the minimum of  $f_c(x)$  occurs at any argument in  $[-1, 1]$ . If  $c$  is positive, the minimum occurs at  $x = -1$ . And if  $c$  is negative, the minimum occurs at  $x = 1$ . A program that could find to  $k$  decimal places one point at which any function  $f_c(x)$  is minimum would let us infer that  $c \geq 0$  if the  $k$  decimal value supplied were negative or zero, and would let us infer that  $c \leq 0$  if the value supplied were positive. However, this contradicts Nonsolvable Problem 3.13.

Thus we must retreat from attempting to determine where the minimum occurs. It is convenient here to use vector notation for  $f$  arguments, so now  $f(x_1, x_2, \dots, x_n)$  is denoted by  $f(x)$ . An  $x$  box is defined by a set of intervals, one interval for each  $x_i$  variable. An alternate problem, with an escape, is

**Solvable Problem 13.1** For an elementary function  $f(x)$  defined in a box  $B$  of  $n$ -space, and for any positive integer  $k$ , give to  $k$  decimal places the minimum value of  $f$  in  $B$ , and give to  $k$  decimal places all  $f$  arguments in  $B$  where  $f$  has a matching  $k$  decimal value, or else define to  $k$  decimal places an  $x$  subbox in which the minimum occurs.

When  $f$  arguments are supplied, no claim is made that the minimum occurs at any of these arguments. If  $k$  is increased, a more accurate minimum value is obtained, and possibly a shorter list of arguments. The escape is for difficult cases, like the example function  $f_c$  with  $c$  close to zero. Here the escape subbox would be identical to the search box, that is, the  $x_1$  interval  $[-1, 1]$ .

In the computation, it is helpful once more to use a reference box  $B^{(u)}$ , where all component intervals of  $\mathbf{u}$  are  $[0, 1]$  and the  $i$ th interval with variable  $u_i$  is mapped linearly into the corresponding interval  $[a_i, b_i]$  of  $x_i$ , according to the rule

$$x_i = (1 - u_i)a_i + u_i b_i$$

This device was described previously in Section 12.3 for the problem of finding zeros. When we say that a subbox  $B^{(s)}$  is divided into two, what occurs is that the reference box for  $B^{(s)}$  gets divided.

The first step in finding the minimum is to divide  $B$  into a list of subboxes such that  $f$  can be computed without error over each of the subboxes. This is a task queue computation, with the queue initially holding just the box  $B$ . The queue cycle consists of attempting to compute an interval for  $f$ , via formal interval arithmetic, over the leading queue subbox. If there is a series error, the leading box is divided into two by bisecting its largest component interval, and the two subboxes replace it on the queue. If an  $f$  interval is obtained, the leading box is added to an initially empty list  $L$ . Eventually the task queue is empty, and the list  $L$  has our required  $B$  dissection.

Combining ideas from computation procedures suggested by Moore [5] [6], Skelboe [10], and Ichida-Fujii [4], we obtain the  $k$ -decimal bound on the minimum of  $f$  efficiently as follows. For each subbox  $B^{(s)}$  on  $L$ , if  $B^{(s)}$  has no boundary points of  $B$ , a degree 1 series evaluation of  $f$  is attempted. If this is obtained, then we have intervals for all the partial derivatives of  $f$  over  $B^{(s)}$ . If any of these partial derivative intervals do not contain 0, then it is certain that the minimum of  $f$  does not occur in  $B^{(s)}$ , and the subbox is discarded. When a degree 1 evaluation is not obtained, a degree 0 evaluation of  $f$  can be obtained, because this is just the formal interval arithmetic computation of  $f$  over  $B^{(s)}$ . For any box  $B^{(s)}$  containing boundary points of  $B$ , just a degree 0 evaluation is obtained. Thus for every subbox undergoing a degree 0 evaluation, and for all the nondiscarded subboxes with degree 1 evaluations, we obtain an interval  $m_s \pm w_s$  for  $f$  over  $B^{(s)}$ . The  $f$  minimum over  $B^{(s)}$  may be as large as  $m_s$ , the  $f$  value at the centerpoint of  $B^{(s)}$ , and may be as small as  $m_s - w_s$ . Thus the  $f$  minimum over  $B^{(s)}$  is somewhere in the interval  $[m_s - w_s, m_s]$ .

The surviving subboxes on the list  $L$  are now arranged in the following order. Subbox  $B_s$  precedes subbox  $B_{s'}$  on the list if  $m_s - w_s < m_{s'} - w_{s'}$ , or, in the case where  $m_s - w_s = m_{s'} - w_{s'}$ , if  $w_s > w_{s'}$ . An initial interval bound  $[M_1, M_2]$  for the minimum of  $f$  may be taken as  $[m_{s_0} - w_{s_0}, m_{s_0}]$  where  $B_{s_0}$  is the first subbox on the list  $L$ .

The following process is repeatedly performed to reduce the width of the interval  $[M_1, M_2]$ . The first subbox on  $L$  is removed and divided into two subboxes by bisecting its largest component interval. Each of these two subboxes goes through the  $f$  evaluation procedure described, and if not discarded, is placed in its appropriate position on the  $L$  list. The new value of  $M_1$  is  $m_{s_0} - w_{s_0}$  where  $B_{s_0}$  is the new leading element on  $L$ . The other value  $M_2$  is not changed unless  $m_{s_0} < M_2$ , in this case  $M_2$  is set equal to  $m_{s_0}$  and then a pass is made through the list  $L$  to discard any subbox  $B_s$  for which  $m_s - w_s > M_2$ .

The minimum of  $f$  equals the ranged number  $\frac{1}{2}(M_1 + M_2) \oplus \frac{1}{2}(M_2 - M_1)$ . The process of bisecting the first element of  $L$  continues until an  $f$  minimum accurate to  $k$  decimal places is obtained.

The next step is to use the surviving members of the final  $L$  list to find where in  $B$  the minimum occurs. The  $L$  subboxes are arranged into connected sets of subboxes, and for each set a containing box  $B^{(c)}$  is constructed, just large enough to hold the members of the set. If each box  $B^{(c)}$  is small enough to fit within the tilde-box of its centerpoint to  $k$  decimal places, then we satisfy the requirements of Problem 13.1 by displaying all the centerpoints to  $k$  decimal places.

If some of the  $B^{(c)}$  boxes fail this criterion, the list  $L$  is processed anew, obtaining  $f$ 's minimum value to more decimal places, in an attempt to get container boxes small enough that centerpoints can be displayed. If there is no progress toward this goal, the escape exit is used. That is, we construct a containing box just large enough to hold all the subboxes of the final list  $L$ , and display to  $k$  decimal places the endpoints of its defining intervals.

### 13.2 Finding where a function's gradient is zero

Often it is useful to determine whether a function  $f$  has one or more local extrema inside a certain region  $R$ . To find these points, one tries to locate the points of  $R$  where the  $f$  gradient is  $\mathbf{0}$ . The gradient of  $f$  is the vector whose  $i$ th component is the partial derivative of  $f$  with respect to variable  $x_i$ . (For dimension  $n = 1$ , the gradient's single component is the derivative of  $f$  with respect to  $x_1$ .) If the  $f$  gradient is defined and is nonzero, it points in the direction where  $f$  increases at the greatest rate.

Assume now that the function  $f$  is such that the  $f$  derivative ( $n = 1$ ) or the several  $f$  partial derivatives ( $n > 1$ ) are defined inside a specified bounded region, which we take to be an  $n$ -dimensional box  $B$ , and that our problem is to locate all points in  $B$  at which the  $f$  gradient vector is  $\mathbf{0}$ . For conciseness, our procedure description presumes  $n > 1$ .

We have  $n$  partial derivative functions  $\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}$  defining the gradient of  $f$ , and we want the points in  $B$  where these functions are simultaneously 0. This problem is similar to Solvable Problem 12.1, which is the problem of finding the points where  $n$  functions of  $n$  variables are simultaneously zero when the search domain is an  $n$ -dimensional box  $B$ . As with that zero-finding problem, to have a solvable problem we must require the  $f$  gradient to be nonzero on the  $B$  boundary. Also, as with that zero-finding problem, to avoid computerbound varieties of the problem, we require the  $f$  gradient to be zero only at a finite number of points in  $B$ .

For the zero-finding problem we used  $\mathbf{x}$  to denote the  $n$  component argument point  $x_1, x_2, \dots, x_n$ , and this notation is useful for the gradient problem too. We are searching for points  $\mathbf{x}$  where the functions  $\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n}$  are simultaneously zero. For the zero-finding problem, it was important to compute the Jacobian of the functions  $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x})$ , and for our problem we must compute the Jacobian of  $\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n}$ , which has the form

$$\det \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix}$$

This determinant is called the *Hessian* of  $f$ , and the corresponding matrix is the Hessian matrix.

We call a point  $\mathbf{x}$  where the gradient is zero a simple zero gradient point if the Hessian of  $f$  at  $\mathbf{x}$  is nonzero. A solvable version of our problem, analogous to Problem 12.1, is

**Solvable Problem 13.2** For an elementary function  $f(\mathbf{x})$  defined and having a gradient on a box  $B$ , with the gradient being  $\mathbf{0}$  at most a finite number of points in  $B$ , for any positive integers  $k_1$  and  $k_2$ , locate to  $k_1$  decimal places a point on the boundary of  $B$  where every component of the gradient is less in magnitude than  $10^{-k_1}$  and halt. Or, if the gradient is not  $\mathbf{0}$  on the boundary of  $B$ , bound all the zero gradient points in  $B$  by:

- (1) giving, to  $k_2$  decimal places, points identified as simple zero gradient points, or
- (2) giving, to  $k_2$  decimal places, points identified as containing within their tilde-box a subbox where  $\max_i \left| \frac{\partial f(\mathbf{x})}{\partial x_i} \right| < 10^{-k_2}$ , the subbox is certain to contain at least one zero gradient point if the crossing number for  $\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n}$  over the subbox is nonzero.

At a simple zero gradient point, it is possible to determine from the point's Hessian matrix whether the point is a local maximum, a local minimum, or a saddle point for  $f$ . To see this, take any such zero gradient point  $\mathbf{x}_0$  and translate  $\mathbf{x}$  coordinates to make this point the origin. A translation does not alter any derivative at the point, so the point's Hessian matrix is unchanged, and its gradient is still  $\mathbf{0}$ . A Taylor series for  $f$ , with the series expansion point  $\mathbf{0}$ , has the form

$$f(\mathbf{x}) = f(\mathbf{0}) + \frac{1}{2!} \sum_{i=0}^n \frac{\partial^2 f(\mathbf{0})}{\partial x_i^2} x_i^2 + \sum_{\substack{i,j=0 \\ i < j}}^n \frac{\partial^2 f(\mathbf{0})}{\partial x_i \partial x_j} x_i x_j + \dots$$

All series terms up to degree 2 are shown. If we let  $H$  denote the Hessian matrix, the series expansion may be expressed in matrix-vector notation as

$$f(\mathbf{x}) = f(\mathbf{0}) + \frac{1}{2} \mathbf{x}^T H \mathbf{x} + \dots$$

Here the symbol  $\mathbf{x}^T$  denotes the row vector transpose of the column vector  $\mathbf{x}$ . (Transpose notation was introduced in Section 9.1.)

The Hessian matrix  $H$  is symmetric, and making use of a basic result of linear algebra, there exists an orthogonal matrix  $P$  with inverse  $P^T$  that transforms  $H$  to a real diagonal matrix  $D$ :

$$D = P^T H P$$



This implies  $H = PDP^T$ , and the equation for  $f(\mathbf{x})$  can be written

$$\begin{aligned} f(\mathbf{x}) &= f(\mathbf{0}) + \frac{1}{2} \mathbf{x}^T PDP^T \mathbf{x} + \dots \\ &= f(\mathbf{0}) + \frac{1}{2} (P^T \mathbf{x})^T D (P^T \mathbf{x}) + \dots \end{aligned}$$

The diagonal elements of  $D$  are  $H$  eigenvalues, and if they are all positive, then  $f(\mathbf{x})$  equals  $f(\mathbf{0})$  plus a sum of squares with positive coefficients, which indicates that  $f$  has a local minimum at the zero gradient point. Similarly, if all the diagonal elements of  $D$  are negative, then  $f$  has a local maximum at the point. If the diagonal elements are a mixture of positive and negative quantities, then  $f$  has a saddlepoint there. No diagonal element of  $D$  can be 0, because this would imply  $\det D = 0$ . But at a simple zero gradient point, the Hessian,  $\det H$ , is nonzero and equals  $\det D$  because

$$\begin{aligned} \det H &= \det(PDP^T) = \det P \det D \det P^T = \det D \cdot (\det P \det P^T) \\ &= \det D \cdot (\det PP^T) = \det D \det I = \det D \end{aligned}$$

The procedure for finding zero gradient points within a box  $B$  is very much like the procedure given in Section 12.3 for the problem of finding zeros of a set of functions  $f_i$  within a box  $B$ . In that procedure, we sometimes compute a Jacobian interval over a container subbox of  $B$ . So if we copy that procedure, then we sometimes compute a Hessian interval over a container subbox. To get a Jacobian interval over a container subbox  $B^{(C)}$ , we obtain a degree 1 series expansion for each component function  $f_i$  over  $B^{(C)}$ . The coefficient of the  $f_i$  series term in  $(x - x_j)$  yields the interval value for the element  $\partial f_i / \partial x_j$  of the Jacobian. To compute a Hessian interval, we need to obtain a degree 2 series expansion of  $f$ . Line (5.21) indicates that twice the coefficient of the series term in  $(x - x_i)^2$  yields the value for the element  $\partial^2 f / \partial x_i^2$  of the Hessian, and the coefficient of the series term in  $(x - x_i)(x - x_j)$  yields the value for both elements  $\partial^2 f / \partial x_i \partial x_j$  and  $\partial^2 f / \partial x_j \partial x_i$  of the Hessian.

The computation for a Jacobian interval, using a Jacobian matrix  $J$  having interval elements, is done in the way described in Section 9.4 for computing a determinant. Using allowable matrix row operations, we attempt to bring the Jacobian matrix to upper triangular form  $J'$ . If we are successful, the product of the  $J'$  diagonal elements gives us the Jacobian interval.

When computing a Hessian interval, it is best to do the determinant computation differently, to determine the type of the simple zero gradient point within  $B^{(C)}$ , that is, whether the point is a local maximum, a local minimum, or a saddle point.

Let  $H$  be the beginning Hessian matrix with interval elements. The first step is to test whether the interval element  $h_{11}$  does not contain the zero point, and if so, to clear column 1 below it. To clear any column 1 element  $h_{k1}$ , we subtract row 1 multiplied by the interval quantity  $h_{k1}/h_{11}$ . The clearing of column 1 is equivalent to a premultiplication of  $H$  by a matrix  $Q_1$  identical to the identity matrix  $I$  except for the elements in column 1 below the diagonal. The column 1 element in row  $k$  is  $-h_{k1}/h_{11}$  instead of being 0.

If the process of testing diagonal elements and using them as column clearers is successful with all columns, we have brought the Hessian matrix to upper triangular form. We show next that the signs of the diagonal elements match the signs of the Hessian's eigenvalues, so the diagonal elements can be tested to determine the type of a simple zero gradient point in the subbox.

When column 1 is cleared, the changed Hessian matrix is  $Q_1H$ , and all its column 1 elements, except the diagonal element, are 0. The first diagonal element of  $Q_1H$  is identical to the first diagonal element of  $H$ . If we were to postmultiply  $Q_1H$  by  $Q_1^T$ , the transpose of  $Q_1$ , the only change made to  $Q_1H$  would be to convert the row 1 elements to the right of the diagonal element to 0. The resulting matrix  $Q_1HQ_1^T$  is symmetric, because  $H$  is symmetric.

The process of using diagonal elements one by one to clear the elements of  $H$  below the diagonal, is equivalent to premultiplying  $H$  by a succession of matrices  $Q_1, Q_2, \dots, Q_{n-1}$ . If we set  $Q$  equal to  $Q_{n-1} \dots Q_2Q_1$ , then the final upper triangular matrix is  $QH$ . By the reasoning of the previous paragraph, the matrix  $QHQ^T$  is a diagonal matrix, with diagonal elements identical to the diagonal elements of the final matrix  $QH$ . The diagonal elements of  $QHQ^T$  are not necessarily eigenvalues of  $H$ , because  $Q$  is not necessarily an orthogonal matrix. However, by Sylvester's law of inertia [3, p. 445], the diagonal elements of  $QHQ^T$  have the same signs as do the eigenvalues of  $H$ . If all the diagonal elements are positive, a simple zero gradient point is a local minimum. If all the diagonal elements are negative, a simple zero gradient point is a local maximum. If any two diagonal elements have opposite signs, then a simple zero gradient point must be a saddle point.

A difficulty may occur in the process of computing the Hessian determinant. It may happen that a diagonal element in a certain column overlaps 0, but there is an element lower down in the column that does not contain 0. Whenever this occurs, this implies that any simple zero gradient point in the subbox is a saddle point. (The determinant computation can be completed by imitating the process of computing Jacobian determinants. That is, the diagonal element is replaced by a more suitable element by a row exchange, and then elements below are cleared as usual.) Suppose this difficulty occurs for column  $n-1$ . The determinant of the current matrix equals the product of the first  $n-2$  diagonal elements times the 2 by 2 subdeterminant in diagonal position in columns  $n-1$  and  $n$ . This subdeterminant is negative, so the two  $H$  eigenvalues associated with this subdeterminant have opposite signs. This implies a simple zero gradient point is a saddle point.

If the difficulty occurs earlier than for column  $n - 1$ , imagine performing symmetry preserving row exchange and column exchange operations on the matrix, bringing the suitable element to just below the diagonal position, with another same valued element brought to the right of the diagonal position. Then imagine clearing the elements below the first element and to the right of the second element by appropriate symmetry preserving operations. Again the resulting symmetric matrix has a negative subdeterminant in diagonal position, implying that a simple zero gradient point is a saddle point.

### 13.3 The demo program `extrema`

The program `extrema` locates zero gradient points by following a procedure much like the zero-finding procedure for Problem 12.1, described in Section 12.3.

The first step is to test  $f(\mathbf{x})$  to make certain that all partial derivatives are defined over  $B$ , and this is done via a task queue similar to that described for the zero-finding problem.

Next  $f$  is tested over the boundary of  $B$  to make certain that the  $f$  gradient is nonzero there. After these preliminary tests are made, the general zero-finding procedure of Section 12.3 is used, with the  $n$  functions  $f_1(\mathbf{x}), \dots, f_n(\mathbf{x})$  now being  $\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_n}$ .

At the point in the zero-finding procedure where a Jacobian is computed for a subbox  $B^{(C)}$ , the Hessian computation is made instead, in the manner described in the preceding section, to detect the type of any zero gradient points found in  $B^{(C)}$ .

## Software Exercises K

These exercises are with the demo programs `maxmin` and `extrema`.

1. This exercise and the next demonstrate how `maxmin` displays its results. Call up `maxmin` and obtain to 5 decimal places both the minimum and the maximum of the function  $x + y$  in the square determined by the inequalities  $1 \leq x \leq 10, 1 \leq y \leq 10$ . Here a single point is displayed for either extreme value.
2. Edit the log file to change the function from  $x + y$  to  $x$ , and then call up `maxmin` `maxmin`. This time, because the minimum and maximum computation leads to large connected  $(x, y)$  sets, the program displays only intervals for each variable.
3. Call up `extrema` and find to 10 decimal places the local minima and local maxima of the function  $\cos(x) + \cos(2x) + \cos(3x)$  in the interval  $[-10, 10]$ .

There are 19 local extrema in  $[-10, 10]$ . In cases where many extrema are found, the print file is needed to see them all.

4. The book by Ratschek and Rokne [7] suggests the function

$$4x^2 - 2.1x^4 + x^6/3 + xy - 4y^2 + 4y^4$$

to test minimization programs. Call up `maxmin` to find to 5 decimal places all zero gradient points of this function in the square defined by  $-10 \leq x_1 \leq 10$ ,  $-10 \leq x_2 \leq 10$ .

## Notes and References

- A. Three general references that deal with optimization problems are the books by Fletcher [1], Hansen [2], and Ratschek and Rokne [7].  
 B. Schaefer [8] [9] has treated optimization problems with range arithmetic.

- [1] Fletcher, R., *Practical Methods of Optimization, 2nd Edn*, John Wiley & Sons, New York, Chichester, England, 1987.  
 [2] Hansen, E., *Global Optimization Using Interval Analysis*, Marcel Dekker, Inc., New York, 1992.  
 [3] Hohn, F., *Elementary Matrix Algebra, 3rd Edn*, Macmillan, New York, 1973.  
 [4] Ichida, K. and Fujii, Y., An interval arithmetic method for global optimization, *Computing* **23** (1979), 85–97.  
 [5] Moore, R. E., *Interval Analysis, Prentice-Hall*, Englewood Cliffs, NJ, 1966.  
 [6] Moore, R. E., On computing the range of values of a rational function of  $n$  variables over a bounded region, *Computing* **16** (1976), 1–15.  
 [7] Ratschek, H. and Rokne, J., *New Computer Methods for Global Optimization*, Chichester, England, John Wiley & Sons, New York, 1988.  
 [8] Schaefer, M. J., Precise optimization using range arithmetic, *J. Comp. Appl. Math.* **53** (1994), 341–351.  
 [9] Schaefer, M. J., Verification of constrained minima, *J. Comp. Appl. Math.* **67** (1996), 195–205.  
 [10] Skelboe, S., Computation of rational interval functions, *BIT* **14** (1974), 87–95.

This page intentionally left blank

# Ordinary Differential Equations



Two demo programs solve problems of ordinary differential equations. The program `difsys` is for initial value problems and the program `difbnd` is for boundary-value problems. These problem types are defined in the next two sections.

## 14.1 Introduction

A function  $y(x)$  satisfies an *ordinary differential equation of the first order* if it satisfies an equation of the form

$$y' = f(x, y)$$

More generally, a function  $y(x)$  satisfies an *ordinary differential equation of order  $n$*  if the  $n$ -th derivative of  $y(x)$  is connected with lower order derivatives by the relation

$$y^{(n)} = f(x, y, y', \dots, y^{(n-1)}) \quad (14.1)$$

Thus the equation

$$y'' = (y' + xy)^2 \quad (14.2)$$

is an ordinary differential equation of order 2. The differential equation (14.1) is *linear* if it can be expressed in the form

$$y^{(n)} + g_{n-1}(x)y^{(n-1)} + \dots + g_1(x)y' + g_0(x)y = h(x)$$

Equation (14.2) is not a linear differential equation, but the next equation is:

$$y''' = xy'' - x^2y' + 3y - \sin(x)$$

More generally,  $s$  functions  $y_1(x), y_2(x), \dots, y_s(x)$ , satisfy  $s$  interrelated differential equations of respective orders  $n_1, n_2, \dots, n_s$ , if the following relations hold:

$$\begin{aligned} y_1^{(n_1)} &= f_1(x, y_1, \dots, y_1^{(n_1-1)}, y_2, \dots, y_2^{(n_2-1)}, \dots, y_s, \dots, y_s^{(n_s-1)}) \\ y_2^{(n_2)} &= f_2(x, y_1, \dots, y_1^{(n_1-1)}, y_2, \dots, y_2^{(n_2-1)}, \dots, y_s, \dots, y_s^{(n_s-1)}) \\ &\vdots \\ y_s^{(n_s)} &= f_s(x, y_1, \dots, y_1^{(n_1-1)}, y_2, \dots, y_2^{(n_2-1)}, \dots, y_s, \dots, y_s^{(n_s-1)}) \end{aligned} \quad (14.3)$$

The  $i$ th differential equation is linear if it can be expressed in the form

$$y_i^{(n_i)} + \sum_{j=1}^s \sum_{k=0}^{n_j-1} g_{i,j,k}(x) y_j^{(k)} = h_i(x)$$

For example, the functions  $y_1$  and  $y_2$  may satisfy the interrelated equations

$$\begin{aligned} y_1'' &= y_1^2 + y_1' y_2 \\ y_2'' &= 2y_1 + 3y_2 \end{aligned}$$

The second equation is linear, but not the first.

The set of interrelated differential equations (14.3) involving the functions  $y_1(x), y_2(x), \dots, y_s(x)$  is equivalent to a set of interrelated **first order** differential equations involving another set of functions  $u_1(x), u_2(x), \dots, u_m(x)$ :

$$\begin{aligned} u_1' &= f_1(x, u_1, u_2, \dots, u_m) \\ u_2' &= f_2(x, u_1, u_2, \dots, u_m) \\ &\vdots \\ u_m' &= f_m(x, u_1, u_2, \dots, u_m) \end{aligned} \quad (14.4)$$

Thus for the single equation (14.1), we may set  $u_i(x)$  equal to the  $(i-1)$ th derivative of  $y(x)$ , for  $i = 1, 2, \dots, n$ , to obtain the equations

$$\begin{aligned} u_1' &= u_2 \\ u_2' &= u_3 \\ &\vdots \\ u_{n-1}' &= u_n \\ u_n' &= f(x, u_1, u_2, \dots, u_n) \end{aligned}$$

Similarly, the set of equations (14.3) can be rewritten in terms of  $m$  variables  $u_i$ , where  $m = \sum_{i=1}^s n_i$ , and with  $n_i$  of the new variables associated with  $y_i(x)$ .

From now on, we use the system of equations (14.4) with  $m$  unspecified to designate the set of differential equations under study. If  $\mathbf{u}(x)$  denotes the vector with components  $u_1(x), u_2(x), \dots, u_m(x)$ , the system (14.4) may be written concisely in vector form as

$$\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$$

Here  $\mathbf{f}$  is an  $m$ -component vector function of  $x$  and  $\mathbf{u}$ .

## 14.2 Two standard problems of ordinary differential equations

In general, with any instance of the system  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$ , one finds that over a particular  $x$  interval  $[a, b]$  there are many functions  $\mathbf{u}(x)$  satisfying the equation. To obtain a unique solution, additional conditions are required.

The *initial value problem* for the system  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$  is the determination of a solution  $\mathbf{u}(x)$  in  $[a, b]$  such that  $\mathbf{u}$  equals a specified value  $\mathbf{v}$  at a particular argument  $x_1$  in  $[a, b]$ . Most often  $x_1$  is at the left endpoint  $a$  of  $[a, b]$ , and if this is the case, the initial value problem is to find a function  $\mathbf{u}$  defined in  $[a, b]$  such that

$$\begin{aligned} \mathbf{u}' &= \mathbf{f}(x, \mathbf{u}) \\ \mathbf{u}(a) &= \mathbf{v} \end{aligned} \tag{14.5}$$

The *two-point boundary-value problem* for the system  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$  is the determination of a solution  $\mathbf{u}(x)$  in  $[a, b]$  such that  $\mathbf{u}(a)$  and  $\mathbf{u}(b)$  satisfy the relation

$$g_i(\mathbf{u}(a), \mathbf{u}(b)) = 0 \quad \text{for } i = 1, 2, \dots, m$$

where each function  $g_i$  has the  $2m$  variables  $u_1(a), \dots, u_m(a), u_1(b), \dots, u_m(b)$ . These relations are called the *boundary conditions*, and can be written in the vector form shown next if we understand that vectors always have  $m$  components, the same number of components as the system.

$$\mathbf{g}(\mathbf{u}(a), \mathbf{u}(b)) = \mathbf{0} \tag{14.6}$$

The vector boundary condition (14.6) is *linear* if it can be expressed in the form

$$A\mathbf{u}(a) + B\mathbf{u}(b) = \mathbf{c}$$



where  $A$  and  $B$  are  $m$ -square real matrices, and  $\mathbf{c}$  is an  $m$ -vector. For instance, if  $A = I$ ,  $B = -I$ , and  $\mathbf{c} = \mathbf{0}$ , we have the *periodic* boundary condition

$$\mathbf{u}(a) = \mathbf{u}(b)$$

Linear boundary conditions are often encountered, and a frequent case is the *separated* variety:

$$\begin{aligned}\widehat{A}\mathbf{u}(a) &= \mathbf{c}_a \\ \widehat{B}\mathbf{u}(b) &= \mathbf{c}_b\end{aligned}$$

Here  $\widehat{A}$  is of size  $m_a \times m$  and  $\widehat{B}$  is of size  $m_b \times m$ , with  $m_a + m_b = m$ . Then  $\mathbf{c}_a$  is an  $m_a$ -vector and  $\mathbf{c}_b$  is an  $m_b$ -vector. A particularly simple set of separated linear boundary conditions is given by this set of equations:

$$u_{n_i}(a) = c_{n_i} \quad \text{for } i = 1, \dots, k \quad \text{and} \quad u_{n_i}(b) = c_{n_i} \quad \text{for } i = k + 1, \dots, m$$

Here  $m$  conditions are obtained by fixing in some fashion  $m$  of the  $2m$  components of  $\mathbf{u}(a)$  and  $\mathbf{u}(b)$ . Note that if all  $m$  components of  $\mathbf{u}(a)$  are fixed, or all  $m$  components of  $\mathbf{u}(b)$  are fixed, then the two-point boundary-value problem becomes the initial value problem. Thus the two-point boundary-value problem includes the initial value problem as a special case.

Often a differential equation problem that appears to be different from either of the two basic types can be changed in some way to fit one of the patterns. For example, the motion of a weightless rigid pendulum of length 1, with its attached end at the origin and its free end with a unit mass at the point  $(x, y)$  of the vertical cartesian plane, is determined by the following equations, obtained by calculus of variations methods [2, p. 5]:

$$\frac{d^2x}{dt^2} = Kx; \quad \frac{d^2y}{dt^2} = Ky - g; \quad x^2 + y^2 = 1 \quad (14.7)$$

The constant  $g$  equals the gravitational acceleration. Here a force along the pendulum has  $x$  and  $y$  components  $Kx$  and  $Ky$ , where  $K$  is an unknown time dependent parameter. Suppose the values of  $x$  and  $y$  are desired when an initial velocity  $v_0$  is imparted to the free end of the pendulum at its rest position. The parameter  $K$  can be eliminated by multiplying the first differential equation by  $y$ , the second by  $x$ , and subtracting, to obtain

$$y \frac{d^2x}{dt^2} - x \frac{d^2y}{dt^2} = gx$$

The third equation of (14.7) can be differentiated twice with respect to  $t$  to obtain

$$x \frac{d^2x}{dt^2} + y \frac{d^2y}{dt^2} = - \left( \frac{dx}{dt} \right)^2 - \left( \frac{dy}{dt} \right)^2$$

When these two equations are solved for  $d^2x/dt^2$  and  $d^2y/dt^2$ , we obtain these equations and initial conditions:

$$\begin{aligned}\frac{d^2x}{dt^2} &= gxy - x \left[ \left( \frac{dx}{dt} \right)^2 + \left( \frac{dy}{dt} \right)^2 \right] \\ \frac{d^2y}{dt^2} &= -gx^2 - y \left[ \left( \frac{dx}{dt} \right)^2 + \left( \frac{dy}{dt} \right)^2 \right] \\ x(0) &= 0; \quad \frac{dx(0)}{dt} = v_0; \quad y(0) = -1; \quad \frac{dy(0)}{dt} = 0\end{aligned}\tag{14.8}$$

If we identify  $x$ ,  $dx/dt$ ,  $y$ ,  $dy/dt$ , and  $t$  with the variables  $u_1$ ,  $u_2$ ,  $u_3$ ,  $u_4$ , and  $x$ , we obtain the initial value problem

$$\begin{aligned}u_1' &= u_2 \\ u_2' &= g u_1 u_3 - u_1 [u_2^2 + u_4^2] \\ u_3' &= u_4 \\ u_4' &= -g u_1^2 - u_3 [u_2^2 + u_4^2] \\ u_1(0) &= 0; \quad u_2(0) = v_0; \quad u_3(0) = -1; \quad u_4(0) = 0\end{aligned}$$

As another example, consider the eigenvalue problem of a vibrating string:

$$y'' + \lambda y = 0$$

The eigenvalues  $\lambda$  are sought such that there is a solution function  $y(x)$  in  $[0, \pi]$  satisfying the boundary conditions

$$y(0) = y(\pi) = 0 \quad \text{and} \quad y'(0) = 1$$

The condition  $y'(0) = 1$  is present to “normalize” the solution function. (The solution to this problem is  $y = \frac{1}{\sqrt{\lambda}} \sin(\sqrt{\lambda}x)$ , for  $\lambda = 1^2, 2^2, 3^2, \dots$ ) Here if we identify  $u_1$ ,  $u_2$ , and  $u_3$  with  $y$ ,  $y'$ , and  $\lambda$ , we have a two-point boundary-value problem with the differential equations

$$\begin{aligned}u_1' &= u_2 \\ u_2' &= -u_3 u_1 \\ u_3' &= 0\end{aligned}$$

and the separated boundary conditions

$$u_1(0) = 0$$

$$u_2(0) = 1$$

$$u_1(\pi) = 0$$

### 14.3 Difficulties with the initial value problem

There are certain difficulties that may be encountered in treating the initial value problem numerically. Take the case where  $m$ , the number of functions, is 1, and the problem is to find in  $[a, b]$  a solution to

$$u_1' = f_1(x, u_1)$$

satisfying the initial condition

$$u_1(a) = v_1$$

Sometimes a solution in  $[a, b]$  is not possible, because the solution function  $u_1(x)$  tends to infinity at some point within the designated interval. For instance, suppose in the  $x$  interval  $[0, 2]$  we wanted a solution to the problem

$$\begin{aligned} u_1' &= u_1^2 \\ u_1(0) &= 1 \end{aligned} \tag{14.9}$$

The nonlinear differential equation is separable and the solution is easily found to be

$$u_1(x) = \frac{1}{1-x}$$

Because  $\lim_{x \rightarrow 1^-} = \infty$ , an accurate numerical solution cannot reach the midpoint of the  $[0, 2]$  interval. Thus we must be prepared to occasionally encounter solutions that grow without bound as  $x$  approaches some point  $c$  within  $[a, b]$ . For such cases an attempt to numerically approximate the solution accurately throughout  $[a, b]$  would be futile.

A difficulty of a different variety is illustrated by the problem of finding in  $[0, 1]$  a solution to the problem

$$\begin{aligned} u_1' &= u_1^{\frac{1}{3}} \\ u_1(0) &= 0 \end{aligned} \tag{14.10}$$

The differential equation, again nonlinear, is separable, and it is not difficult to verify that there are an infinite number of solutions to this problem. First note

that  $u_1 = 0$  and  $u_1 = [\frac{2}{3}x]^{\frac{3}{2}}$  are two distinct solutions to the problem. These two solutions can be used to construct an infinite family of solutions. We use the parameter  $t$  to designate the following different members of the family:

$$u_1(x; t) = \begin{cases} 0 & \text{for } 0 \leq x \leq t \\ [\frac{2}{3}(x-t)]^{\frac{3}{2}} & \text{for } t < x \leq 1 \end{cases}$$

The parameter  $t$  may assume any value in  $[0, 1]$ . When  $t = 1$ ,  $u_1$  is the 0 solution; when  $t = 0$ ,  $u_1$  is  $[\frac{2}{3}x]^{\frac{3}{2}}$ ; and when  $t$  is between 0 and 1,  $u_1$  combines features of the two solutions. Note that when  $t$  is in  $(0, 1)$ , the two parts of the solution join without introducing a discontinuity in  $u_1$  or its derivative.

Multiple solutions to the initial value problem (14.5) can be eliminated by imposing restrictions on  $\mathbf{f}(x, \mathbf{u})$ . Note that the function  $u_1^{1/3}$  of (14.10) does not have a derivative with respect to  $u_1$  anywhere along the line  $u_1 = 0$ .

Suppose for the initial value problem (14.10) we designate a *containment box*  $B_{[a,b],M}$  of  $m+1$  dimensions, defined by the relations

$$a \leq x \leq b \quad \text{and} \quad [v_i - M \leq u_i \leq v_i + M] \quad \text{for } i = 1, \dots, m$$

The theory of ordinary differential equations shows that if  $\mathbf{f}(x, \mathbf{u})$  has a bounded partial derivative with respect to all its  $\mathbf{u}$  variables within this box, then multiple solutions to the initial value problem are not possible if  $\mathbf{u}(x)$  stays within the box. The two difficulties of this section are taken into account with

**Solvable Problem 14.1** Given the initial value problem  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$ ,  $\mathbf{u}(a) = \mathbf{v}$ , where  $\mathbf{f}(x, \mathbf{u})$  is an elementary function differentiable with respect to its  $\mathbf{u}$  variables over a containment box  $B_{[a,b],M}$ , for any argument  $c$  in  $[a, b]$ , find  $\mathbf{u}(c)$  to  $k$  decimal places, or else indicate that the solution  $\mathbf{u}(x)$  exits the containment box before  $x$  reaches  $c$ .

## 14.4 Linear differential equations

Suppose for an initial value problem, all the individual differential equation components of  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$  are linear, that is, each equation can be written in the form

$$u'_i + \sum_{j=1}^m g_{ij}(x)u_j = h_i(x)$$

The vector function  $\mathbf{f}(x, \mathbf{u})$  is then called *linear*. All the components of a linear  $\mathbf{f}(x, \mathbf{u})$  have partial derivatives with respect to each component of  $\mathbf{u}$ . If all functions  $g_{i,j}$  and  $h_i$  are elementary and defined in the interval  $[a, b]$  of the initial

value problem, then all the solution components are bounded, and we do not have to deal with the possibility of a solution component tending to infinity as  $x$  approaches some point  $c$  inside  $[a, b]$ . For we can find a positive constant  $p$  bounding the magnitude of all these elementary functions in  $[a, b]$ . We have then

$$\begin{aligned} |u'_i| &\leq \sum_{j=1}^m |g_{ij}(x)||u_j| + |h_i(x)| \\ &\leq \sum_{j=1}^m p|u_j| + p \end{aligned}$$

A component of the solution  $\mathbf{u}(x)$  can grow in  $[a, b]$  no faster than the corresponding component of the solution to the following initial value problem:

$$\widehat{u}'_i = p\widehat{u}_i + p, \quad \widehat{u}_i(a) = |v_i| \quad i = 1, 2, \dots, m$$

The solution  $\widehat{u}(x)$  to this second problem grows exponentially but is bounded in  $[a, b]$ . Accordingly, we can dispense with a containment box for  $\mathbf{u}(x)$ , and we have

**Solvable Problem 14.2** Given the initial value problem  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$ ,  $\mathbf{u}(a) = \mathbf{v}$ , where  $\mathbf{f}(x, \mathbf{u})$  is a linear elementary function defined for  $x$  in  $[a, b]$ , find  $\mathbf{u}(c)$  to  $k$  decimal places, where  $c$  is any  $x$  value in  $[a, b]$ .

## 14.5 Solving the initial value problem by power series

In Chapter 6 we computed definite integrals of elementary functions using power series. Power series methods are useful also for solving differential equations. However, to apply such methods, the function  $\mathbf{f}(x, \mathbf{u})$  must be differentiable to arbitrary order with respect to all of its variables. Although this condition commonly holds, this is a stronger condition on  $\mathbf{f}(x, \mathbf{u})$  than what either of the two preceding solvable problems require.

Assume now that this stronger condition holds. If  $\mathbf{u}$  is the solution to the initial value problem, then for each component  $u_i$ , we have the differential equation  $u'_i = f_i(x, u_1, \dots, u_m)$ . This equation can be differentiated any number of times with respect to  $x$ , because all partial derivatives of all functions  $f_i$  exist, so  $u_i$  is infinitely differentiable at any  $x$  point in  $[a, b]$ . So the  $\mathbf{u}$  components can be approximated by Taylor polynomials over small subintervals of  $[a, b]$ .

The power series method to be described has some similarity to the power series method for computing definite integrals, described in Chapter 6, but is more complicated than that method. We will suppose that we have advanced our  $\mathbf{u}(x)$  approximation from  $x = a$  up to  $x = x_k$ , and have obtained the interval

$\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}$  for  $\mathbf{u}(x_k)$ . Here we are using vector notation to represent the  $m$  components of  $\mathbf{u}(x_k)$ . The halfwidth vector  $\mathbf{w}^{(k)}$ , called the *global error vector*, defines a halfwidth bound for the solution  $\mathbf{u}$  at  $x_k$ , that is,  $\mathbf{u}(x_k)$  is somewhere within the interval  $\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}$ . (At the start of the approximation process,  $x_1 = a$  and  $\mathbf{m}^{(1)} \pm \mathbf{w}^{(1)} = \mathbf{v} \pm \mathbf{0}$ , so the global error vector then is  $\mathbf{0}$ .)

At the series expansion point  $(x_k, \mathbf{m}^{(k)})$  we try to obtain a power series for the solution  $\mathbf{u}(x)$  that bounds the solution in the interval  $[x_k, x_k + h]$ . The step width  $h$  is adjusted so that various requirements to be described are satisfied. Generally the step width of the preceding interval  $[x_{k-1}, x_k]$  is the first  $h$  value tried, but occasionally this initial  $h$  value is doubled to test whether conditions have changed to allow a larger step width. If any of the required conditions are not met,  $h$  is halved, and the attempt repeated. In the starting  $x$  interval  $[a, a + h]$ , the first trial  $h$  is large, for example 0.5.

The bounding series for  $\mathbf{u}$  in powers of  $x - x_k$  is computed up to the terms in  $(x - x_k)^n$ , where  $n$  is reasonably large, such as 12 or higher. The computation for the terms of  $\mathbf{u}$  are made by a recursive procedure. Initially series variables are set to

$$x = (x_k \pm h) + (1 \pm 0)(x - x_k)$$

$$\mathbf{u} = (\mathbf{m}^{(k)} \pm \widehat{\mathbf{w}})$$

The halfwidth vector  $\widehat{\mathbf{w}}$  has components larger than or identical to corresponding components of the global error vector  $\mathbf{w}^{(k)}$ . The first step is to choose the vector  $\widehat{\mathbf{w}}$  so that for any  $x$  in  $[x_k, x_k + h]$  it is certain that  $\mathbf{u}(x)$  is contained within the vector bounding interval  $\mathbf{m}^{(k)} \pm \widehat{\mathbf{w}}$ .

For the chosen step width  $h$ , whether a halfwidth vector  $\widehat{\mathbf{w}}$  is satisfactory is determined by the outcome of a degree 0 series evaluation for  $\mathbf{f}$ :

$$\mathbf{f} = (\mathbf{f}_0 \pm \widehat{\mathbf{w}}_0)$$

The containment condition is satisfied if a halfwidth bound vector for  $\mathbf{u}$  in  $[x_k, x_k + h]$ , calculated using the global error vector  $\mathbf{w}^{(k)}$  and the largest possible rates of increase for  $\mathbf{u}$  components, obtained from  $\mathbf{f}$ , is within the vector  $\widehat{\mathbf{w}}$ :

$$\mathbf{w}^{(k)} + h(|\mathbf{f}_0| + \widehat{\mathbf{w}}_0) < \widehat{\mathbf{w}} \tag{14.11}$$

A value for  $\widehat{\mathbf{w}}$  is desired with components as small as possible. A simple method of determining a suitable  $\widehat{\mathbf{w}}$  is to set  $\widehat{\mathbf{w}}$  initially to  $\mathbf{w}^{(k)}$ , evaluate  $\mathbf{f}$ , and if the vector inequality (14.11) fails for component  $j$ , then component  $j$  of  $\widehat{\mathbf{w}}$  is increased to match component  $j$  of  $\mathbf{w}^{(k)} + h(|\mathbf{f}_0| + \widehat{\mathbf{w}}_0)$ , times a factor slightly above 1, for example 1.1. The test (14.11) then is repeated after  $\mathbf{f}$  is recalculated. After a fixed number of these attempts to adjust  $\widehat{\mathbf{w}}$  fail, the step width  $h$  is halved, and the whole process restarted.

After a suitable  $\widehat{\mathbf{w}}$  is determined, the next step is to determine all the terms of the  $\mathbf{u}$  series. Because  $\mathbf{u}' = \mathbf{f}$ , a term  $(\mathbf{f}_k \pm \mathbf{w}_k)(x - x_k)^k$  determined for the  $\mathbf{f}$  series becomes the term  $(\frac{1}{k+1}\mathbf{f}_k \pm \frac{1}{k+1}\mathbf{w}_k)(x - x_k)^{k+1}$  for  $\mathbf{u}$ . So the degree 0 term for  $\mathbf{f}$  allows  $\mathbf{u}$  to be changed to

$$\mathbf{u} = (\mathbf{m}^{(k)} \pm \widehat{\mathbf{w}}) + (\mathbf{f}_0 \pm \widehat{\mathbf{w}}_0)(x - x_k)$$

The computation for the  $\mathbf{f}$  series now is carried further to obtain the degree 1 term of  $\mathbf{f}$ , which is used to fix the degree 2 term of  $\mathbf{u}$ . This cyclic procedure, of computing the degree  $k$  term of  $\mathbf{f}$  and using it to fix the degree  $k + 1$  term of the  $\mathbf{u}$  series, continues until all the required terms of  $\mathbf{u}$  are obtained.

If the global error vector  $\mathbf{w}^{(k)}$  for  $\mathbf{u}(x_k)$  were  $\mathbf{0}$  (as it is for the first step), then by the series remainder formula (6.2), we would have

$$\mathbf{u}(x_k + h) = \mathbf{m}^{(k)} + h\mathbf{f}_0 + \frac{h^2}{2}\mathbf{f}_1 + \cdots + \frac{h^{n-1}}{n-1}\mathbf{f}_{n-2} + \frac{h^n}{n}(\mathbf{f}_{n-1} \pm \widehat{\mathbf{w}}_{n-1}) \quad (14.12)$$

The vector  $\frac{h^n}{n}\widehat{\mathbf{w}}_{n-1}$  is the *local error vector* over the interval  $[x_k, x_k + h]$ . For each component of  $\mathbf{u}$ , the local errors accumulate and add to the global error as the solution is constructed from  $a$  toward  $b$ . The rate of growth of global error due to the local error is defined by the vector  $\frac{h^{n-1}}{n}\widehat{\mathbf{w}}_{n-1}$ , and if any component exceeds a preset bound (which would depend on the number of correct decimal places wanted), then  $h$  is halved, and the whole process restarted.

The global error  $\mathbf{w}^{(k+1)}$  equals the local error over  $[x_k, x_k + h]$  plus another halfwidth vector, the *global error carryover*, which equals  $\mathbf{w}^{(k)}$  plus other terms. The series terms of (14.12) that do not have a halfwidth part are calculated at the point defined by  $x = x_k$  and  $\mathbf{u} = \mathbf{m}^{(k)}$ , but the solution  $\mathbf{u}(x_k)$  can be anywhere in the interval  $\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}$ . To take this into account, all  $\mathbf{u}$  series terms of degree less than  $n$  are assigned a halfwidth to reflect the effect of the global error  $\mathbf{w}^{(k)}$ , and this requires a new computation of these terms. For this second calculation, the variables are set initially to

$$x = (x_k \pm 0) + (1 \pm 0)(x - x_k)$$

$$\mathbf{u} = (\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)})$$

and the cyclic process described previously is repeated, but this time it ends when the  $\mathbf{u}$  term of degree  $n - 1$  is obtained. The degree 1 term given for  $x$  is needed to make the computation yield the same midpoints as before, but with changed halfwidths. We distinguish these changed halfwidths by using  $\mathbf{w}$  in place of  $\widehat{\mathbf{w}}$ , and after the newly calculated terms are inserted in (14.12), we obtain

$$\mathbf{u}(x_k + h) = (\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}) + h(\mathbf{f}_0 \pm \mathbf{w}_0) + \frac{h^2}{2}(\mathbf{f}_1 \pm \mathbf{w}_1) + \cdots + \frac{h^n}{n}(\mathbf{f}_{n-1} \pm \widehat{\mathbf{w}}_{n-1}) \quad (14.13)$$

At  $x = x + h$  we have  $\mathbf{u}(x_k + h) = \mathbf{m}^{(k+1)} \pm \mathbf{w}^{(k+1)}$ . Equation (14.13) now shows that

$$\mathbf{m}^{(k+1)} = \mathbf{m}^{(k)} + h\mathbf{f}_0 + \frac{h^2}{2}\mathbf{f}_1 + \dots + \frac{h^n}{n}\mathbf{f}_{n-1} \tag{14.14}$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + h\mathbf{w}_0 + \frac{h^2}{2}\mathbf{w}_1 + \dots + \frac{h^{n-1}}{n-1}\mathbf{w}_{n-2} + \frac{h^n}{n}\widehat{\mathbf{w}}_{n-1} \tag{14.15}$$

The global error carryover is

$$\mathbf{w}^{(k)} + h\mathbf{w}_0 + \frac{h^2}{2}\mathbf{w}_1 + \dots + \frac{h^{n-1}}{n-1}\mathbf{w}_{n-2} \tag{14.16}$$

We can calculate  $\mathbf{u}(x)$  for any  $x$  in  $[x_k, x_k + h]$  by using (14.14) and (14.15), with  $h$  replaced by the value of  $x - x_k$ .

As an example, let us apply the proposed system to the test case:

$$\begin{aligned} \mathbf{u}' &= -\mathbf{u} \\ \mathbf{u}(0) &= \mathbf{v} \end{aligned} \tag{14.17}$$

The number of components  $m$  can be any positive integer, and the solution is  $\mathbf{u} = e^{-x}\mathbf{v}$ . Suppose we have carried the numerical solution up to the point  $x = x_k$ , obtaining  $\mathbf{u}(x_k) = \mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}$ . Because  $\mathbf{f} = -\mathbf{u}$ , the  $\mathbf{u}$  series term in  $(x - x_k)^j$  has the coefficient  $(-1)^j \frac{1}{j!}(\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)})$ . Therefore we obtain for  $\mathbf{u}(x_k + h) = \mathbf{m}^{(k+1)} \pm \mathbf{w}^{(k+1)}$  the values

$$\begin{aligned} \mathbf{m}^{(k+1)} &= \mathbf{m}^{(k)} - h\mathbf{m}^{(k)} + \frac{h^2}{2!}\mathbf{m}^{(k)} - \frac{h^3}{3!}\mathbf{m}^{(k)} + \dots + (-1)^n \frac{h^n}{n!}\mathbf{m}^{(k)} \\ \mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} + h\mathbf{w}^{(k)} + \frac{h^2}{2!}\mathbf{w}^{(k)} + \frac{h^3}{3!}\mathbf{w}^{(k)} + \dots + \frac{h^{n-1}}{(n-1)!}\mathbf{w}^{(k)} + \frac{h^n}{n!}\widehat{\mathbf{w}} \end{aligned}$$

(By the rules of interval arithmetic,  $c(\mathbf{m} \pm \mathbf{w}) = c\mathbf{m} \pm |c|\mathbf{w}$ .) We see that  $\mathbf{m}^{(k+1)}$  is nearly  $e^{-h}\mathbf{m}^{(k)}$ , a satisfactory result, but the global error carryover is nearly  $e^h\mathbf{w}^{(k)}$ . It will not be possible to compute  $\mathbf{u}(x)$  to  $k$  decimal places for a large interval  $[a, b]$ , with the global error growing exponentially this way. It is clear that an improvement in the computation of  $\mathbf{w}^{(k+1)}$  is needed.

## 14.6 Degree 1 interval arithmetic

In this section we introduce a variety of interval arithmetic that is helpful in obtaining accurate differential equation solutions.

When performing any of the four operations  $+$ ,  $-$ ,  $\times$ ,  $\div$  on intervals, an overly large result interval may be obtained if the operands are related in some way.



Each of the four interval arithmetic relations (2.3)–(2.6) given in Chapter 2 is obtained under the assumption that the two operands are independent, so that if one operand takes a certain value within its interval, this has no influence on the other operand. If this assumption is not valid, a narrower result interval may be possible.

Two simple examples of this are the computations of  $x - x$  and  $xy - x$  for intervals  $x$  and  $y$ . For instance, if  $x = 2 \pm 1$  and  $y = 3 \pm 1$ , then according to the interval equation for subtraction, we have  $x - x = 0 \pm 2$ , when the answer should be  $0 \pm 0$ . Similarly, according to the interval equations for multiplication and subtraction, we have

$$\begin{aligned} xy - x &= (2 \pm 1) \times (3 \pm 1) - (2 \pm 1) = [2 \cdot 3 \pm (1 \cdot 3 + 2 \cdot 1 + 1 \cdot 1)] - (2 \pm 1) \\ &= (6 \pm 6) - (2 \pm 1) = 4 \pm 7 \end{aligned}$$

As  $x$  varies in the interval  $[1, 3]$  and  $y$  varies in the interval  $[2, 4]$ , the function  $xy - x$  varies in  $[1, 9]$ . (These bounds for  $xy - x$  are readily found with the program `maxmin`.) So using the  $xy - x$  midpoint value of 4, the best possible interval result is  $4 \pm 5$ .

Suppose a certain interval computation depends on  $N$  initial quantities having interval values. Imagine the  $i$ th such interval quantity,  $m_i \pm w_i$ , defining a variable  $v_i$  in an interval  $[m_i - w_i, m_i + w_i]$ . Suppose the computation can be represented as a certain elementary function  $f$  of the variables  $v_1, \dots, v_N$ . When we do a formal interval computation for  $f$ , we obtain an interval result which may be considered a degree 0 series expansion of  $f$  in terms of the variables  $v_i$ . If we compute  $f$  as a degree 1 series expansion of the variables  $v_i$ , then we obtain interval bounds for all of  $f$ 's degree 1 terms. This type of computation occurred in Chapter 6, where we needed to bound a high degree term of an integrand function  $f(x)$  expressed as a power series. To obtain these interval bounds, we use formal interval arithmetic, and each series variable  $v_i$  is set to

$$v_i = (m_i \pm w_i) + (1 \pm 0)(v_i - m_i)$$

The computation  $f$  proceeds by generating a degree 1 series representation for each successive quantity occurring in the calculation. In general an intermediate result  $q$  will have the form

$$q = \alpha + \sum_k \beta_k (v_k - m_k)$$

Here the interval  $\alpha$  defines  $q$ 's degree 0 term, and an interval coefficient  $\beta_k$  is present to define a degree 1 term for each variable  $v_k$  on which  $q$  depends. The degree 0 term gives  $q$ 's value as computed by ordinary interval arithmetic. The degree 1 terms give interval bounds for  $q$ 's partial derivatives with respect to the

various variables  $v_i$ . The quantity  $q$  may be viewed as a function  $q(v_1, \dots, v_N)$  of the  $N$  variables  $v_1, \dots, v_N$ . The domain of this function is the  $N$ -space box  $B$  defined by the  $N$  intervals  $m_i \pm w_i$ , and the function  $q$  has the Taylor series expansion

$$q(m_1, \dots, m_N) + \sum_k \frac{\partial q(m_1, \dots, m_N)}{\partial v_k} (v_k - m_k) + \dots$$

with the series expansion point  $(m_1, \dots, m_N)$  in  $B$ , so that  $q(m_1, \dots, m_N)$  is the midpoint of  $q$ 's interval  $\alpha$ , and  $\partial q(m_1, \dots, m_N) / \partial v_k$  is the midpoint of the interval  $\beta_k$ . We now make use of equation (6.2). For any point  $(v_1, \dots, v_N)$  within  $B$ , we have the interval relation

$$q(v_1, \dots, v_N) = \text{midpoint } \alpha + \sum_k \beta_k (v_k - m_k)$$

The maximum absolute value of  $(v_k - m_k)$  in  $B$  is  $w_k$ , so we obtain a second halfwidth for  $q$ 's interval, namely

$$\text{halfwidth } q = \sum_k |\beta_k| w_k \tag{14.18}$$

Here  $|\beta_k|$  equals  $|\text{midpoint } \beta_k| + \text{halfwidth } \beta_k$ . Thus we always have two halfwidths for  $q$ , one by ordinary interval arithmetic (which we will call the degree 0 halfwidth), and one by using equation (14.18) (which we will call the degree 1 halfwidth). With formal interval arithmetic, the degree 1 halfwidth for  $q$  is not automatically computed as is the degree 0 halfwidth, but whenever it is computed, it can replace  $q$ 's degree 0 halfwidth if it is smaller.

Repeating the computations given earlier for  $x - x$  and  $xy - x$  in degree 1 format, if  $x$  is  $v_1$  and  $y$  is  $v_2$ , we have

$$\begin{aligned} x - x &= [(2 \pm 1) + (1 \pm 0) \cdot (v_1 - 2)] - [(2 \pm 1) + (1 \pm 0) \cdot (v_1 - 2)] \\ &= [(0 \pm 2) + (0 \pm 0) \cdot (v_1 - 2)] \end{aligned}$$

and

$$\begin{aligned} xy - x &= [(2 \pm 1) + (1 \pm 0) \cdot (v_1 - 2)] \cdot [(3 \pm 1) + (1 \pm 0) \cdot (v_2 - 3)] \\ &\quad - [(2 \pm 1) + (1 \pm 0) \cdot (v_1 - 2)] \\ &= [(6 \pm 6) + (3 \pm 1) \cdot (v_1 - 2) + (2 \pm 1) \cdot (v_2 - 3)] \\ &\quad - [(2 \pm 1) + (1 \pm 0) \cdot (v_1 - 2)] \\ &= [(4 \pm 7) + (2 \pm 1) \cdot (v_1 - 2) + (2 \pm 1) \cdot (v_2 - 3)] \end{aligned}$$

For  $x - x$  we obtain a degree 1 halfwidth of  $0 \cdot 1 = 0$ , and for  $xy - x$  a degree 1 halfwidth of  $3 \cdot 1 + 3 \cdot 1 = 6$ . These halfwidths are an improvement over the degree 0 halfwidths of 2 and 7, respectively. We see that now we obtain the correct halfwidth for  $x - x$ , and an improved but not optimum halfwidth for  $xy - x$ .

In general, degree 1 interval arithmetic requires more computation than the ordinary variety, but the halfwidths it obtains often are significantly better for cases where the computation requires many additions and subtractions of intermediate results.

The degree 1 halfwidth can also be *larger* than the degree 0 halfwidth. For instance, using the  $x$  and  $y$  intervals that have appeared in these examples, for  $x \cdot y$  we obtain

$$\begin{aligned} x \cdot y &= [(2 \pm 1) + (1 \pm 0) \cdot (v_1 - 2)] \cdot [(3 \pm 1) + (1 \pm 0) \cdot (v_2 - 3)] \\ &= [(6 \pm 6) + (3 \pm 1) \cdot (v_1 - 2) + (2 \pm 1) \cdot (v_2 - 3)] \end{aligned}$$

For  $x \cdot y$  we obtain a degree 1 halfwidth of  $4 \cdot 1 + 3 \cdot 1 = 7$ , which is larger than the degree 0 halfwidth of 6. The situation is similar for the division  $x/y$ .

One can imagine obtaining a “degree 2 halfwidth” also. Here we have the same representation for all starting constants as before, but any quantity  $q$  computed in terms of these constants is calculated up to degree 2. Thus  $q$  has the form

$$q = \alpha + \sum_k \beta_k (v_i - m_k) + \sum_{k,l} \gamma_{kl} (v_k - m_k)(v_l - m_l)$$

Here  $\gamma_{kl}$ , like  $\beta_k$ , is an interval. Again making use of equation (6.2), and using similar reasoning as before,  $q$ 's degree 2 halfwidth is

$$\text{halfwidth } q = \sum_k |\text{midpoint } \beta_k| w_k + \sum_{k,l} |\gamma_{kl}| w_k w_l$$

For the example  $xy - x$  we obtain now

$$\begin{aligned} xy - x &= [(2 \pm 1) + (1 \pm 0) \cdot (v_1 - 2)] \cdot [(3 \pm 1) + (1 \pm 0) \cdot (v_2 - 3)] \\ &\quad - [(2 \pm 1) + (1 \pm 0) \cdot (v_1 - 2)] \\ &= [(4 \pm 7) + (2 \pm 1) \cdot (v_1 - 2) + (2 \pm 1) \cdot (v_2 - 3) \\ &\quad + (1 \pm 0)(v_1 - 2)(v_2 - 3)] \end{aligned}$$

The degree 2 halfwidth is  $[(2 \cdot 1) + (2 \cdot 1)] + [1 \cdot 1 \cdot 1] = 5$ , which is the optimum value.

If there are  $N$  initial constants, the number of terms an end result must carry to obtain a degree 2 halfwidth is proportional to  $N^2$ , whereas the number of terms for a degree 1 halfwidth is proportional only to  $N$ . This increased computational cost makes it difficult to find applications where it is worthwhile computing degree 2 halfwidths instead of degree 1 halfwidths.

### 14.7 An improved global error

By using the type of interval arithmetic introduced in the preceding section, the global error carryover computation can be greatly improved. The degree 1 halfwidth computation (14.18) depends only on the  $\mathbf{u}$  intervals. To obtain an improved halfwidth, we set  $\mathbf{u}$  components to

$$u_i = m_i^{(k)} \pm w_i^{(k)} + (1 \pm 0)(u_i - m_i^{(k)}) \quad i = 1, \dots, m$$

This can be expressed compactly in matrix-vector notation as

$$\mathbf{u} = \mathbf{m}^{(k)} \pm \mathbf{w}^{(k)} + (I \pm O)(\mathbf{u} - \mathbf{m}^{(k)}) \tag{14.19}$$

All  $\mathbf{f}$  computations are now made to degree 1 with respect to  $\mathbf{u}$ , so this changes the coefficient of the  $\mathbf{f}$  series term in  $(x - x_k)^j$  from  $\mathbf{f}_j \pm \mathbf{w}_j$  to

$$\mathbf{f}_j \pm \mathbf{w}_j + (A_j \pm B_j)(\mathbf{u} - \mathbf{m}^{(k)})$$

where  $A_j$  and  $B_j$  are matrices. Instead of the expression (14.13) for  $\mathbf{u}(x_k + h)$ , we now obtain an expression which is degree 1 with respect to  $\mathbf{u}$ , with a degree 0 part identical to the right side of (14.13), and with a degree 1 part equal to

$$\begin{aligned} & \left[ (I \pm O) + h(A_0 \pm B_0) + \frac{h^2}{2}(A_1 \pm B_1) + \dots + \frac{h^{n-1}}{n-1}(A_{n-2} \pm B_{n-2}) \right] (\mathbf{u} - \mathbf{m}^{(k)}) \\ & = (A \pm B)(\mathbf{u} - \mathbf{m}^{(k)}) \end{aligned} \tag{14.20}$$

where the matrices  $A$  and  $B$  are determined by the equations

$$\begin{aligned} A &= I + hA_0 + \frac{h^2}{2}A_1 + \dots + \frac{h^{n-1}}{n-1}A_{n-2} \\ B &= hB_0 + \frac{h^2}{2}B_1 + \dots + \frac{h^{n-1}}{n-1}B_{n-2} \end{aligned}$$

We replace the global error carryover (14.16), which is degree 0 with respect to  $\mathbf{u}$ , with the degree 1 halfwidth vector

$$(|A| + B)\mathbf{w}^{(k)}$$

There is no change in the midpoint value (14.14) for  $\mathbf{m}^{(k+1)}$ , and also no change in the local error contribution to  $\mathbf{w}^{(k+1)}$ .

For the test problem (14.17), because  $\mathbf{f} = -\mathbf{u}$ , we find the coefficient of the  $\mathbf{u}$  series term in  $(x - x_k)^j$  equals

$$(-1)^j \frac{1}{j!} [(\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}) + (I \pm O)(\mathbf{u} - \mathbf{m}^{(k)})]$$

so that we obtain

$$A = I - hI + \frac{h^2}{2!}I - \cdots + (-1)^{n-1} \frac{h^{n-1}}{(n-1)!}I$$

$$B = O$$

The global error carryover is now very nearly  $e^{-h\mathbf{w}^{(k)}}$ , and there no longer is difficulty obtaining accurate values of the solution over wide intervals  $[a, b]$ .

A second instructive test problem is the differential equation  $y'' + y = 0$ , with initial conditions  $y(0) = v_1$ ,  $y'(0) = v_2$ , or, equivalently, setting  $u_1 = y$  and  $u_2 = y'$ ,

$$\begin{bmatrix} u_1' \\ u_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}; \quad \begin{bmatrix} u_1(0) \\ u_2(0) \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \quad (14.21)$$

The solution is

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \cos x & \sin x \\ -\sin x & \cos x \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Again taking  $\mathbf{u}(x_k)$  to be  $\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}$ , and using degree 1 interval arithmetic, this time we find the coefficient of the  $\mathbf{u}$  series term in  $(x - x_k)^j$  is

$$\frac{1}{j!} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}^j [(\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}) + (I \pm O)(\mathbf{u} - \mathbf{m}^{(k)})]$$

If we set the matrix  $S$  equal to  $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ , for this problem we find

$$\mathbf{m}^{(k+1)} = \left[ I + hS + \frac{h^2}{2!}S^2 + \frac{h^3}{3!}S^3 + \cdots + \frac{h^n}{n!}S^n \right] \mathbf{m}^{(k)}$$

Note that  $S^2 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$ , and  $S^4 = I$ . We find that very nearly

$$\mathbf{m}^{(k+1)} = \begin{bmatrix} \cos h & \sin h \\ -\sin h & \cos h \end{bmatrix} \mathbf{m}^{(k)} = \begin{bmatrix} m_1^{(k)} \cos h + m_2^{(k)} \sin h \\ -m_1^{(k)} \sin h + m_2^{(k)} \cos h \end{bmatrix}$$

The matrix  $B$  of (14.20) is  $O$ , and the matrix  $A$  is given by

$$A = I + hS + \frac{h^2}{2!}S^2 + \frac{h^3}{3!}S^3 + \cdots + \frac{h^{n-1}}{(n-1)!}S^{n-1}$$

The global error carryover, the  $\mathbf{u}$  degree 1 halfwidth vector  $|A|\mathbf{w}^{(k)}$ , is very nearly

$$\begin{bmatrix} w_1^{(k)} \cos h + w_2^{(k)} \sin h \\ w_1^{(k)} \sin h + w_2^{(k)} \cos h \end{bmatrix}$$

This is not a good result. For instance, if  $w_1^{(k)} = w_2^{(k)}$  the global error carryover vector has components that approximate  $(\sin h + \cos h)w_1^{(k)}$ , which is close to  $(h + 1)w_1^{(k)}$ . After executing  $q$  of these  $h$  steps, the global error components are close to  $(h + 1)^q w_1^{(k)}$ . The global error will grow exponentially for this problem, preventing determination of the solution  $\mathbf{u}$  over a large interval  $[a, b]$ . Another improvement in the treatment of global error carryover is needed.

The major part of the global error carryover is generally the vector  $|A|\mathbf{w}^{(k)}$ . The term  $A\mathbf{w}^{(k)}$  is converted to the form  $|A|\mathbf{w}^{(k)}$  to get positive halfwidths to combine with the  $B\mathbf{w}^{(k)}$  halfwidths. The global error vector  $\mathbf{w}^{(k)}$  defines intervals for the  $\mathbf{u}$  solution components. Let us think of this vector as if it were varying, with its  $i$ th component always never greater than  $w_i^{(k)}$  and never less than  $-w_i^{(k)}$ . Then the vector defines a variation box in  $m$  space, with sides perpendicular to the coordinates. As  $\mathbf{w}^{(k)}$  varies within its allotted box, the vector  $A\mathbf{w}^{(k)}$  varies within an  $m$ -dimensional parallelepiped. The halfwidth vector  $|A|\mathbf{w}^{(k)}$ , with  $i$ th component  $\sum_k |a_{ik}|w_k^{(k)}$ , defines the smallest box containing this parallelepiped. If  $\mathbf{w}^{(k)}$  were to vary within a subregion of its box, then  $A\mathbf{w}^{(k)}$  would vary within the corresponding subregion image.

An improvement in global error carryover is possible by avoiding the conversion of  $A\mathbf{w}^{(k)}$  to  $|A|\mathbf{w}^{(k)}$ . Suppose instead of maintaining the global error as a vector  $\mathbf{w}^{(k)}$ , we maintain it in the form  $C^{(k)}\mathbf{w}^{(k)}$ , using both a matrix  $C^{(k)}$  and a vector  $\mathbf{w}^{(k)}$ . We take  $C^{(1)} = I$  at the starting point  $x_1 = a$ . The global error carryover is now

$$\begin{aligned} (A \pm B)C^{(k)}\mathbf{w}^{(k)} &= AC^{(k)}\mathbf{w}^{(k)} \pm BC^{(k)}\mathbf{w}^{(k)} \\ &= AC^{(k)}\mathbf{w}^{(k)} + \mathbf{w}_B \end{aligned}$$

where  $\mathbf{w}_B = B|C^{(k)}|\mathbf{w}^{(k)}$ . For the global error  $C^{(k+1)}\mathbf{w}^{(k+1)}$ , we take  $C^{(k+1)}$  equal to  $AC^{(k)}$  and  $\mathbf{w}^{(k+1)}$  equal to  $\mathbf{w}^{(k)}$  plus a correction vector  $\mathbf{w}^+$  large enough to account for  $\mathbf{w}_B$  plus the local error  $\frac{h^n}{n}\widehat{\mathbf{w}}_{n-1}$ . If we set  $\mathbf{w}' = \mathbf{w}_B + \frac{h^n}{n}\widehat{\mathbf{w}}_{n-1}$ , we can continue the global error carryover computation, obtaining

$$AC^{(k)}\mathbf{w}^{(k)} + \mathbf{w}' = C^{(k+1)}\mathbf{w}^{(k)} + \mathbf{w}' = C^{(k+1)}[\mathbf{w}^{(k)} + (C^{(k+1)})^{-1}\mathbf{w}']$$

so the correction vector  $\mathbf{w}^+$  may be taken as  $|(C^{(k+1)})^{-1}|\mathbf{w}'$ .

With the test problem (14.21), we find that the vector  $\mathbf{w}^{(k)}$  now grows linearly instead of exponentially, with the addition of only a small halfwidth vector at each  $h$  step to compensate for the local error. The matrix  $C^{(k)}$  changes at each  $h$  step by being premultiplied by a matrix that is very nearly

$$\begin{bmatrix} \cos h & \sin h \\ -\sin h & \cos h \end{bmatrix}$$

and the elements of the successive  $C^{(k)}$  matrices vary in the interval  $[-1, 1]$ . This revision of the error bounding system allows the second test problem to be solved to  $k$  decimal places surprisingly large distances from the initial  $x$  value  $a$ .

The system so far described works well with both test problems, but there is one final difficulty that is occasionally encountered. It is possible the differential equation system is such that the increment vector  $\mathbf{w}^+$  needed to adjust  $\mathbf{w}^{(k+1)}$ , although initially small, eventually becomes sizeable, even though the local error and the error due to the matrix  $B$  are small. An example where this happens is the initial value problem

$$\begin{bmatrix} u_1' \\ u_2' \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}; \quad \begin{bmatrix} u_1(0) \\ u_2(0) \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

which has the solution

$$\begin{bmatrix} u_1(x) \\ u_2(x) \end{bmatrix} = \begin{bmatrix} 1 + e^{-x} \\ 1 - e^{-x} \end{bmatrix}$$

What occurs here is that the matrices  $C^{(k)}$  approach singularity, so the elements of their inverses become ever larger, and eventually the correction vector  $\mathbf{w}^+$  contains huge components.

The two demonstration programs that solve differential equations, `difsys` and `difbnd`, monitor the magnitude of the elements of the  $C^{(k)}$  inverse matrix. When the largest component magnitude passes a certain bound,  $C^{(k)}$  is reset to  $I$ , and  $\mathbf{w}^{(k)}$  reset to  $|C^{(k)}|\mathbf{w}^{(k)}$ .

## 14.8 Solvable two-point boundary-value problems

To obtain a solution to the two-point boundary-value problem

$$\begin{aligned} \mathbf{u}' &= \mathbf{f}(x, \mathbf{u}) \\ \mathbf{g}(\mathbf{u}(a), \mathbf{u}(b)) &= \mathbf{0} \end{aligned} \tag{14.22}$$

we must find a vector  $\mathbf{v}$  such that the solution  $\mathbf{u}(x)$  to the initial value problem

$$\begin{aligned} \mathbf{u}' &= \mathbf{f}(x, \mathbf{u}) \\ \mathbf{u}(a) &= \mathbf{v} \end{aligned} \tag{14.23}$$

satisfies the boundary condition  $\mathbf{g}(\mathbf{u}(a), \mathbf{u}(b)) = \mathbf{0}$ . Generally a region  $R$  of  $m$ -space is prescribed in which to search for suitable vectors  $\mathbf{v}$ . If any of the

boundary conditions are of the form  $u_j(a) = c$ , where  $c$  is some constant, this simplifies the search, because the  $j$  component of  $\mathbf{v}$  is fixed at  $c$ , and the dimension of the search region is reduced by one. We use the notation  $\mathbf{u}(x; \mathbf{v})$  to denote the solution to the initial value problem corresponding to  $\mathbf{v}$ , so that the boundary condition can be written as  $\mathbf{g}(\mathbf{v}, \mathbf{u}(b; \mathbf{v})) = \mathbf{0}$ , or, more simply, as  $\mathbf{p}(\mathbf{v}) = \mathbf{0}$ , where  $\mathbf{p}(\mathbf{v})$  is  $\mathbf{g}(\mathbf{v}, \mathbf{u}(b; \mathbf{v}))$ . The two-point boundary-value problem is essentially a problem of finding zeros of the function  $\mathbf{p}$ . For each zero we find, the solution to the boundary problem is obtained by numerically solving the initial value problem (14.23) with  $\mathbf{v}$  set equal to the zero.

We will assume that the search region for  $\mathbf{v}$  is a box  $B$  in  $\mathbf{v}$ -space, possibly of dimension less than  $m$ , if some  $\mathbf{v}$  components are fixed by boundary conditions. We must be certain that for  $\mathbf{v}$  in  $B$ ,  $\mathbf{u}(x; \mathbf{v})$  is defined. This is a strong condition, requiring the solution of the initial value problem (14.23) to be defined and bounded throughout  $[a, b]$  for any  $\mathbf{v}$  in  $B$ .

We will phrase the solvable problem so that the goal is to approximate the  $\mathbf{p}$  zeros that lie in the search box  $B$ . If we obtain a zero accurately enough, we can obtain the corresponding solution of the boundary problem to the required number of decimal places. We model the problem after Solvable Problem 12.1 as follows:

**Solvable Problem 14.3** Given the two-point boundary-value problem  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$ ,  $\mathbf{g}(\mathbf{u}(a), \mathbf{u}(b)) = \mathbf{0}$ , where  $\mathbf{f}$  and  $\mathbf{g}$  are elementary, with the function  $\mathbf{p}(\mathbf{v}) = \mathbf{g}(\mathbf{v}, \mathbf{u}(b; \mathbf{v}))$  defined for  $\mathbf{v}$  in a box  $B$ , nonzero on the boundary of  $B$ , and having at most a finite number of zeros in  $B$ , given a containment box  $B_{[a,b],M}$  within which  $\mathbf{f}$  has partial derivatives with respect to all the  $\mathbf{u}$  variables, which is large enough to contain the solution to the initial value problem  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$ ,  $\mathbf{u}(a) = \mathbf{v}$ , for any  $\mathbf{v}$  in  $B$ , bound the zeros of  $\mathbf{p}(\mathbf{v})$  in  $B$  by

- (1) giving to  $k$  decimal places points identified as simple zeros, or
- (2) giving to  $k$  decimal places points identified as containing within their tilde-box a region where  $\max_i |p_i(\mathbf{v})| < 10^{-k}$ .

Solvable Problem 12.1 requires a search of the boundary of the box  $B$  to find points where the problem function is close to zero. And the crossing number helps to resolve whether a type (2) point actually is a zero. Although in principle these possibilities are open to us, the difficulty in obtaining a value for  $\mathbf{p}(\mathbf{v})$  precludes these luxuries in a practical program, so they have not been included in the statement of the preceding problem.

A type (1) “simple zero” point, is a  $\mathbf{v}$  value in  $B$  where the  $\mathbf{p}(\mathbf{v})$  Jacobian is nonzero, and definitely gives a solution to the boundary-value problem; whereas a type (2) point gives only a solution possibility. Such a point obtained for a certain  $k$  could be missing for a larger  $k$ .

The procedure of Chapter 12 for finding the zeros of a function  $\mathbf{f}$  can be adapted to Solvable Problem 14.3. This procedure, described in Section 12.3, has two parts. The search region  $B$  is repeatedly subdivided into smaller and smaller



subboxes, and intervals are found for the components of  $\mathbf{f}$  over these subboxes. A subbox is discarded whenever a component interval does not contain 0. Periodically, for the sets of subboxes remaining, container boxes are constructed, each bounding off a group of mutually adjacent subboxes, and these container boxes are the means of locating the simple zeros. We need to find a way to carry out the two parts of this procedure on the function  $\mathbf{p}$ .

Assume we can use the power series method of Sections 14.5 and 14.7 on the initial value problem (14.23). With this method, an interval for  $\mathbf{u}$  at an  $x$  point in  $[a, b]$  is converted to an interval for  $\mathbf{u}$  at another  $x$  point further along. So if we have intervals for the components of  $\mathbf{v}$ , we can obtain intervals for the components of  $\mathbf{u}(b; \mathbf{v})$ . These  $\mathbf{u}(b; \mathbf{v})$  intervals may be overly wide, but their widths decrease toward zero as the widths of the  $\mathbf{v}$  intervals decrease toward zero. And intervals for  $\mathbf{u}(b; \mathbf{v})$  lead to intervals via interval arithmetic for the components of  $\mathbf{p}(\mathbf{v})$ . Thus there is no fundamental difficulty carrying through the first part of the Chapter 12 procedure.

The Chapter 12 container box subroutine determines a simple zero point is present by first computing a Jacobian interval over the container box. So for our problem, determining that a container box contains a single simple zero of  $\mathbf{p}$  requires computing intervals for the partial derivatives of  $\mathbf{p}(\mathbf{v})$  with respect to the components of  $\mathbf{v}$ . This requires computing intervals for the partial derivatives of  $\mathbf{u}(b; \mathbf{v})$  with respect to the components of  $\mathbf{v}$ . When we differentiate the differential equations for  $\mathbf{u}(x; \mathbf{v})$  with respect to these components, we obtain the equations

$$\frac{\partial u'_i(x; \mathbf{v})}{\partial v_j} = \frac{\partial f_i(x, \mathbf{u})}{\partial v_j} \quad i, j = 1, \dots, m$$

which also may be written as

$$\left( \frac{\partial u_i(x; \mathbf{v})}{\partial v_j} \right)' = \sum_k \frac{\partial f_i(x, \mathbf{u})}{\partial u_k} \frac{\partial u_k(x; \mathbf{v})}{\partial v_j} \quad i, j = 1, \dots, m \quad (14.24)$$

We now have ordinary differential equations for the partial derivatives  $\partial u_i(x; \mathbf{v})/\partial v_j$ . If the function  $\mathbf{f}(x, \mathbf{u})$  has all the required partial derivatives, then, in principle at least, we can obtain intervals for  $\partial u_i(b; \mathbf{v})/\partial v_j$  over a  $\mathbf{v}$  box just as we can obtain intervals for  $u_i(b; \mathbf{v})$  over a  $\mathbf{v}$  box. We consider practical methods of getting such intervals next.

## 14.9 Solving the boundary-value problem by power series

Problem 14.3 needs to be revised so that power series methods are possible. The function  $\mathbf{f}$  is required to be infinitely differentiable with respect to all of its variables, and a containment box  $B_{[a,b],M}$  is not necessary if an appropriate escape is provided.

**Solvable Problem 14.4** Given the two-point boundary-value problem  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$ ,  $\mathbf{g}(\mathbf{u}(a), \mathbf{u}(b)) = \mathbf{0}$ , where  $\mathbf{f}$  and  $\mathbf{g}$  are elementary, where  $\mathbf{f}$  has all partial derivatives with respect to all its variables when  $x$  is in  $[a, b]$ , with the function  $\mathbf{p}(\mathbf{v}) = \mathbf{g}(\mathbf{v}, \mathbf{u}(b; \mathbf{v}))$  having at most a finite number of zeros for  $\mathbf{v}$  in a given box  $B$ , with  $\mathbf{p}(\mathbf{v})$  nonzero on the boundary of  $B$ , bound all the  $\mathbf{p}$  zeros in  $B$  by

- (1) giving to  $k$  decimal places points identified as simple zeros, or
- (2) giving to  $k$  decimal places points identified as containing within their tilde-box a region where  $\max_i |p_i(\mathbf{v})| < 10^{-k}$ ,  
or find to  $k$  decimal places a point  $\mathbf{v}_0$  in  $B$  where for some  $x$  in  $[a, b]$  a component of  $\mathbf{u}(x; \mathbf{v}_0)$  is larger in magnitude than a specified positive bound  $M$ .

In preceding sections of this chapter that were devoted to the initial value problem, we used a series evaluation method to obtain interval bounds for  $\mathbf{u}(x)$  for any  $x$  in  $[a, b]$ . With this method, when we have the representation  $\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}$  for  $\mathbf{u}$  at  $x_k$ , we then obtain a similar representation for  $\mathbf{u}$  at a point  $x_{k+1}$  to the right of  $x_k$ . Now, instead of starting at  $x = a$  with the  $\mathbf{u}$  representation  $\mathbf{v} \pm \mathbf{0}$ , as we did for the initial value problem, we start with the  $\mathbf{u}(a)$  representation  $\mathbf{v}^{(S)} \pm \mathbf{w}^{(S)}$  specified by the first  $\mathbf{v}$  subbox  $B^{(S)}$  on the queue. After the differential equation solution process reaches  $b$ , if  $x_{k'} = b$ , then the interval  $\mathbf{m}^{(k')} \pm \mathbf{w}^{(k')}$  is obtained for  $\mathbf{u}(b)$ , which becomes the interval representation of  $\mathbf{u}(b; \mathbf{v})$  over  $B^{(S)}$ . An interval arithmetic computation for  $\mathbf{g}(\mathbf{v}, \mathbf{u}(b; \mathbf{v}))$  gives us an interval for  $\mathbf{p}(\mathbf{v})$  over  $B^{(S)}$ . We have now determined how to dot the first part of the Chapter 12 procedure for locating zeros.

To do the second part of the procedure, interval bounds must be obtained for the elements of the  $\mathbf{p}(\mathbf{v})$  Jacobian matrix over a  $\mathbf{v}$  container box  $B^{(C)}$ . For this part we need to do an interval degree 1 series computation for  $\mathbf{g}(\mathbf{v}, \mathbf{u}(b; \mathbf{v}))$ . If the midpoint-halfwidth representation for the box  $B^{(C)}$  is  $\mathbf{v}^{(C)} \pm \mathbf{w}^{(C)}$ , we set the degree 1 series representation for  $\mathbf{u}$  at  $x = a$  over this box to

$$\mathbf{u} = (\mathbf{v}^{(C)} \pm \mathbf{w}^{(C)}) + (I \pm O)(\mathbf{v} - \mathbf{v}^{(C)})$$

We need a representation for  $\mathbf{u}(b; \mathbf{v})$  of the form

$$\mathbf{u}(b; \mathbf{v}) = (\mathbf{m}^{(k')} \pm \mathbf{w}^{(k')}) + (A^{(k')} \pm B^{(k')})(\mathbf{v} - \mathbf{v}^{(C)})$$

Here we are supposing that somehow the  $\mathbf{v}$  partial derivatives of  $\mathbf{u}(b; \mathbf{v})$  have been bounded in the intervals defined by matrices  $A^{(k')}$  and  $B^{(k')}$ . With its variables  $\mathbf{v}$  and  $\mathbf{u}(b; \mathbf{v})$  set this way, a degree 1 series expansion of the elementary function  $\mathbf{g}$  will give us the  $\mathbf{p}$  partial derivative intervals we need. In the expression just given for  $\mathbf{u}$  at  $a$ , we presumed that none of the  $\mathbf{v}$  components had been fixed as constants by the boundary conditions. If component  $j$  of  $\mathbf{v}$  is fixed at the value  $c$ , then its degree 0 term becomes  $c \pm 0$ , and its degree 1 terms all have coefficients  $0 \pm 0$ .

We can obtain the required partial derivative intervals for  $\mathbf{u}(b; \mathbf{v})$  over a box  $B^{(C)}$  by extending the procedure for obtaining intervals for  $\mathbf{u}(b; \mathbf{v})$  over  $B^{(S)}$ . Here we not only bound in intervals the solution to the differential equation  $\mathbf{u}' = \mathbf{f}(x, \mathbf{u})$  at selected points  $x_k$  in  $[a, b]$ , but we must also bound in intervals the solution to the differential equations (14.24). Suppose then that at a point  $x_k$  in  $[a, b]$  we have obtained the interval vector  $\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}$  for  $\mathbf{u}(x_k; \mathbf{v})$  over the box  $B^{(C)}$ , and also have obtained the interval array  $A^{(k)} \pm B^{(k)}$  for the Jacobian elements of  $\mathbf{u}(x_k; \mathbf{v})$  over the same box. The representation for  $\mathbf{u}$  as a series in  $\mathbf{v}$  is

$$\mathbf{u} = (\mathbf{m}^{(k)} \pm \mathbf{w}^{(k)}) + (A^{(k)} \pm B^{(k)})(\mathbf{v} - \mathbf{v}^{(C)}) \quad (14.25)$$

In Section 14.5 there were two parts to the procedure for obtaining a series for  $\mathbf{u}$  in  $[x_k, x_k + h]$ . In the first part, a containment box for  $\mathbf{u}$  is obtained, with  $h$  determined in the process and the local error found. In the second part, the global error for  $\mathbf{u}$  at  $x_{k+1}$  is found. Since we are solving differential equations just as before, our procedure has the same two parts. At the point  $x_k$ , we have  $\mathbf{u}$  given by (14.25). (Initially, when  $k = 1$  and  $x_1 = a$ , then  $A^{(1)} = I$  and  $B^{(1)} = O$ .) In the first part of the procedure, variables are set to

$$\begin{aligned} x &= (x_k \pm h) + (1 \pm 0)(x - x_k) \\ \mathbf{u} &= (\mathbf{m}^{(k)} \pm \widehat{\mathbf{w}}) + (A^{(k)} \pm \widehat{B})(\mathbf{v} - \mathbf{v}^{(C)}) \end{aligned}$$

The halfwidth vector  $\widehat{\mathbf{w}}$  and halfwidth matrix  $\widehat{B}$  have components that are larger than or identical to corresponding components of  $\mathbf{w}^{(k)}$  and  $B^{(k)}$ . For the chosen step width  $h$ , we determine whether the  $\widehat{\mathbf{w}}$  and  $\widehat{B}$  estimates are acceptable by the outcome of a series computation for  $\mathbf{f}$  that is degree 0 in  $x$ :

$$\mathbf{f} = (\mathbf{f}_0 \pm \widehat{\mathbf{w}}_0) + (A_0 \pm \widehat{B}_0)(\mathbf{v} - \mathbf{v}^{(C)})$$

(In this series computation,  $\mathbf{u}$  is viewed not only as a function of  $x$ , but also as a function of the  $\mathbf{v}$  components, which are variables independent of  $x$ . If an  $x$  degree  $k$  computation of  $\mathbf{f}$  is made, the expression is always obtained to  $\mathbf{v}$  degree 1. This gives us the needed  $x$  degree  $k$  values for  $(\partial u_i / \partial v_j)'$ .) Now we make the containment test

$$\mathbf{w}^{(k)} + h(|\mathbf{f}_0| + \widehat{\mathbf{w}}_0) < \widehat{\mathbf{w}} \quad B + h(|A_0| + \widehat{B}_0) < \widehat{B}$$

If any component of  $\widehat{\mathbf{w}}$  or  $\widehat{B}$  fails this test, it is replaced by the corresponding component on the other side of the inequality, times a factor slightly above 1, such as 1.1, and the test is repeated after  $\mathbf{f}$  is recalculated. After a fixed number of these tests fail,  $h$  is halved, and the whole process is restarted.

After  $\widehat{\mathbf{w}}$  and  $\widehat{B}$  are found, the next step is to obtain all the higher degree  $x$  terms of the  $\mathbf{u}$  series. Note that if the  $x$  degree  $k$  term of  $\mathbf{f}$  is determined to be

$$[(\mathbf{f}_k \pm \mathbf{w}_k) + (A_k \pm B_k)(\mathbf{v} - \mathbf{v}^{(C)})](x - x_k)^k$$

then the  $x$  degree  $k + 1$  term of  $\mathbf{u}$  is

$$\left[ \left( \frac{1}{k+1} \mathbf{f}_k \pm \frac{1}{k+1} \mathbf{w}_k \right) + \left( \frac{1}{k+1} A_k \pm \frac{1}{k+1} B_k \right) (\mathbf{v} - \mathbf{v}^{(C)}) \right] (x - x_k)^{k+1}$$

All the other parts of the power series computation described in Section 14.5 have their counterparts in this computation. The difference is that now we compute additional series terms in  $\mathbf{v}$  when we make our  $\mathbf{u}$  power series computations.

The method for improving halfwidth computations described in Section 14.6 needs to be used here also to keep halfwidth values from becoming unnecessarily large. There are more intervals now to be viewed as variables than previously, these being all the intervals in (14.25).

### 14.10 The linear boundary-value problem

The boundary-value problem becomes easier to treat if the differential equations are linear and the boundary conditions are linear. In this case the two-point boundary-value problem itself is called *linear*. The function  $\mathbf{f}(x, \mathbf{u})$  has the vector-matrix representation

$$f(x, \mathbf{u}) = G(x)\mathbf{u} + \mathbf{h}(x)$$

where  $G(x)$  is an  $m$ -square matrix whose elements are functions of  $x$ , and  $\mathbf{h}(x)$  is an  $m$ -vector whose components also are functions of  $x$ . If all these functions of  $x$  are elementary and defined in  $[a, b]$ , then according to Solvable Problem 14.4, the components of  $\mathbf{u}(b; \mathbf{v})$  can be determined for any  $\mathbf{v}$ .

Because the differential equations are linear, it is possible to obtain the following representation for  $\mathbf{u}(b; \mathbf{v})$ :

$$\mathbf{u}(b; \mathbf{v}) = Q\mathbf{v} + \mathbf{r} \tag{14.26}$$

Here  $Q$  is a real  $m$ -square matrix and  $\mathbf{r}$  is a real  $m$ -vector. Column  $j$  of  $Q$  equals the solution at  $x = b$  of the initial value problem

$$\mathbf{u}' = G(x)\mathbf{u}, \quad \mathbf{u}(a) = \mathbf{e}_j \tag{14.27}$$

where  $\mathbf{e}_j$  equals column  $j$  of the  $m$ -square identity matrix  $I$ . The vector  $\mathbf{r}$  equals the solution at  $x = b$  of the initial value problem

$$\mathbf{u}' = \mathbf{h}(x), \quad \mathbf{u}(a) = \mathbf{0} \tag{14.28}$$

If for each  $j$  we multiply by  $v_j$  the solution to (14.27) and sum, and then add the solution to (14.28), we obtain the solution to the initial value problem

$$\mathbf{u}' = G(x)\mathbf{u} + \mathbf{h}(x), \quad \mathbf{u}(a) = \mathbf{v}$$

If we use equation (14.26) in the linear boundary condition equation

$$A \mathbf{u}(a) + B \mathbf{u}(b) = \mathbf{c}$$

we obtain the matrix equation

$$A\mathbf{v} + BQ\mathbf{v} = \mathbf{c} - B\mathbf{r} \quad (14.29)$$

Thus if the matrix  $A + BQ$  is nonsingular, there is only one vector  $\mathbf{v}$  which gives a solution to the boundary-value problem, the vector  $(A + BQ)^{-1}(\mathbf{c} - B\mathbf{r})$ . Note that the matrix  $A + BQ$  equals the Jacobian matrix of  $\mathbf{p}(\mathbf{v}) = A\mathbf{v} + B\mathbf{u}(b, \mathbf{v}) - \mathbf{c}$ . For the linear boundary-value problem, a possible solvable problem is

**Solvable Problem 14.5** Given the linear two-point boundary-value problem

$$\mathbf{u}' = G(x)\mathbf{u} + \mathbf{h}(x), \quad A \mathbf{u}(a) + B \mathbf{u}(b) = \mathbf{c}$$

where the elements of the matrix  $G(x)$  and the components of the vector  $\mathbf{h}(x)$  are elementary functions of  $x$  defined in  $[a, b]$ , find to  $k$  decimal places the lone simple zero of  $\mathbf{p}(\mathbf{v}) = A\mathbf{v} + B\mathbf{u}(b, \mathbf{v}) - \mathbf{c}$ , or indicate that the magnitude of the  $\mathbf{p}$  Jacobian is less than  $10^{-k}$ .

This problem can be solved numerically by finding  $m + 1$  separate solutions to various initial value problems, namely the solutions whose values at  $x = b$  are needed to form  $Q$  and  $\mathbf{r}$ . But generally it is more efficient to get these values by using the procedure described in the preceding section for obtaining the  $\mathbf{G}(\mathbf{v})$  partial derivatives, because then we need only one sweep through  $[a, b]$  with the differential equation solution procedure.

## Software Exercises L

These exercises are with the demo programs `difsys` and `difbnd`.

1. Call up `difsys` and solve the simple initial value problem  $u' = u$ ,  $u(0) = 1$  to 10 fixed-point decimal places for  $x$  in the interval  $[0, 10]$ . Set the  $x$  increment between printouts to 0.1. The problem's solution is the exponential function  $u = e^x$ . Check the print file to see that 10 decimal place accuracy is not obtained toward the end of the  $[0, 10]$  interval. Edit the log file to change the 10 fixed-point decimal places to 10 floating-point decimal places, and then call up `difsys` `difsys`. This time 10 decimal place accuracy is obtained throughout  $[0, 10]$ .

2. In this exercise and the next, we see the response of `difsys` to the two difficult example problems of Section 14.3. Call up `difsys` and attempt to solve to 10 fixed-point decimal places the problem  $u' = u^2$ ,  $u(0) = 1$  in the  $x$  interval  $[0, 2]$ . Set the  $x$  increment between printouts to 0.01. `difsys` gives values until  $x = 0.99$ , then indicates that the integration step width has become too small, and terminates. The program `difsys` does not require a user to specify a containment box, as is required by Problem 14.1, so the method of indicating trouble differs from that in Problem 14.1.

3. Call up `difsys` and solve to 10 fixed-point decimal places the problem  $u' = u^{1/3}$ ,  $u(0) = 0$  in the  $x$  interval  $[0, 1]$ . Set the  $x$  increment between printouts to 0.1. The problem is rejected by `difsys` because  $u$  does not satisfy differentiability conditions at  $x = 0$ . Edit the log file to change  $u(0)$  from 0 to 0.01, and then call up `difsys difsys`. This time a solution is obtained.

4. Call up `difsys` and solve an initial value problem for  $u'' = -u$  in the interval  $[0, 20\pi]$  to 10 fixed-point decimal places. Take the initial conditions to be  $u(0) = 0$  and  $u'(0) = 1$ , and set the  $x$  increment between printouts to  $\pi/10$ . The problem's solution is  $u = \sin x$ . Afterwards, edit the log file to change from 10 fixed-point decimal places to 10 floating-point decimal places and then call up `difsys difsys` to see the changed output.

5. Call up `difbnd` and solve the string eigenvalue problem given at the end of Section 14.2. The differential equations and boundary conditions are

$$\begin{aligned} u_1' &= u_2 & u_1(0) &= 0 \\ u_2' &= -u_3 u_1 & u_2(0) &= 1 \\ u_3' &= 0 & u_1(\pi) &= 0 \end{aligned}$$

Take the  $[a, b]$  interval to be  $[0, \pi]$ . After you specify the initial conditions, `difbnd` requests an interval for  $u_3$ , so specify the interval  $[0, 10]$ . Set the display interval to  $[0, \pi]$ , the  $x$  increment between printouts to  $\pi/10$ , and choose 10 decimal-place accuracy. The program finds three distinct solutions.

6. Call up `difbnd` and attempt to solve to 5 decimal places in the interval  $[0, \pi]$  the linear two-point boundary-value problem  $u'' + u = 0$ ,  $u(0) = u(\pi) = 0$ . Take  $\pi/10$  as the printout increment. The function  $u(x) = c \sin x$  with  $c$  arbitrary satisfies the problem, so there are an infinite number of solutions. With this problem, the program `difbnd` takes the escape of Solvable Problem 14.5, and indicates that the Jacobian of the boundary condition matrix is too small. Edit the log file to change  $b$  to  $\pi/2$ , which changes the boundary conditions to  $u(0) = u(\pi/2) = 0$ , and call up `difbnd difbnd`. This time `difbnd` finds the unique solution.

7. Call up `difbnd` and solve to 5 decimal places in the interval  $[0, 1]$  the two-point nonlinear boundary-value problem  $u'' = -e^u$ ,  $u(0) = u(1) = 0$ . Use a search interval for  $u'$  of  $[0, 12]$  and set the printout increment to 0.1. The

program `difbnd` locates 2 separate solutions, one with  $u'(0)$  near 0.5 and one with  $u'(0)$  near 11.

## Notes and References

- A. Degree 1 interval arithmetic was first proposed by Eldon Hansen [6] [7] [8].
  - B. The global error improvement method of Section 14.7, maintaining the vector  $\mathbf{w}^{(k)}$  in the form  $C^{(k)}\mathbf{w}^{(k)}$ , was first proposed by Rudolf Lohner in his Ph.D. dissertation [9].
  - C. Three general references for the problems of this chapter are the book by Ascher, Mattheij, and Russell [1], the book by Coddington and Levinson [3], and the two volumes by Hairer et al. [4] [5].
- 
- [1] Ascher, U. M., Mattheij, R. M., and Russell, R. D., *Numerical Solution of Boundary Value Problems of Ordinary Differential Equations*, Prentice Hall, Englewood Cliffs, NJ, 1988.
  - [2] Brenan, K. E., Campbell, S. L., and Petzold, L. R., *Numerical Solution of Initial-Value Problems of Differential-Algebraic Equations*, Elsevier, New York, 1989.
  - [3] Coddington, E. A. and Levinson, N., *Theory of Ordinary Differential Equations*, Reprint Edn published by Robert E. Krieger Publ. Co., Malabar, FL, 1987.
  - [4] Hairer, E., Nørsett, S. P., and Wanner, G., *Solving Ordinary Differential Equations I, Nonstiff Problems*, Springer series in computational mathematics 8, Springer-Verlag, Berlin, 1987.
  - [5] Hairer, E. and Wanner, G., *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, Springer series in computational mathematics 14, Springer-Verlag, Berlin, 1991.
  - [6] Hansen, E., On the solution of linear algebraic equations with interval coefficients *Linear Algebra Appl.* **2** (1969), 153–165.
  - [7] Hansen, E., *A generalized interval arithmetic*, in *Interval Mathematics*, K. Nickel editor, Springer-Verlag, New York, 1975, pp. 7–18.
  - [8] Hansen, E., Computing zeros of functions using generalized interval arithmetic, *Interval Computations* **3** (1993), 3–28.
  - [9] Lohner, R., *Einschliesung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*, Doctorate of Science dissertation, Faculty of Mathematics of Karlsruhe University, 1988.

# Partial Differential Equations



The demo program `pde` solves what is known as the initial value problem for a system of partial differential equations. In this chapter the term “function analytic at an argument point  $P$ ” means the function at  $P$  has a derivative of any specified order with respect to its variables.

## 15.1 Partial differential equation terminology

With ordinary differential equations (ODEs), it is customary to use  $x$  as the lone independent variable. With partial differential equations (PDEs), where there are two or more independent variables, it has become customary to use  $t$  as the last independent variable. In many PDE problems,  $t$  designates time, but in general, it is best to think of  $t$  as simply the final independent variable.

To prepare for PDE problems, we reconsider the initial value ODE problem, using  $t$  in place of  $x$ . We suppose that in some  $t$  interval  $[t_0, t_1]$  a system of differential equations is to be solved. A typical equation for one of the functions  $u(t)$  has the form

$$\frac{d^r u}{dt^r} = F\left(t, u, \frac{du}{dt}, \dots, \frac{d^{r-1}u}{dt^{r-1}}, \dots\right)$$

and the corresponding initial conditions are

$$\frac{d^j u(t_0)}{dt^j} = a_j, \quad j = 0, 1, \dots, r-1$$

Here  $u$  and its derivatives are the “central variables” of the function  $F$ , which is presumed to be elementary and analytic in its domain. The  $F$  starred dots ( $\dots^*$ )



are there to allow the appearance of central variables from the other ODEs of the system. When we convert the list of equations in the usual way to a collection of  $m$  first order differential equations in the new variables  $v_1, v_2, \dots, v_m$ , the equations and initial conditions take the form

$$\begin{aligned} \frac{dv_i}{dt} &= f_i(t, v_1, v_2, \dots, v_m) \\ v_i(t_0) &= \alpha_i \end{aligned} \quad (15.1)$$

$$i = 1, 2, \dots, m$$

(If  $r_i$  is the order of the  $i$ th original equation, then  $m = \sum_i r_i$ .)

The initial value problem for a PDE system defined over a rectangle in two dimensional  $(x, t)$  space is similar. The rectangle  $R$  is the product of the  $x$  interval  $[x_0, x_1]$  and the  $t$  interval  $[t_0, t_1]$ , and all initial values are given either along the  $t = t_0$  boundary line of  $R$  or along the  $t = t_1$  boundary line. For definiteness, we presume the boundary line  $t = t_0$  is specified. A typical equation for one of the functions  $u(x, t)$  has the following form:

$$\frac{\partial^r u}{\partial t^r} = F \left( x, t, u, \frac{\partial u}{\partial x}, \dots, \frac{\partial^{q+r-1} u}{\partial x^q \partial t^{r-1}}, * \dots \right)$$

Here the  $r$ th order  $t$  derivative of  $u$  equals a function  $F$  that is allowed to depend on  $u$  and the mixed derivatives of  $u$ , of order at most  $q$  in  $x$  and of order at most  $r-1$  in  $t$ , these being the ‘‘central variables’’ of  $F$ . Again the  $F$  starred dots ( $*$ ) are there to allow the appearance of central variables from the other PDEs of the system. The initial conditions along  $R$ 's  $t = t_0$  boundary line are

$$\frac{\partial^j u}{\partial t^j}(x, t_0) = G_j(x) \quad j = 0, 1, \dots, r-1$$

Assume that the functions  $F$  and  $G_j$  are elementary and analytic in their domains, and that each solution function  $u$  is uniquely defined and bounded in  $R$ . In analogous fashion as with the ODE system, one can convert the PDE system into  $m$  PDEs that are first order in  $t$ , defining  $m$  functions  $v_1, v_2, \dots, v_m$ . These PDEs with their initial conditions have the form:

$$\begin{aligned} \frac{\partial v_i}{\partial t} &= f_i \left( x, t, v_1, \dots, \frac{\partial^{p_1} v_1}{\partial x^{p_1}}, v_2, \dots, \frac{\partial^{p_2} v_2}{\partial x^{p_2}}, \dots, v_m, \dots, \frac{\partial^{p_m} v_m}{\partial x^{p_m}} \right) \\ v_i(x, t_0) &= g_i(x) \end{aligned} \quad (15.2)$$

$$i = 1, 2, \dots, m$$

Each integer  $p_j$  defines the highest  $x$  derivative needed for the function  $v_j$ .

Sometimes one or more of the functions  $v_i$  has a known value along the  $R$  boundary line  $x = x_0$ , the  $R$  boundary line  $x = x_1$ , or along both lines. It has become customary to make use of this information in the numerical treatment of PDE initial value problems, and the problem then is called “the initial value–boundary value PDE problem”.

## 15.2 ODE and PDE initial value problems

For the ODE problem (15.1), we assume the functions  $f_i$  are such that unique solutions  $v_i$  are defined in  $[t_0, t_1]$ . Similarly, for the PDE problem (15.2), we assume the functions  $f_i$  and  $g_i$  are such that unique solutions  $v_i$  are defined in  $R$ . For either problem, suppose our numerical task is to compute to  $k$  decimal places at any designated point in the  $t$  or  $(x, t)$  domains the functions  $v_i$  and all  $v_i$  derivatives that occur in the differential equations governing the problem. The solvable problems of Chapter 14 indicate that the ODE problem is solvable. For the PDE problem, we have

**Nonsolvable Problem 15.1** Given the initial value problem (15.2), with the elementary functions  $f_i$  analytic in  $R$ , the elementary functions  $g_i$  analytic in  $[t_0, t_1]$ , and with the solution functions  $v_i$  defined in  $R$ , at any designated point  $(x, t)$  in  $R$  compute to  $k$  decimal places the functions  $v_i$  and all partial derivatives occurring in the PDEs.

Although the PDE problem in general is nonsolvable, there are some cases where the problem becomes solvable. For instance, suppose the equations (15.2) have the form

$$\begin{aligned}\frac{\partial v_i}{\partial t} &= f_i(x, t, v_1, v_2, \dots, v_m) \\ v_i(x, t_0) &= g_i(x) \\ i &= 1, 2, \dots, m\end{aligned}\tag{15.3}$$

Since the PDEs have no  $x$  partial derivatives, they can be treated as ODEs, and in that case we have seen that the problem is solvable.

To show that the objective of Nonsolvable Problem 15.1 is in general not possible, we need to consider any other PDE system not of the type shown in (15.3). We will use the following simple example problem:

$$\frac{\partial v}{\partial t} = \frac{\partial v}{\partial x}$$

Here  $[x_0, x_1] \times [t_0, t_1]$  is taken as  $[-\pi, \pi] \times [0, 10]$ . The initial condition along the line  $t = 0$  is

$$v(x, 0) = g(x)$$

The solution is  $v(x, t) = g(x + t)$ . Suppose now that  $g(x)$  has the following form with the parameter  $a$ :

$$g_a(x) = \begin{cases} a \sin\left(\frac{x}{a}\right) & \text{if } a \neq 0 \\ 0 & \text{if } a = 0 \end{cases} \quad (15.4)$$

If  $a \neq 0$ , we have  $v(x, t) = a \sin((x + t)/a)$  and  $\partial v/\partial x = \cos((x + t)/a)$ ; but if  $a = 0$ , we have  $v(x, t) = 0$ . The difficulty here is that if the parameter  $a$  makes an arbitrarily small change from zero to a nonzero value, it leads to large changes in  $\partial v/\partial x$  throughout the rectangle  $R$ . For instance, if  $a \neq 0$ ,  $\partial v(x, t)/\partial x = 1$  at all points in  $R$  where  $x + t = 0$ ; but if  $a = 0$ , then  $\partial v(x, t)/\partial x = 0$  throughout  $R$ . Thus an accurate value for  $\partial v(x, t)/\partial x$  is not possible in general. If there were some method to compute  $\partial v(x, t)/\partial x$  accurately for  $g_a$ , this would contradict Nonsolvable Problem 3.1.

The PDE problem becomes solvable if we give up trying to compute the true solution and its partial derivatives accurately, and try only to find functions  $\widehat{v}_i$  that satisfy the PDEs and initial conditions to a prescribed error bound. With our example problem, we obtain a value for  $\partial \widehat{v}(x, t)/\partial x$ , but because no claim is made that  $\widehat{v}$  is close to the true solution, the troublesome initial value function  $g_a$  poses no difficulty.

We make this change explicit with

**Solvable Problem 15.2** Given the initial value problem (15.2), with the elementary functions  $f_i$  analytic in  $R$ , the elementary functions  $g_i$  analytic in  $[t_0, t_1]$ , and with the solution functions  $v_i$  defined in  $R$ , generate functions  $\widehat{v}_i$  defined in  $R$  such that  $\widehat{v}_i$  satisfies all PDE equations and initial conditions to  $k$  decimal places.

The remainder of this chapter is about methods of generating  $\widehat{v}_i$ . The preceding chapter described power series methods for ordinary differential equations, and we extend those methods to apply to our problem.

### 15.3 A power series method for the ODE problem

The PDE method is easier to describe if first we explain how the method is used with the ODE initial value problem (15.1) to bound the error of the ODEs. Suppose an error bound  $\epsilon$  is prescribed for the ODEs, and assume an initial  $t$  interval  $[t_0, t_0 + h]$  is chosen. The  $t$  series, valid in the interval  $t_0 \pm h$ , now is

$$t = (t_0 \pm h) + (1 \pm 0)(t - t_0)$$

A preliminary goal is to find Taylor series expressions for all functions  $v_i$ . With these series available in interval arithmetic form, we can calculate  $f_i$  series, and

this allows us to bound the errors of the ODEs. The series eventually obtained for the functions  $v_i$  are the following:

$$v_i(t) = \sum_{k=0}^K (v_{i,k} \pm w_{i,k})(t - t_0)^k \quad i = 1, 2, \dots, m \quad (15.5)$$

With these series and the  $t$  series one can compute the  $f_i$  series

$$f_i(t) = \sum_{k=0}^K (f_{i,k} \pm w'_{i,k})(t - t_0)^k \quad i = 1, 2, \dots, m \quad (15.6)$$

All the series of lines (15.5) and (15.6) are constructed together, one term at a time. Imagine now that the highest degree is  $k$ , a variable that starts at 0 and rises up to  $K$ . For each  $k$  setting, terms of degree  $k$  are computed for all series. Note that each function  $f_i$ , besides possibly being a function of  $t$ , with a known interval halfwidth, may also be a function of some of the variables  $v_1, \dots, v_m$ , with unknown interval halfwidths. We describe next the procedure for assigning halfwidths to the  $v_i$  variables, to obtain valid interval series terms for all functions  $f_i$ .

When  $k = 0$ , the series of line (15.5) have the form

$$v_i(t) = (v_{i,0} \pm w_{i,0}) \quad i = 1, \dots, m \quad (15.7)$$

Here  $v_{i,0}$  can be set equal to  $\alpha_i$ , using the initial condition part of line (15.1), and only the interval halfwidths  $w_{i,0}$  need to be determined. The computation for these halfwidths when  $k = 0$  is the most time-consuming part of the series construction process. First suppose these halfwidths are set to arbitrary nonnegative values, and afterwards the functions  $f_i$ , when computed, yield the intervals

$$f_{i,0} \pm w'_{i,0} \quad i = 1, 2, \dots, m \quad (15.8)$$

For any function  $y(t)$  and nonnegative number  $M$ , if we find  $|\frac{dy}{dt}| \leq M$  for  $t$  in some interval  $[a, b]$ , this implies  $|y(t) - y(a)| \leq M(b - a)$ . So if we find that

$$(|f_{i,0}| + w'_{i,0}) \cdot h < w_{i,0} \quad i = 1, 2, \dots, m \quad (15.9)$$

it is certain that the halfwidths  $w_{i,0}$  have not been set too small. A cyclic process is used to obtain sufficiently wide halfwidths that are not overly large. Initially all halfwidths  $w_{i,0}$  are set to zero. Then the intervals of line (15.8) are computed and the tests of line (15.9) are made. If for any  $i$  we find  $M = (|f_{i,0}| + w'_{i,0})$  is so large that we do not have  $Mh < w_{i,0}$ , then  $w_{i,0}$  is reset to  $Mh$  times a factor slightly larger than 1, such as 1.1. Whenever any  $w_{i,0}$  is increased, the entire computation is repeated using the larger halfwidths. Usually the test of

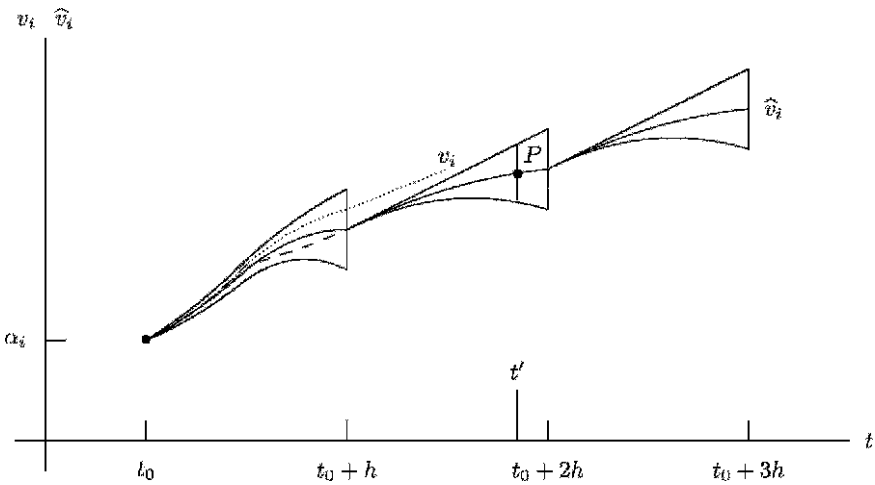
line (15.9) succeeds after a few cycles, but in cases where this does not occur, the  $t$  interval width  $h$  is halved, and the whole process begins anew.

When all interval widths  $w_{i,0}$  have been chosen successfully, with the degree zero terms of the  $f_i$  expansions (15.6) now also known in interval form, all higher degree series terms are obtained relatively easily. For instance, suppose all degree  $k$  terms are known. Then the degree  $k + 1$  terms of the  $v_i$  series can be found by setting  $(v_{i,k+1} \pm w_{i,k+1})$  equal to  $\frac{1}{k+1}(f_{i,k} \pm w'_{i,k})$  (because the derivative of  $v_i$  is  $f_i$ ). After these terms are obtained, the degree  $k + 1$  terms of the functions  $f_i$  also can be calculated. If there should be a problem calculating an  $f_i$  term, perhaps caused by using a divisor interval containing the zero point, the interval width  $h$  is halved, and the entire process begins anew.

When the degree parameter  $k$  reaches its terminal value  $K$ , a test for ODE accuracy is made. For any  $t$  in  $[t_0, t_0 + h]$ , the error of the  $f_i$  series  $\sum_{k=0}^K f_{i,k}(t - t_0)^k$  is bounded by  $w'_{i,K} |t - t_0|^K$ . The ODE error bound  $\epsilon$  is achieved if for all  $i$  we find  $w'_{i,K} h^K < \epsilon$ . If all inequalities hold, and similar inequalities hold for the  $v_i$  series, the collection of series is saved for later evaluation, and the whole process starts again in a new  $t$  subinterval  $[t^*, t^* + h]$ , with  $t^* = t_0 + h$ , using the new initial values  $\alpha_i^* = \sum_{k=0}^K v_{i,k} h^k$ ,  $i = 1, \dots, m$ . If any of these inequalities fail,  $h$  is halved and the entire process is repeated. The procedure continues until the  $t$  subintervals cover  $[t_0, t_1]$ .

The described ODE method produces in each  $t$  subinterval polynomial approximations to  $v_i$  and  $dv_i/dt$ ,  $i = 1, \dots, m$ , which we now designate as  $\widehat{v}_i$  and  $d\widehat{v}_i/dt$ . A pair of  $\widehat{v}_i$  polynomials for two adjoining subintervals have the same values at the point separating the subintervals, but this is not generally true for the pair of  $d\widehat{v}_i/dt$  polynomials.

Fig. 15.1 shows a typical function  $\widehat{v}_i(t)$  over a  $t$  interval containing the first three adjoining subintervals in which approximations are obtained. The dotted line represents the solution  $v_i$  to the ODE initial value problem, and the central solid line represents the successive polynomial approximations  $\sum_{k=0}^K v_{i,k}(t - t_0)^k$  used by  $\widehat{v}_i$ . Surrounding the central line are the two boundary lines provided by the interval arithmetic computation. The upper line is defined by  $\sum_{k=0}^{K-1} v_{i,k}(t - t_0)^k + (v_{i,K} + w_{i,K})(t - t_0)^K$ , and the lower line is defined by  $\sum_{k=0}^{K-1} v_{i,k}(t - t_0)^k + (v_{i,K} - w_{i,K})(t - t_0)^K$ . (The separation between the two boundary lines is exaggerated to make the diagram clearer.) At any  $t$  point, such as  $t'$ , the  $\widehat{v}_i$  interval arithmetic approximation has a midpoint  $P$  on the  $\widehat{v}_i$  midline graph corresponding to  $t'$ , with a halfwidth given by the vertical distance from  $P$  to either of the  $\widehat{v}_i$  boundary lines. The  $\widehat{v}_i$  approximation thus is always within the two boundary lines, and we can consider the graph of  $\widehat{v}_i$  to be the graph obtained after all the space between the two boundary lines is shaded in. The discontinuity of the derivative of this graph's center line does not imply that the function  $\widehat{v}_i$  has a discontinuous derivative at  $t_0 + h$ , because  $\widehat{v}_i$ 's value is known only in interval form at  $t_0 + h$ . Within the bounds of  $\widehat{v}_i$ 's graph, the graph of a function having a continuous derivative could be drawn. In Fig. 15.1, a graph of this hypothetical function is shown in the first  $t$  subinterval as the dashed line.



**Fig. 15.1** A schematic graph of  $v_i$  and  $\widehat{v}_i$

The function  $\widehat{v}_i$  satisfies its ODE to within the  $\epsilon$  bound. Note, however, that the true solution  $v_i$  is certain to lie within the  $\widehat{v}_i$  boundary lines only in the first  $t$  subinterval. In Fig. 15.1, the  $v_i$  graph is shown outside these lines in the second  $t$  subinterval.

### 15.4 The first PDE solution method

Suppose in the rectangle  $R = [x_0, x_1] \times [t_0, t_1]$  an error bound  $\epsilon$  is prescribed for the PDEs, and the same error bound is prescribed for the difference between the computed PDE solutions and the specified initial values on  $R$ 's boundary line at  $t = t_0$ .

With the ODE problem, initial values are real numbers, but now initial values are functions  $g_i$  that are approximated by polynomials. An approximation polynomial is obtained by Taylor series expansion, with again all terms computed in interval arithmetic. If  $x_m$  is the  $x$  interval midpoint  $(x_0 + x_1)/2$ , and  $w_m$  is the  $x$  interval halfwidth  $(x_1 - x_0)/2$ , then the variable  $x$  has the two-term series

$$x = (x_m \pm w_m) + (1 \pm 0)(x - x_m) \tag{15.10}$$

Using the  $x$  series, each function  $g_i$  is expanded into a Taylor series of degree  $J$ :

$$g_i(t) = (g_{i,0} \pm w''_{i,0}) + (g_{i,1} \pm w''_{i,1})(x - x_m) + \dots + (g_{i,J} \pm w''_{i,J})(x - x_m)^J, \quad i = 1, \dots, m$$

Here  $J$  is the initial choice for the expansion degree, which should be a reasonably large value, such as 12. The accuracy test that must be passed is

$$w''_{i,J}(w_m)^J < \epsilon \quad i = 1, 2, \dots, m \quad (15.11)$$

Whenever the test fails for some function  $g_i$ , then  $J$  is increased by an amount  $\Delta J$ , such as 6, and the Taylor series is computed to the higher degree, the process continuing until either the accuracy test holds or the  $J$  upper bound of the implementation is reached.

It is possible that a  $g_i$  approximating polynomial is not obtained for either of two reasons. It could be that the function  $g_i(t)$ , although analytic in  $[x_0, x_1]$ , cannot be developed into a Taylor series using the  $x$  series of line (15.10). This occurs, for example, if  $[x_0, x_1] = [-2, 2]$ , and an initial value function  $g_i$  is  $1/(1+x^2)$ . This function is undefined in the complex plane at the points  $i$  and  $-i$ , and in elementary complex function theory, it is shown that such a function cannot have a Taylor expansion at the series expansion point 0, the midpoint of  $[-2, 2]$ , that is valid at any point in the complex plane further from 0 than  $i$  or  $-i$ . So finding a Taylor series that is valid in the wide interval  $[-2, 2]$  is impossible. In a case like this, one can divide the interval  $[x_0, x_1]$  into a few smaller intervals and solve the PDE problem in each of the resulting smaller  $R$  rectangles. With functions for which there is no difficulty generating Taylor series when using the  $x$  series of line (15.10), it is possible that the accuracy test eventually increases  $J$  beyond the implementation upper bound. For instance, with our implementation, this occurs if  $[x_0, x_1] = [0, \pi]$  and a  $g_i$  function is  $\sin(100x)$ . For such a case, we can use the previous remedy of subdividing  $R$ .

When the accuracy test is passed for all initial value functions, the degree bound  $J$  is set equal to the largest value needed for any  $g_i$ , and an approximating polynomial to this now fixed degree is used in place of every function  $g_i$ . The PDE method is used to obtain series of the general form  $\sum_{j=0}^J \sum_{k=0}^K a_{jk} (x-x_m)^j (t-t_0)^k$  for all  $v_i$  functions and their  $t$  partial derivatives in the  $R$  subrectangle  $[x_0, x_1] \times [t_0, t_0+h]$ . Before giving the description of the first PDE method, we need to obtain the remainder for such a series.

Suppose now that  $f(x, t)$  is an analytic function defined in the subrectangle. According to the remainder equation (6.1), if we hold  $x$  fixed,  $f(x, t)$  may be expanded in a  $t$  Taylor series with the series expansion point  $t = t_0$  as follows:

$$f(x, t) = \sum_{k=0}^{K-1} \frac{1}{k!} \frac{\partial^k f(x, t_0)}{\partial t^k} (t-t_0)^k + \frac{1}{K!} \frac{\partial^K f(x, \hat{t})}{\partial t^K} (t-t_0)^K$$

The point  $\hat{t}$  depends on  $x$ , and is somewhere in  $(t_0, t_0+h)$ . Next, make substitutions in the coefficients of this series, using the following  $x$  Taylor expansions with  $x_m$  as the series expansion point:

$$\frac{\partial^k f(x, t_0)}{\partial t^k} = \sum_{j=0}^{J-1} \frac{\partial^{j+k} f(x_m, t_0)}{j! \partial x^j \partial t^k} (x-x_m)^j + \frac{\partial^{J+k} f(\hat{x}_k, t_0)}{J! \partial x^J \partial t^k} (x-x_m)^J$$

$$k = 0, 1, \dots, K - 1$$

$$\frac{\partial^K f(x, \hat{t})}{\partial t^K} = \sum_{j=0}^{J-1} \frac{\partial^{j+K} f(x_m, \hat{t})}{j! \partial x^j \partial t^K} (x - x_m)^j + \frac{\partial^{J+K} f(\hat{x}_K, \hat{t})}{J! \partial x^J \partial t^K} (x - x_m)^J$$

The points  $\hat{x}_k$  and the point  $\hat{x}_K$  are somewhere in  $(x_0, x_1)$ . We obtain in this way a Taylor series expression with  $J + K + 1$  remainder terms:

$$\begin{aligned} f(x, t) &= \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} \frac{1}{j!k!} \frac{\partial^{j+k} f(x_m, t_0)}{\partial x^j \partial t^k} (x - x_m)^j (t - t_0)^k \\ &+ \sum_{k=0}^{K-1} \frac{1}{J!k!} \frac{\partial^{J+k} f(\hat{x}_k, t_0)}{\partial x^J \partial t^k} (x - x_m)^J (t - t_0)^k \\ &+ \sum_{j=0}^{J-1} \frac{1}{j!K!} \frac{\partial^{j+K} f(x_m, \hat{t})}{\partial x^j \partial t^K} (x - x_m)^j (t - t_0)^K \\ &+ \frac{1}{J!K!} \frac{\partial^{J+K} f(\hat{x}_K, \hat{t})}{\partial x^J \partial t^K} (x - x_m)^J (t - t_0)^K \end{aligned} \tag{15.12}$$

If we generate an interval version of the preceding equation with the typical coefficient being  $f_{jk} \pm w_{jk}$ , then the interval arithmetic equation that corresponds to line (6.2) is

$$\begin{aligned} f(x, t) &= \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} f_{jk} (x - x_m)^j (t - t_0)^k + \sum_{k=0}^{K-1} (f_{jk} \pm w_{jk}) (x - x_m)^J (t - t_0)^k \\ &+ \sum_{j=0}^{J-1} (f_{jK} \pm w_{jK}) (x - x_m)^j (t - t_0)^K + (f_{JK} \pm w_{JK}) (x - x_m)^J (t - t_0)^K \end{aligned} \tag{15.13}$$

The PDE method for finding solution series terms in the first  $t$  subinterval can now be described. The series representation eventually obtained for the functions  $v_i$  is

$$v_i(x, t) = \sum_{j=0}^J \sum_{k=0}^K (v_{i,j,k} \pm w_{i,j,k}) (x - x_m)^j (t - t_0)^k \quad i = 1, \dots, m \tag{15.14}$$

Any  $v_i$  partial derivative with respect to  $x$  that appears in Eq. (15.2) is easily obtained in series form by differentiating the  $v_i$  series an appropriate number of times. With all the  $v_i$  series and their  $x$  derivatives, and the  $t$  and  $x$  series, one can compute the  $f_i$  series:

$$f_i(x, t) = \sum_{j=0}^J \sum_{k=0}^K (f_{i,j,k} \pm w'_{i,j,k}) (x - x_m)^j (t - t_0)^k \quad i = 1, \dots, m \tag{15.15}$$



When the complete series for the functions  $f_i$  are obtained, an accuracy test for the PDEs is possible, which is a test of the  $f_i$  errors. According to Eq. (15.13), we have the error bound  $\epsilon$  if

$$\sum_{k=0}^{K-1} w'_{i,J,k}(w_m)^J h^k + \sum_{j=0}^{J-1} w'_{i,j,K}(w_m)^j h^K + w'_{i,J,K}(w_m)^J h^K < \epsilon \quad i = 1, \dots, m \tag{15.16}$$

The general procedure is similar to the procedure described for ODEs. Again imagine the parameter  $k$  of lines (15.14) and (15.15) is a variable that starts at 0 and rises to  $K$ . For each  $k$  setting, corresponding series terms are computed for all functions  $v_i$  and  $f_i$ . Again the most time-consuming part of the computation occurs when  $k = 0$ , because interval halfwidths for the functions  $v_i$  must be estimated. When  $k = 0$ , we have

$$v_i(x, t_0) = \sum_{j=0}^J (v_{i,j,0} \pm w_{i,j,0}^0)(x - x_m)^j \quad i = 1, \dots, m \tag{15.17}$$

Here the polynomials obtained from the initial conditions have been expanded into series using the  $x$  variable setting shown in Eq. (15.10). The halfwidths obtained,  $w_{i,j,0}^0$ , are valid for the  $[x_0, x_1]$  interval on the initial value line  $t = t_0$  at the bottom of  $R$ , but not for the  $t$  strip, which is  $[x_0, x_1] \times [t_0, t_0 + h]$ . We require instead

$$v_i(x, t) = \sum_{j=0}^J (v_{i,j,0} \pm w_{i,j,0})(x - x_m)^j \quad i = 1, \dots, m \tag{15.18}$$

where  $w_{i,j,0}$  is for this first  $t$  strip. A cyclic procedure is used to assign values to  $w_{i,j,0}$  that are not overly large. Initially, the halfwidths  $w_{i,j,0}$  are set equal to  $w_{i,j,0}^0$ . The functions  $f_i$  are evaluated for  $k = 0$  to obtain the series

$$f_i(x, t) = \sum_{j=0}^J (f_{i,j,0} \pm w'_{i,j,0})(x - x_m)^j \quad i = 1, \dots, m \tag{15.19}$$

The following test is made:

$$w_{i,j,0}^0 + (|f_{i,j,0}| + w'_{i,j,0}) \cdot h < w_{i,j,0}, \quad \begin{matrix} i = 1, \dots, m \\ j = 0, 1, \dots, J \end{matrix} \tag{15.20}$$

If any of these inequalities fail for an  $ij$  combination, then the corresponding halfwidth  $w_{i,j,0}$  is reset according to the plan described previously for ODEs. Whenever any  $w_{i,j,0}$  is increased, the entire halfwidth computation is repeated with the larger halfwidths. More cycles are allowed for this computation than

were allowed for ODEs, and in cases where the test never succeeds, the  $t$  subinterval width  $h$  is halved and the whole process begins anew. After all halfwidths  $w_{i,j,0}$  have been chosen successfully, the other series terms for  $v_i$  and  $f_i$  corresponding to higher  $k$  settings are obtained in a more straightforward fashion. The computation for these other terms of  $v_i$  and  $f_i$  is similar to the procedure described for ODEs.

When the parameter  $k$  reaches its terminal value  $K$ , the accuracy test (15.16) is made on all  $f_i$  series, and a similar test is made on all  $v_i$  series. If these tests are passed, the collection of series is saved for later evaluation, and the whole process starts anew in the next strip  $[x_0, x_1] \times [t^*, t^* + h]$ , with the initial value polynomial for  $v_i$  now taken to be equal to  $\sum_{j=0}^J \sum_{k=0}^K v_{i,j,k}(x - x_m)^j h^k$ . The procedure continues until the  $t$  strips cover  $R$ .

## 15.5 A simple PDE problem as an example

Suppose the rectangle  $R$  is given by

$$R = \begin{matrix} x & t \\ [0, \pi] & \times & [0, 10] \end{matrix}$$

and the single PDE to be solved is  $\partial v_1 / \partial t = \partial v_1 / \partial x$ , with the initial condition  $v_1(x, 0) = \sin(x)$ . This problem has the true solution  $v_1(x, t) = \sin(x + t)$ . Suppose the error bound  $\epsilon$  is chosen to be  $2 \cdot 10^{-6}$ , so that the computed  $v_1$  solution will satisfy the PDE and the initial condition to 5 decimal places.

The first task is to obtain a polynomial that approximates the function  $\sin(x)$  over the interval  $[0, \pi]$  to within  $\epsilon$ . We need a Taylor series for  $\sin x$  with the series expansion point  $x_m = \frac{\pi}{2}$ , the midpoint of  $[0, \pi]$ . Taking the  $x$  series as  $(\frac{\pi}{2} \pm \frac{\pi}{2}) + (1 \pm 0)(x - \frac{\pi}{2})$ , and the degree of the expansion equal to 12, the sin series obtained, with interval coefficients, passes the accuracy test (15.11). The interval halfwidths of the sin expansion may now be discarded, and the degree 12 polynomial that replaces  $\sin x$  and becomes  $v_1(x, 0)$  is

$$P(x) = 1 - \frac{1}{2!}(x - \frac{\pi}{2})^2 + \frac{1}{4!}(x - \frac{\pi}{2})^4 - \dots + \frac{1}{12!}(x - \frac{\pi}{2})^{12} \quad (15.21)$$

The computation for the  $f_1$  and  $v_1$  series on the strip  $[0, \pi] \times [0, h]$  is next. We will do this computation as if it were an ordinary real number computation, and consider later the benefits of doing it in interval arithmetic. Because  $f_1 = \partial v_1 / \partial x$ , the  $f_1$  terms when  $k = 0$  are obtained by differentiation of  $P(x)$  and are

$$-(x - \frac{\pi}{2}) + \frac{1}{3!}(x - \frac{\pi}{2})^3 - \dots + \frac{1}{11!}(x - \frac{\pi}{2})^{11}$$

These  $f_1$  terms get integrated with respect to  $t$  to become the following  $v_1$  terms for  $k = 1$ :

$$-(x - \frac{\pi}{2})(t - 0) + \frac{1}{3!}(x - \frac{\pi}{2})^3(t - 0) - \dots + \frac{1}{11!}(x - \frac{\pi}{2})^{11}(t - 0)$$

These  $v_1$  terms yield the following  $f_1$  terms for  $k = 1$ :

$$-(t-0) + \frac{1}{2!}(x - \frac{\pi}{2})^2(t-0) - \dots + \frac{1}{10!}(x - \frac{\pi}{2})^{10}(t-0)$$

This pattern of computation continues until  $k$  reaches the end value  $K$ , which we take to be 12; the single  $k = 12$  term for  $v_1$ , which is  $\frac{1}{12!}(t-0)^{12}$ , yields no corresponding  $f_1$  term. (The reduction in number of terms as  $k$  advances is an accident of the example, and does not occur if the PDE is changed to  $\frac{\partial v_1}{\partial t} = \frac{\partial v_1}{\partial x} + v_1$ .)

This real number computation delivers an  $x, t$  polynomial for  $v_1$  that approximates the  $v_1$  solution near the line  $t = 0$ , but we do not know how far from this line the approximation may be used. By doing the computation in interval arithmetic in the manner described in the preceding section, we obtain two key benefits. First, the strip width  $h$  is determined. Second, the interval halfwidths associated with the  $v_1$  coefficients “shade in” the  $v_1$  graph (as in the Fig. 15.1 diagram), so that the expected shift in  $f_1$  at the start of the next strip is not a concern.

The computation for  $v_1$  and  $f_1$  on the second strip (and on all later strips) is easier because a  $\sin x$  evaluation is not needed to obtain the beginning  $x$  polynomial. We ignore the interval halfwidths of the first strip’s computed  $v_1(x, t)$  function and use the degree 12 polynomial  $v_1(x, h)$  as the beginning  $x$  polynomial for the second strip.

## 15.6 A defect of the first PDE method

The method described in Section 15.4 has a serious shortcoming for many simple PDE problems. For instance, for the example problem of the preceding section, the computed solution is close to the true solution  $\sin(x+t)$  for  $t$  near the initial value line  $t = 0$ . But it becomes larger than 9 for higher  $t$  values, even though the PDE is approximated to within 5 decimal places, and the initial condition is satisfied to the same accuracy. Recall that the initial condition function  $\sin(x)$  is replaced by the polynomial  $P(x)$  of Eq. (15.21), the degree 12 Taylor polynomial obtained by expanding  $\sin(x)$  in a Taylor series using the series expansion point  $x = \pi/2$ . We can understand the divergence of the computed solution from the true solution by examining the graph of  $P(x)$  shown in Fig. 15.2b, and comparing it with the graph of  $\sin(x)$ , shown in Fig. 15.2a. Notice that  $P(x)$  is quite close to  $\sin(x)$  in the interval  $[0, \pi]$ , but then eventually rises to infinity on either side of  $\pi/2$ , as any even-degree Taylor polynomial must do. When the initial condition function for the example PDE is  $P(x)$  instead of  $\sin(x)$ , the true solution becomes  $P(x+t)$  instead of  $\sin(x+t)$ . To obtain a computed solution close to  $\sin(x+t)$  on the rectangle  $R$ , the Taylor polynomial  $P(x)$  must be of such high degree that it is close to  $\sin(x)$  not just on the interval  $[0, \pi]$ , but on the wider interval  $[0, \pi + 10]$  required by  $R$ .

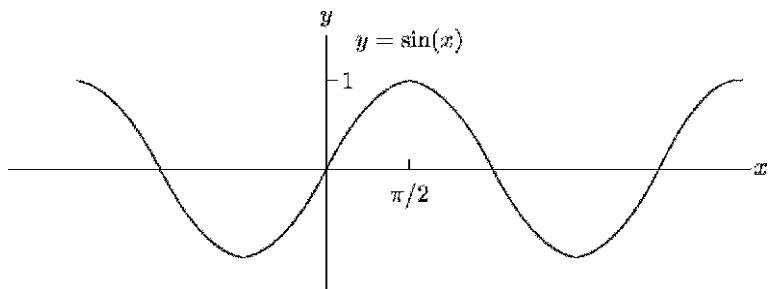


Fig. 15.2a

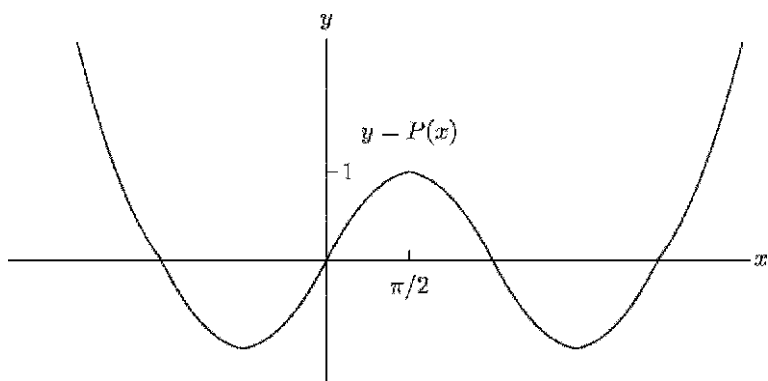


Fig. 15.2b

For any PDE initial value problem over a rectangle  $R$ , we see that the initial value polynomials  $v_i(x, t_0)$  need to be accurate approximations to the initial value functions  $g_i$  over an interval wider than the interval  $[x_0, x_1]$ , with the size of the interval depending on the PDEs and the dimensions of the rectangle  $R$ .

## 15.7 The revised PDE method with comparison computation

A more accurate solution method is obtained by making a series of trial runs to determine how large to make the degree  $J$  in the series expansions of lines (15.14) and (15.15).

Let the initial value polynomial for  $v_i$  on any  $t$  subinterval be  $\sum_{j=0}^J v_{i,j}(x - x_m)^j$ . Recall that  $x_m$  is the midpoint of  $[x_0, x_1]$  and the  $x$  series is

$$x = (x_m \pm w_m) + (1 \pm 0)(x - x_m)$$

The previously described procedure generates another  $v_i$  polynomial to be used in the next  $t$  subinterval  $[t^*, t^* + h]$ . Let this initial value polynomial be designated  $\sum_{j=0}^J v_{i,j}^*(x - x_m)^j$ . On a trial run, the computation for the starred polynomial is done twice, once at the chosen degree  $J$  and once at the higher degree  $J' = J + \Delta J$ . The two  $J'$  degree polynomials for  $v_i$  are  $\sum_{j=0}^{J'} v'_{i,j}(x - x_m)^j$  and  $\sum_{j=0}^{J'} v_{i,j}^*(x - x_m)^j$ . The two computed starred polynomials differ in value at any point  $(x, t^*)$  by no more than  $\delta_i = \sum_{j=0}^{J'} |v'_{i,j} - v_{i,j}^*| (w_m)^j$ . In this expression take  $v_{i,j}^* = 0$  for  $j > J$ .

On the first trial run, as previously described,  $J$  is chosen high enough to make the polynomial error  $< \epsilon$  on the line  $t = t_0$ . A trial run continues from one  $t$  subinterval to the next if we obtain  $\delta_i < \epsilon$  for all  $i$ . Whenever this result is not obtained, the trial run ends,  $J$  is reset to the higher value  $J'$ , and a new trial run ensues, starting once more at  $t = t_0$ . Eventually  $J$  is high enough for the trial run to go to completion. When this occurs the successful run is repeated one last time, without doing duplicate computation, to store the  $v_i$  and  $f_i$  expansions of lines (15.14) and (15.15) for later evaluation at points of interest in  $R$ .

The series of trial runs is not needed if all initial functions  $g_i$  are polynomials of degree less than or equal to the starting degree  $J$ . In this case, the quantities  $w'_{i,j}$  of the line (15.11) test are zero. This occurrence signals that increasing  $J$  to  $J + \Delta J$  does not change the beginning  $v_i$  polynomials.

## 15.8 Higher dimensional spaces

Although we have posed the PDE problems and examples in  $(x, t)$  space, the method is applicable to  $(x, y, t)$  or  $(x, y, z, t)$  space. The relation (15.13) and test of line (15.16) for  $(x, t)$  space can be generalized without difficulty to apply to the higher dimensional cases.

The computation time for a  $t$  subinterval is roughly proportional to the number of terms in the beginning polynomial for  $v_i$ . In  $(x, t)$  space, the  $v_i$  polynomial is

$$\sum_{j=0}^J v_{i,j}(x - x_m)^j$$

and has  $J + 1$  terms. In  $(x, y, t)$  space, the corresponding polynomial is

$$\sum_{j=0}^J \sum_{j_1+j_2=j} v_{i,j_1,j_2}(x - x_m)^{j_1}(y - y_m)^{j_2}$$

and has  $\frac{1}{2!}(J+1)(J+2)$  terms. And in  $(x, y, z, t)$  space the polynomial is

$$\sum_{j=0}^J \sum_{j_1+j_2+j_3=j} v_{i,j_1,j_2,j_3}(x - x_m)^{j_1}(y - y_m)^{j_2}(z - z_m)^{j_3}$$

and has  $\frac{1}{3!}(J+1)(J+2)(J+3)$  terms. For example, with  $J$  presumed to be 12, the various execution times for a  $t$  subinterval in  $(x, t)$ ,  $(x, y, t)$ , and  $(x, y, z, t)$  spaces may be expected to be roughly proportional to the numbers 13, 91, and 455, respectively.

Because the test of line (15.16) involves more terms as the space dimension increases, the  $t$  subintervals become correspondingly smaller, further increasing the execution time for higher dimensional problems.

### 15.9 Satisfying boundary conditions

With the described method, boundary conditions cannot be directly imposed. If there are boundary conditions for a PDE problem, these must be obtained by restricting the initial value functions  $g_i$  appropriately, so that the boundary conditions come about as a consequence of the PDEs.

Suppose, for instance, that the boundary condition  $v_1(t, x_0) = b(t)$  is required for the function  $v_1$ , and that  $b(t)$  is an elementary analytic function of  $t$ . At the point  $t = t_0$  we have the Taylor expansion

$$b(t) = b_0 + b_1(t - t_0) + b_2(t - t_0)^2 + b_3(t - t_0)^3 + \dots$$

The  $v_1$  initial value function  $g_1$ , besides being restricted by the equation  $g_1(x_0) = b_0$ , has additional requirements. In the first  $t$  subinterval, the  $v_1$  differential equation of line (15.2) will be satisfied to high accuracy, and this implies that at the point  $(x_0, t_0)$  we have

$$b_1 = \frac{\partial v_1}{\partial t} = f_1\left(x_0, t_0, v_1, \dots, \frac{\partial^{p_1} v_1}{\partial x^{p_1}}, v_2, \dots, \frac{\partial^{p_2} v_2}{\partial x^{p_2}}, \dots, v_m, \dots, \frac{\partial^{p_m} v_m}{\partial x^{p_m}}\right)$$

Thus the differential equation connects the Taylor coefficient  $b_1$  with the initial value functions. Because of our assumption of analyticity for all functions, we can take higher  $t$  derivatives and obtain relations connecting the other Taylor coefficients of  $b(t)$  with the initial value functions.

As an example, consider the parabolic heat equation  $\frac{\partial v}{\partial t} = c^2 \frac{\partial^2 v}{\partial x^2}$  for a heat conducting bar of length  $L$  lying on the  $x$  interval  $[0, L]$ , where  $v(x, t)$  is the temperature, and  $c^2$  is a positive constant. If the left endpoint of the bar is maintained at the temperature  $T$ , this leads to the boundary condition  $v(0, t) = T$ . The differential equation implies that

$$0 = \frac{\partial^j v(0, 0)}{\partial t^j} = c^{2j} \frac{\partial^{2j} v(0, 0)}{\partial x^{2j}} \quad j = 1, 2, \dots$$

The initial value function  $g(x) = v(x, 0)$ , besides satisfying the relation  $g(0) = T$ , must have all even order derivatives equal to zero at  $x = 0$ .

If the left endpoint of the bar is perfectly insulated, this leads to the boundary condition  $\partial v(0, t)/\partial x = 0$ . The differential equation implies that

$$0 = \frac{\partial^{j+1} v(0, 0)}{\partial t^j \partial x} = c^{2j} \frac{\partial^{2j+1} v(0, 0)}{\partial x^{2j+1}} \quad j = 1, 2, \dots$$

This time every odd order derivative of  $g(x)$  must be zero at  $x = 0$ .

## Software Exercises M

These exercises are with the demo program `pde`. Because of the long computation needed to solve a typical initial value problem, the program `pde` chooses the accuracy goal, which is unlike the other demo programs. It chooses 5 decimal-place accuracy for a 1- or 2-dimensional problem, 3 decimal-place accuracy for a 3-dimensional problem, and 1 decimal-place accuracy for a 4-dimensional problem. The program `pde` creates a log file, but does not create a print file. After the program approximates the solution to a PDE or ODE initial value problem, it enters a loop in which it repeatedly requests values for the independent variables, and then displays the computed PDE or ODE solution at the designated point, as well as all the derivatives that occur in the differential equations. To exit the loop, type the single letter **q** (for “quit”) and hit the ENTER key.

1. Call up `pde` and solve the example problem of Section 15.6 by taking  $[x_0, x_1]$  as  $[0, \pi]$  and  $[t_0, t_1]$  as  $[0, 10]$ , entering the differential equation  $\partial u/\partial t = \partial u/\partial x$ , choosing the  $t = t_0$  line for initial conditions, and choosing  $\sin(x)$  as the  $u$  initial value function. The true solution is  $u = \sin(x+t)$ .
2. Edit the log file to change the  $t$  interval from  $[0, 10]$  to  $[0, 100]$  and then call up `pde pde`. The `pde` program now repeatedly increases the degree of its series computations because the comparison computations indicate this is needed. Whenever it becomes clear that the size of the  $t$  interval is too large, one should stop the `pde` computation by entering a **control-c**, and then edit the log file to reduce the  $t$  interval.
3. Call up `pde` and solve the equation  $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$ , specifying  $[0, 10]$  as the interval for each independent variable, in this case  $x, y, z$ , and  $t$ ; supply an  $x, y, z$  polynomial of moderate degree, such as  $x^3 + y^2 + xyz$ , as the initial condition function  $u(x, y, z)$  at  $t = t_0$ . For polynomial initial conditions of degree  $< 12$ , the `pde` program is able to obtain the polynomial solution in  $x, y, z$  and  $t$  quickly. For the example initial condition, the true solution is  $x^3 + y^2 + xyz + (6x + 2)t$ .
4. Call up `pde` and solve the ODE initial value problem  $d^2 u/dt^2 = -u$  over the  $t$  interval  $[0, 20\pi]$ , with the initial conditions  $u(0) = 0$  and  $du(0)/dt = 1$ .

The true solution to this problem is  $u = \sin(t)$ . The `pde` program can solve ODE initial value problems, but unlike the program `difsys`, the program `pde` does not attempt to approximate the true solution. Nevertheless, its solutions are often surprisingly accurate.

## Notes and References

- A. Brian Hassard of SUNY at Buffalo and his students [1] [2] made early precise computations of specific PDE problems.
- [1] B. Hassard and S. Zhixin, Precise solution of Laplace's equation, *Math. Comp.* **64** (1995), 515–536.
  - [2] X. Hongliand, *Precise Solution of Wave Equation*, Ph.D. Dissertation, SUNY at Buffalo, June, 1998.



This page intentionally left blank

# Numerical Methods with Complex Functions



This chapter requires some acquaintance with complex analysis, to the extent of understanding the differentiation and integration of complex functions.

A number of demo programs have their computation systems explained in this chapter. They are `c_calc` for evaluating complex constants, `c_fun` for evaluating complex functions, `c_deriv` for computing a complex function's derivatives, `c_integ` for computing the line integral of a complex function within the complex plane, `c_roots` for finding a complex polynomial's roots, and `c_zeros` for finding the zeros of a complex function or the zeros of several complex functions.

## 16.1 Elementary complex functions

Like elementary real functions, the elementary complex functions are frequently encountered functions for which it is easy to form power series and to compute derivatives.

**Definition 16.1** A function of a finite number of complex variables  $z_1, z_2, \dots, z_n$  is an *elementary complex function* if it can be expressed in terms of its variables and complex constants  $c_1, c_2, \dots, c_m$  by a finite number of the binary operations of addition, subtraction, multiplication, division, or exponentiation, and the unary operations of function evaluation with  $\sin$ ,  $\cos$ ,  $\tan$ ,  $\sin^{-1}$ ,  $\cos^{-1}$ ,  $\tan^{-1}$ ,  $\exp$ , or  $\log$ .

For some elementary real functions, there is no elementary complex analogue. For instance, the max or min elementary real functions do not have complex versions. And the function  $|z|$ , though relatively easy to compute, is not an elementary complex function, even though  $|x|$  is an elementary real function.

Some issues concerning function evaluation need to be addressed. The inverse of the exponential function  $e^z = e^x(\cos y + i \sin y)$  is  $\log z$ , and it is well known that it is not possible to define  $\log z$  in the complex plane without introducing a “cut line” on which the function is discontinuous. This cut line is usually taken as the negative real axis, and the definition of  $\log z$  is then  $\ln |z| + i\Theta(z)$ , where  $\ln$  denotes the natural logarithm (to base  $e$ ), and  $\Theta(z)$  gives the *principal argument* of  $z$ , that is, an angle in radians between  $\pi$  and  $-\pi$ . As  $z$  approaches a point  $x_0$  on the negative real axis from above,  $\log z$  approaches  $\ln |x_0| + i\pi$ , and as  $z$  approaches this point from below,  $\log z$  approaches  $\ln |x_0| - i\pi$ , so  $\log z$  is discontinuous on the cut line. In accordance with the view of how discontinuous functions should be treated numerically, as explained in Section 3.7, we take  $\log z$  to be undefined on the negative real axis, and of course also at the origin because  $\ln |z|$  is not defined there.

The computation of  $z^a$ , where the exponent  $a$  may be any real or complex number, is done in the following way. If  $a = 0$ , then  $z^a = 1$ . If  $a$  is a positive integer, then  $z^a$  is formed by repeated multiplication and is defined over the entire complex plane. If  $a$  is a negative integer, then  $z^a$  is computed as  $(1/z)^{-a}$  and is defined everywhere except at the origin. Now suppose  $a$  is some real or complex number other than a real integer. In this case we take  $z^a$  to be defined by the relation

$$z^a = e^{a \log z}$$

This implies that  $z^a$  is undefined along the negative real axis and origin because the log function is not defined there.

With our range arithmetic routines for  $z^a$ , whenever  $a$  and  $z$  are real, with the imaginary parts of  $a$  and  $z$  being exact zeros, the complex operation  $z^a$  defers to the real operation  $x^a$ . If  $a$  is a rational number  $p/q$  that is in reduced form with the integer  $q$  positive, the usual interpretation of  $x^a$  is

$$x^{\frac{p}{q}} = (\sqrt[q]{x})^p = \sqrt[q]{x^p}$$

The real range arithmetic operations for  $x^a$  employ this interpretation, and so the complex operation  $z^a$  also obtains this interpretation when both  $z$  and  $a$  are real.

The inverse functions  $\sin^{-1} z$ ,  $\cos^{-1} z$ , and  $\tan^{-1} z$  are computed in the complex plane according to the equations

$$\sin^{-1} z = -i \log [iz + (1 - z^2)^{\frac{1}{2}}]$$

$$\cos^{-1} z = \frac{\pi}{2} - \sin^{-1} z$$

$$\tan^{-1} z = \frac{i}{2} \log \frac{i+z}{i-z}$$

These equations imply that  $\sin^{-1} z$  and  $\cos^{-1} z$  are undefined when  $1 - z^2$  is negative real, which occurs when  $z$  is real and greater in absolute value than 1.

These are the only points where these two functions are undefined, because in the equation for  $\sin^{-1} z$ , the log argument is never a negative real number or zero. On the other hand, for  $\tan^{-1} z$ , the log argument is negative real for  $z = iy$  with  $|y| > 1$ , and the log argument is either not defined or zero when  $y$  equals 1 or  $-1$ . So  $\tan^{-1} z$  is not defined on the imaginary axis except for the open interval between  $-1$  and  $+1$ .

These notes help explain the complex values that the program `c_calc` generates in response to a user's entry of a complex constant. A complex number  $c = x + iy$  in range arithmetic has two parts that are real ranged numbers. These of course are the real  $x$  component and the real  $y$  component. Computation of complex constants is done via an evaluation list in the same way that the computation of real constants is done.

The program `c_fun` evaluates complex functions in much the same way real functions are evaluated, described in Chapter 4, except that complex range arithmetic replaces the real range arithmetic.

## 16.2 The demo program `c_deriv`

The series relations developed in Chapter 5 for computing values and derivatives of real functions can be extended to complex functions. However, some changes in notation are needed. For instance, the opening equation (5.1) now is replaced by

$$f(z) = a_0 + a_1(z - z_0) + a_2(z - z_0)^2 + \cdots + a_k(z - z_0)^k + \cdots \quad (16.1)$$

Most of the series relations presented in Chapter 5 apply to elementary complex functions, but with obvious notation changes.

The `c_deriv` program for elementary complex functions is similar to the `deriv` program, and uses complex power series in place of real power series. Like the `deriv` program, `c_deriv` checks the coefficients of the generated series to determine whether enough correct decimal places are obtained. If not, precision is increased and the series is recomputed. This process continues until the test for decimal place accuracy is passed.

## 16.3 Computing line integrals in the complex plane

The program `c_integ` computes the complex plane integral  $\int_C f(z) dz$ , where  $f(z)$  is an elementary complex function, and the curve  $C$  in the complex plane is either a straight line segment or a closed circle. The method of computation is similar to the method previously described for the program `integ`.

Consider first the case where the curve  $C$  is the straight line segment that begins at the complex point  $a$  and ends at the complex point  $b$ . Here the path of

integration is defined by the equation  $z = a + (b - a) \cdot u$ , where the real variable  $u$  starts at 0 and ends at 1. We have then

$$\int_C f(z) dz = (b - a) \int_0^1 f(a + (b - a)u) du$$

The integral over the  $u$  interval  $[0, 1]$  has a real part and an imaginary part, and the errors of the two parts are each monitored in the way the `integ` program monitors error. If the error of one of these parts is larger than the error allotment for a  $u$  subinterval of that size, then the  $u$  subinterval is divided into two equal-length parts, and the integration reattempted for these smaller subintervals.

If the curve  $C$  is a circle with centerpoint  $m$  and radius  $r$ , the path of integration is defined by the equation  $z = m + r \cdot e^{2\pi i u}$ , where again the real variable  $u$  starts at 0 and ends at 1. In this case we have

$$\int_C f(z) dz = 2\pi i r \int_0^1 f(m + r e^{2\pi i u}) e^{2\pi i u} du$$

This integral over the  $[0, 1]$   $u$  interval can be obtained accurately using the method described for the previous integral.

## 16.4 Computing the roots of a complex polynomial

We explain here the procedure by which the program `c_roots` finds accurate approximations to the roots of a complex polynomial  $P(z)$ . The dissection of  $P(z)$  into subpolynomials  $N_i(z)$  is done similarly to what was described for the demo program `roots`, except of course that complex range arithmetic is used in place of real range arithmetic. This dissection makes finding multiple roots easier.

The roots of each polynomial  $N_i(z)$  are found one by one, because the complex roots of a complex polynomial do not come in conjugate pairs; so the Bairstow method cannot be used. A complex version of Newton's method is used instead. The Taylor series expansion for an analytic function  $f(z)$  at a series expansion point  $w_k$  has the leading terms

$$f(z) = f(w_k) + f'(w_k)(z - w_k) + \dots$$

If we follow the usual reasoning of Newton's method and approximate  $f(z)$  by using just the first two terms of its series expansion, then setting  $f(z)$  equal to zero, solving for  $z$ , and calling our solution  $w_{k+1}$ , we obtain the complex Newton's method iteration equation

$$w_{k+1} = w_k - \frac{f(w_k)}{f'(w_k)} \quad (16.2)$$

This iteration equation can be used to find roots of  $N_i(z)$  if the progress of the iteration is closely monitored. A convenient point  $w_0$  is chosen within a computed

root bounding circle  $|z| \leq R$ . (The circular bounds derived in Section 8.2 for real polynomials also are valid for complex polynomials.). As each iterate  $w_k$  is found, some indicator of progress toward a root is also needed. This could be the quantity  $|N_i(w_k)|$ , which should decrease with each iteration.

For the complex number  $w_k = x_k + iy_k$ , define midpoint  $w_k$  as midpoint  $x_k + i(\text{midpoint } y_k)$ , and define  $w_k \doteq w_{k+1}$  as  $x_k \doteq x_{k+1}$  and  $y_k \doteq y_{k+1}$ . After each iteration cycle, we check whether  $w_{k+1} \doteq w_k$ , and if this relation does not hold, then  $w_{k+1}$  is replaced by midpoint  $w_{k+1}$  in preparation for the next iteration cycle. After we finally obtain  $w_{k+1} \doteq w_k$ , then midpoint  $w_k$  can serve as the final root approximation  $w_z$ . If the degree of the problem polynomial  $N_i(z)$  is greater than 1, the factor  $z - w_z$  is divided into the problem polynomial to get one of lower degree, and the process is repeated with the new polynomial in order to obtain another root approximation. The procedure continues until approximations to all the roots have been assembled. Error bounds for these roots are computed in the way described in Section 8.5.

## 16.5 Finding a zero of an elementary complex function $f(z)$

Suppose the elementary complex function  $f(z)$  is defined in some rectangle  $R$  in the complex plane, and we want to find all the zeros of  $f(z)$  in  $R$ . Here  $R$  is specified by inequalities applying to the real and imaginary parts of  $z = x + iy$ , that is, by inequalities  $a \leq x \leq b$  and  $c \leq y \leq d$ . An  $f(z)$  zero  $z_0$  is *simple* if  $f'(z_0) \neq 0$ . The zero is a multiple zero with multiplicity  $m$ , if  $f^{(m)}(z_0) \neq 0$  and

$$0 = f(z_0) = f'(z_0) = f''(z_0) = \cdots = f^{(m-1)}(z_0)$$

The beginning solvable problem of Chapter 7, finding the zeros for an elementary real function  $f(x)$  in a search interval  $[a, b]$ , requires that  $f(x)$  test nonzero on the boundary of  $[a, b]$ . Similarly, to obtain a solvable problem here, we must require that  $f(z)$  test nonzero on the boundary of  $R$ .

In Section 7.1, for the problem of finding zeros of  $f(x)$  in  $[a, b]$ , two computation difficulties were described, illustrated by specific functions  $f_d(x)$  and  $f_e(x)$ . For our problem, a function analogous to  $f_d(x)$  is

$$f_d(z) = (z - 1)^2 + |d|$$

where  $d$  is any real number. Suppose  $R$  is some rectangle enclosing the point  $1 + 0i$ . If  $d$  is 0, there is a zero of multiplicity 2 at  $z = 1$ . If  $d$  is unequal to 0, but near 0, instead of having no zero, as would be the case for  $f_d(x)$ , there are the two zeros  $1 \pm i\sqrt{|d|}$  within  $R$ . Thus we do not need to report "possible zeros" for the  $f(z)$  problem. However, note the difficulty here in deciding whether we have found a multiple zero. If  $d = 0$ , the number 1 is a multiple zero, but if  $d \neq 0$  there are 2 simple zeros. If we could decide the multiplicity of a zero correctly for every  $d$ , we could determine whether any real number is zero,

contradicting Nonsolvable Problem 3.1. Thus the multiplicity of any zero not designated “simple”, must be reported as the zero’s “apparent multiplicity”.

The other difficulty mentioned in Section 7.1 is the possibility that  $f(x)$  might be zero on a subinterval inside  $[a, b]$ , and thus have an infinite number of zeros. This difficulty cannot occur for an elementary complex function  $f(z)$  that is defined throughout  $R$  and is nonzero on the  $R$  boundary. This is because an elementary complex function defined at all points in  $R$  is analytic in the interior of  $R$ , and if such a function is 0 on a line segment in  $R$ , it is 0 everywhere in  $R$ , and would not pass the  $R$  boundary test.

These considerations make the solvable problem for  $f(z)$  simpler than the corresponding problem for  $f(x)$ . First we need some terminology for the elementary complex function problem: A *search rectangle*  $R$  is a rectangle in the complex plane defined by four constants  $a, b, c, d$ , with  $z = x + iy$  restricted by inequalities  $a \leq x \leq b$  and  $c \leq y \leq d$ . A zero is determined to  $k$  decimal places by giving both its real and imaginary parts to  $k$  decimal places.

**Solvable Problem 16.1** For any elementary complex function  $f(z)$  defined in a search rectangle  $R$ , and any positive integer  $k$ , find a point  $z_0$  on the boundary of  $R$  where  $|f(z_0)| < 10^{-k}$  and halt. Or if  $f(z) \neq 0$  on the boundary of  $R$ , bound all the zeros of  $f$  in  $R$  by

- (1) giving, to  $k$  decimal places, points identified as simple zeros,
- (2) giving, to  $k$  decimal places, points identified as zeros of apparent multiplicity  $m$ , with the integer  $m \geq 2$ .

If a real number  $r$  is displayed to  $k$  correct decimal places, the number  $r$  may lie anywhere in the corresponding tilde-interval. Similarly, in the complex plane a complex number  $z = x + iy$  displayed to  $k$  decimal places may lie anywhere in a tilde-rectangle centered at the displayed value, with sides parallel to the real and imaginary axes. If a zero of apparent multiplicity  $m$  is displayed to  $k$  decimal places, there are exactly  $m$  zeros within the tilde-rectangle, if we add the multiplicities of the various individual zeros. For instance, if  $m = 3$ , then within the tilde-rectangle there could be one zero of multiplicity 3, there could be three simple zeros, or there could be a simple zero and a zero of multiplicity 2. If the number of correct decimal places is increased, it is possible but not certain that the zero of apparent multiplicity 3 gets changed into 3 simple zeros, or changed into a simple zero and a zero of apparent multiplicity 2.

We have

$$f(z) = f(x + iy) = u(x, y) + iv(x, y)$$

so our problem may be considered equivalent to the problem of finding the zeros of the two real functions  $u(x, y)$  and  $v(x, y)$  in the real  $(x, y)$  space rectangle  $R$ . Accordingly, the concept of crossing number, defined for real elementary functions, can be used for our problem. For an analytic function  $f(z)$  we have

$f'(z) = \frac{\partial u}{\partial x} + i \frac{\partial v}{\partial x}$ , and the Cauchy-Riemann equations  $\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$  and  $\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$  hold. So the Jacobian for  $u$  and  $v$  at a point  $(x, y)$  is

$$\det \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} = \det \begin{bmatrix} \frac{\partial u}{\partial x} & -\frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial x} & \frac{\partial u}{\partial x} \end{bmatrix} = \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 = |f'(z)|^2$$

Thus the Jacobian is always positive or zero. This implies, when computing the crossing number, that the normal  $\mathbf{n}$  defined by equation 12.3 always points outward from the boundary image, so a computed crossing number must be positive or zero. At any simple zero of  $u$  and  $v$  where the Jacobian is nonzero, the Jacobian is actually positive. So if  $f(z)$  has exactly  $k$  simple zeros inside the problem rectangle  $R$ , the corresponding  $f(z)$  crossing number is  $k$ . If in  $R$  the function  $f(z)$  has exactly one multiple zero of multiplicity  $m$ , the corresponding crossing number is  $m$ , using a simple continuity argument where  $m$  simple zeros combine to form the multiple zero.

If the analytic function  $f(z)$  is defined inside and on the rectangle  $R$ , and  $f(z)$  is nonzero on  $R$ , then the number of zeros inside  $R$  can be determined by using the *argument principle*, namely that the number of zeros in  $R$ , counting multiplicities, equals the integral

$$\frac{1}{2\pi i} \int_R \frac{f'(z)}{f(z)} dz$$

where the integration around  $R$  is done counterclockwise. The integral displayed is equivalent to the corresponding Kronecker's integration formula, mentioned in Section 12.5. Thus the argument principle is equivalent to the statement that the number of zeros equals the crossing number for the real functions  $u(x, y)$  and  $v(x, y)$  over the real rectangle  $R$ .

Because we always know the number of zeros of  $f(z)$  inside a search rectangle by computing the corresponding crossing number, this simplifies the task of determining those zeros. The search procedure can be done with a task queue holding problem rectangles to be investigated; the task queue initially holds just the beginning rectangle  $R$ . If  $R_Q$  is the first rectangle on the queue, let  $n(R_Q)$  be the number of zeros inside  $R_Q$ , obtained by computing the crossing number. If  $n(R_Q) = 0$ , we discard  $R_Q$ . If  $n(R_Q) = 1$ , the zero must be simple, and we try to find the zero by the complex version of Newton's method, described in the preceding section, using the centerpoint of our rectangle as the initial iteration point. If there is some difficulty with the iteration, then we bisect  $R_Q$  in its longer dimension, find which subrectangle contains the zero by computing the crossing number for either subrectangle, discard the unneeded subrectangle, and resume the Newton's method attempt with a smaller rectangle. Eventually we will be able to determine the simple zero to the required number of decimal places.



If  $n(R_Q) > 1$ , we bisect  $R_Q$  by a division of its larger side, or a division of either side if the two sides test equal in length, obtaining two problem rectangles  $R_{Q_1}$  and  $R_{Q_2}$ . Because of the additive property of crossing numbers, we have  $n(R_Q) = n(R_{Q_1}) + n(R_{Q_2})$ . So a computation of  $n(R_{Q_1})$  also determines  $n(R_{Q_2})$ . However, if there is a multiple zero  $z_0$  inside  $R_Q$  with multiplicity  $m$  equal to  $n(R_Q)$ , we obtain a series of subrectangles enclosing  $z_0$ , and for these subrectangles the crossing number never changes, and never becomes 1, which is the criterion for using Newton's method. The subrectangles surrounding  $z_0$  diminish in size until one is obtained that falls within the tilde-rectangle of its centerpoint if it were displayed to the requisite  $k$  decimal places. The centerpoint then is displayed, but only as a zero of apparent multiplicity  $m$ , because, as far as we know, there could be several separate zeros inside the tilde-rectangle, with multiplicities summing to  $m$ .

We have glossed over a potential difficulty in the procedure, in that when we bisect  $R_Q$ , it may happen that a zero lies on the common side of the two subrectangles. In this case we can not compute the crossing number for  $f(z)$  over a subrectangle, and so can not determine the zero count for either subrectangle. Some means of avoiding this trouble must be provided. Before we accept a bisection of  $R_Q$ , we do an interval arithmetic computation of  $f(z)$  over the line segment dividing  $R_Q$ . If the real part interval or the imaginary part interval for  $f(z)$  does not contain the zero point, the bisection is accepted. Otherwise, we divide the segment into two subsegments and try the interval computation again for each subsegment. This is another task queue computation, where we continue subdividing until the subsegment at the head of the queue has a halfwidth below a preset bound  $\delta$ , at which point the process is halted. Of course if the task queue becomes empty before this happens, the bisection is accepted. In the case of bisection failure, a new division line parallel to but somewhat removed from the previous bisection line is chosen, and the segment check is repeated. Because there are only  $n(R_Q)$  zeros in  $R_Q$ , a suitable division line is certain to be found eventually.

## 16.6 The general zero problem for elementary complex functions

The problem we consider here is finding where  $n$  elementary complex functions of  $n$  variables are simultaneously zero. The equations that must be solved are

$$\begin{aligned}
 f_1(z_1, z_2, \dots, z_n) &= 0 \\
 f_2(z_1, z_2, \dots, z_n) &= 0 \\
 \vdots & \qquad \qquad \qquad \vdots \\
 f_n(z_1, z_2, \dots, z_n) &= 0
 \end{aligned}
 \tag{16.3}$$

The region of interest is defined by restricting each variable  $z_i$  to a rectangle of its complex domain, or, equivalently, restricting the real part and the imaginary part of each variable  $z_i = x_i + iy_i$  to a finite interval, as shown below:

$$a_i \leq x_i \leq b_i \text{ and } c_i \leq y_i \leq d_i, \quad i = 1, 2, \dots, n \quad (16.4)$$

It is convenient to use vector notation once more, with  $\mathbf{z}$  denoting a vector with components  $z_1, z_2, \dots, z_n$ , and  $\mathbf{f}(\mathbf{z})$  denoting the function with the components shown in (16.3). The function  $\mathbf{f}$  is counted elementary if all its components are elementary. For lack of any better term, we still call the domain (16.4) a box. The appropriate generalization of Problem 16.1 is

**Solvable Problem 16.2** For any elementary analytic function  $\mathbf{f}(\mathbf{z})$  defined on a box  $B$ , and any positive integer  $k$ , find a point  $\mathbf{z}_0$  on the boundary of  $B$  where every component of  $\mathbf{f}(\mathbf{z}_0)$  is less in magnitude than  $10^{-k}$  and halt. Or if  $\mathbf{f}(\mathbf{z}) \neq \mathbf{0}$  on the boundary of  $B$ , bound all the zeros of  $\mathbf{f}$  in  $B$  by

- (1) giving, to  $k$  decimal places, points identified as simple zeros, or
- (2) giving, to  $k$  decimal places, points identified as zeros of apparent multiplicity  $m$ , with the integer  $m \geq 2$ .

A “zero of apparent multiplicity  $m$ ” is a zero such that  $m$  is the crossing number over a small domain enclosing the zero.

A procedure for solving this problem is obtained by generalizing the procedure described for solving the previous problem, where there was just one function  $f(z)$ . In that procedure, crossing number computations using the two component functions  $u(x, y)$  and  $v(x, y)$  are the means of isolating the zeros. We only need to show that the crossing number computed for  $\mathbf{f}(\mathbf{z})$ , considered as  $2n$  component elementary real functions of  $2n$  real variables bounded in intervals, cannot be negative. With the help of the next theorem, this result follows by similar reasoning as was given in Section 16.5.

**Theorem 16.1** For an analytic function  $\mathbf{f}(\mathbf{z})$ , the real Jacobian  $\partial(u_1, v_1, \dots, u_n, v_n)/\partial(x_1, y_1, \dots, x_n, y_n)$  equals the square of the absolute value of the complex Jacobian  $\partial(f_1, \dots, f_n)/\partial(z_1, \dots, z_n)$ .

After the Cauchy-Riemann equations are used to replace all partial derivatives with respect to the variables  $y_k$ , the real Jacobian

$$\det \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial y_1} & \cdots & \cdots & \frac{\partial u_1}{\partial x_n} & \frac{\partial u_1}{\partial y_n} \\ \frac{\partial v_1}{\partial x_1} & \frac{\partial v_1}{\partial y_1} & \cdots & \cdots & \frac{\partial v_1}{\partial x_n} & \frac{\partial v_1}{\partial y_n} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ \frac{\partial u_n}{\partial x_1} & \frac{\partial u_n}{\partial y_1} & \cdots & \cdots & \frac{\partial u_n}{\partial x_n} & \frac{\partial u_n}{\partial y_n} \\ \frac{\partial v_n}{\partial x_1} & \frac{\partial v_n}{\partial y_1} & \cdots & \cdots & \frac{\partial v_n}{\partial x_n} & \frac{\partial v_n}{\partial y_n} \end{bmatrix}$$

takes the form

$$\det \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & -\frac{\partial v_1}{\partial x_1} & \cdots & \cdots & \frac{\partial u_1}{\partial x_n} & -\frac{\partial v_1}{\partial x_n} \\ \frac{\partial v_1}{\partial x_1} & \frac{\partial u_1}{\partial x_1} & \cdots & \cdots & \frac{\partial v_1}{\partial x_n} & \frac{\partial u_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ \frac{\partial u_n}{\partial x_1} & -\frac{\partial v_n}{\partial x_1} & \cdots & \cdots & \frac{\partial u_n}{\partial x_n} & -\frac{\partial v_n}{\partial x_n} \\ \frac{\partial v_n}{\partial x_1} & \frac{\partial u_n}{\partial x_1} & \cdots & \cdots & \frac{\partial v_n}{\partial x_n} & \frac{\partial u_n}{\partial x_n} \end{bmatrix} \tag{16.5}$$

The Jacobian matrix of line (16.5) is composed of submatrices having the general form  $\begin{bmatrix} a & -b \\ b & a \end{bmatrix}$ . Consider the complex matrix

$$A = \begin{bmatrix} 1 & 1 \\ -i & i \end{bmatrix}$$

with inverse

$$A^{-1} = \frac{1}{2} \begin{bmatrix} 1 & i \\ 1 & -i \end{bmatrix}$$

For any real constants  $a$  and  $b$  we have

$$\begin{aligned} A^{-1} \begin{bmatrix} a & -b \\ b & a \end{bmatrix} A &= \frac{1}{2} \begin{bmatrix} 1 & i \\ 1 & -i \end{bmatrix} \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -i & i \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} a+ib & -b+ia \\ a-ib & -b-ia \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -i & i \end{bmatrix} \\ &= \begin{bmatrix} a+ib & 0 \\ 0 & a-ib \end{bmatrix} = \begin{bmatrix} a+ib & 0 \\ 0 & \overline{a+ib} \end{bmatrix} \end{aligned}$$

Here we are using the notation that a bar over a complex number indicates the complex conjugate of the number. If we premultiply the matrix of line (16.5) with a  $2n$ -square matrix composed of  $n$  submatrices  $A^{-1}$  in diagonal position, and postmultiply with a  $2n$ -square matrix composed of  $n$  submatrices  $A$  in diagonal position, the determinant of the result equals the initial determinant value, but now has the form

$$\det \begin{bmatrix} \frac{\partial f_1}{\partial z_1} & 0 & \dots & \dots & \frac{\partial f_1}{\partial z_n} & 0 \\ 0 & \frac{\partial f_1}{\partial z_1} & \dots & \dots & 0 & \frac{\partial f_1}{\partial z_n} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ \frac{\partial f_n}{\partial z_1} & 0 & \dots & \dots & \frac{\partial f_n}{\partial z_n} & 0 \\ 0 & \frac{\partial f_n}{\partial z_1} & \dots & \dots & 0 & \frac{\partial f_n}{\partial z_n} \end{bmatrix} \quad (16.6)$$

By performing  $n - 1$  column exchanges on this determinant to move the even numbered columns to the right side, followed by the same number of row exchanges to move the even numbered rows to the bottom, we can make the determinant take the form

$$\det \begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \dots & \frac{\partial f_1}{\partial z_n} & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial z_1} & \dots & \frac{\partial f_n}{\partial z_n} & 0 & \dots & 0 \\ 0 & \dots & 0 & \frac{\partial f_1}{\partial z_1} & \dots & \frac{\partial f_1}{\partial z_n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \frac{\partial f_n}{\partial z_1} & \dots & \frac{\partial f_n}{\partial z_n} \end{bmatrix}$$

Thus the real Jacobian equals the product of the complex Jacobian times its own conjugate, or the square of the absolute value of the complex Jacobian, and so the real Jacobian is never negative.

## Software Exercises N

These exercises are with the demo programs `c_calc`, `c_fun`, `c_deriv`, `c_integ`, `c_roots` and `c_zeros`. All these programs, with the exception of `c_calc`, create both a log file and a print file.

1. Call up `c_calc` and evaluate to 10 decimal places  $\log(c)$ , with  $c$  taken at various points in the complex plane. Note that an error is indicated whenever  $c$  lies on the negative real axis.
2. Call up `c_fun` and evaluate  $e^z$ , starting at  $z = 0$  and using a  $z$  increment of  $\pi i$ . Choose 10 increments, 5 fixed-point decimal places for the  $z$  display, and 10 fixed-point decimal places for the  $f(z)$  display. To see the change in display with floating-point decimal places, edit the log file to change the sign of both decimal specifications, and then call up `c_fun c_fun`.
3. Call up `c_deriv` and find to 5 fixed-point decimal places the partial derivatives of  $\sin(z+w)$  of order 2 or less at the  $(z, w)$  point  $(0, 0)$ . You can edit the log file to choose another  $(z, w)$  point, and then call `c_deriv c_deriv` to obtain the new display.
4. Call up `c_integ` and find to 10 decimal places the integral of  $e^z$  along a straight line from  $a = 0$  to  $b = 2\pi i$ . The function  $e^z$  has a period of  $2\pi i$ , and the integral obtained is 0.
5. Call up `c_integ` and find to 10 decimal places the integral of  $1/z$  along a circle of radius 2 centered at the point  $i$ . This integral equals  $2\pi i$ .
6. Call up `c_roots`, and as a test, find to 10 fixed-point decimal places the roots of

$$(z - i)(z - 2i)(z - 3i)(z - 4i) = z^4 - 10iz^3 - 35z^2 + 50iz + 24$$

7. Call up `c_zeros` and find to 10 decimal places the zeros of  $(\sin z)^2$  in the search rectangle with opposite vertices  $-4 - i$  and  $3 + 2i$ . The zeros are not located by Newton's method but by the slow method of repeated bisection of an enclosing rectangle. If you edit the log file to change the function to  $\sin z$ , the same zeros, now simple, are found quickly by Newton's method.
8. Call up `c_zeros` and find to 10 decimal places the zeros of

$$f_1(z_1, z_2) = z_1^2 - z_2$$

$$f_2(z_1, z_2) = z_2^2 - z_1$$

Use the search rectangle with the opposite endpoints  $-10 - 10i$  and  $10 + 10i$  for both  $z_1$  and  $z_2$ . This problem is similar to the one given in Exercise 3 at the end of Chapter 12, except that in Exercise 3, there the variables range over real intervals instead of over complex rectangles as they do here. Note the four solutions here instead of the two obtained in Chapter 12.

## Notes and References

- A. The demo program `c_zeros` computes the crossing number to determine the number of zeros in a complex plane search rectangle. The papers by Collins and Krandick [1] and by Schaefer [2] discuss using the argument principle for this purpose.
- [1] Collins, G. E. and Krandick, W., *An efficient algorithm for infallible polynomial complex root isolation*, Proceedings of ISSAC, 1992.
- [2] Schaefer, M. J., Precise zero of analytic functions using interval arithmetic, *Interval Comput.* **4** (1993), 22–39.

## **The Precise Numerical Methods Program PNM**

The program **PNM** is for any PC using the Microsoft Windows XP operating system. **PNM** gives you access to a variety of demo programs that solve numerical problems precisely, and **PNM** also supplies the source code for these demo programs. The amount of hard disk space needed is small, just 10 megabytes.

### **Loading PNM and the demo programs onto your hard disk**

First load the CD into the CD reader of your PC. Next click on the **Start** taskbar and then on **Control Panel**. Finally click on **Add/Remove Programs**, and you will be guided through the installation process.

After the installation process is complete, the program **PNM** should be displayed as an accessible program when you click on **Programs**. Click on **PNM** and a form is displayed with the caption “Precise Numerical Methods”. Click on the **Load** menu, and then click on the command

Load exe and source files from CD

You will obtain a “Loading is Complete” message when all demo programs are loaded along with their source code. Chapter 1 has further details on how to run a demo program.

### **How to remove PNM and the demo programs from your hard disk**

Should you ever wish to remove **PNM** and the demo programs from your PC’s hard disk, reverse the loading process just described. With the **PNM** form in view, remove the demo programs by clicking on the **Load** menu, and then clicking on the command

Delete all PNM files

After you obtain the “Deletion is Complete” message, and click on it, the **PNM** form disappears. Next click on the **Start** taskbar and then on **Control Panel**. Finally click on **Add/Remove Programs**, which will allow you to remove the **PNM** program.

### **How to obtain the Windows command subsystem**

First click on **Run**, and then type `cmd` and hit the Enter key.

# Index



- Absolute value, 11
- Add Row Multiple, 101, 102–104
  
- Bairstow method, 86–90
- Basic variables, 143
- Boundary conditions, 193–4
  - linear, 193–4, 213–14
  - partial differential equations, 231–2
  - periodic, 194
  - separated, 194
- Bounding radius  $R$ , 85–6
  
- Calling up program, 2–3
- Companion matrices, 118–22
- Complex functions, elementary
  - see* Elementary complex functions
- Complex numbers, 97
- Computable number, 27
- Convergence, 75–7
- Correct to the last decimal place, 5
- Crossing number, 162, 166–70
  - computation, 171–5
  - properties, 170–1
- Crossing parity, 162, 165–6
  
- Danilevsky method, 122–7
  - error bounds, 127–34
- Decimal place problems, 29–32
- Decimal width, 15
  
- Degree 1 interval, 201–204
- Derivatives, 41
  - demo* program, 54
  - elementary functions, 41–8
  - general method of generating power series, 52–3
  - power series for elementary functions of several variables, 49–52
  - series evaluation, 48–9
- Determinant, 97, 102–104
- Diagonal elements, 187–8
- Discontinuous functions, 33–4
  
- Eigenvalue/eigenvector, 109, 113–18
  - companion/Vandermonde matrices, 118–22
- Danilevsky method, 122–7
  - demo* programs *eigen*, *c\_eigen*, *r\_eigen*, 135–6
  - error bounds for Danilevsky method, 127–34
  - finding solution to  $Ax = 0$  when  $\det A = 0$ , 110–13
  - nonsolvable problems, 112–13, 116–17
  - rational matrices, 134
  - solvable problems, 113, 114–15, 118, 134
  - theorems, 112, 115–16, 127–34
- Elementary complex functions, 235
  - computing line integrals in complex plane, 237–8



- Elementary complex functions (*Continued*)
  - computing roots of complex polynomial, 238–9
  - definition, 235–7
  - demo program *c\_deriv*, 237
  - finding zero of  $f(z)$ , 239–42
  - general zero problem, 242–5
- Elementary function, 37–9, 41–8
  - finding maximum/minimum, 181–3
  - finding solvable problem, 160–2, 178, 182–3
- Elementary region, 177
- Exchange Row, 101–104
- Exponent part, 6
- Extreme values, 181–3
  
- Feasible point, 142–3
- Floating point, 6–7, 9–10
  - scientific, 6, 10
  - variable precision, 10–11
- $f(x)$ :
  - extending solution method to general problem, 163–5
  - interval arithmetic, 72–3
  - Newton's method, 73–5
  - obtaining solvable problem, 69–72
  - order of convergence, 75–7
  
- General problem, 163–5
  - Newton's method, 175–6
- General zero problem, 242–5
- Global error carryover, 200
  - improved, 205–208
  
- Halfwidth, 16
- Hessian matrix, 185–7
  
- Initial value problem, 193
  - difficulties, 196–7
  - solving by power series, 198–201
- Integrals, 57–68
  - definite, 57–9
  - higher dimensional, 64–5
  - impint* program, 66–7
  - improper, 66–7
  - integ* program, 61–3
  - mulint* program, 64–5
- Interval, 11–13
  - formal, 59–61
  - $f(x)$  zero, 72–3
  - notation, 15–17
  
- Jacobian interval, 160, 161, 162, 165, 186, 243–5
  
- Line integrals, computing in complex plane, 237–8
- Linear boundary-value problem, 193–4, 213–14
- Linear differential equations, 197–8
- Linear equations, 97, 137–40
  - computation problems, 98–100
  - computing determinants, 102–104
  - equat*, *r\_quat*, *c\_equat* programs, 105–106
  - finding inverse of square matrix, 104–105
  - i\_equat* program, 156
  - method for solving, 100–102
  - notation, 97–8
  - solvable problem, 139–40
- Linear interval equations, 152–5
- Linear programming, 137
  - introduction to, 141–5
  - linpro* program, 155
  - $n$  linear interval equations in  $n$  unknown, 148–52
  - notation, 141–2
  - objective function, 141
  - simplex method, 142–7
  - solving linear equations, 152–5
- Linearly independent, 109
- Log files, 3–5
  
- Mantissa, 6
- Mean Value Theorem, 160–1
- Midpoint, 16
- Moore, Ramon, 11
- Multiple, 69
  
- $n$  linear interval equations in  $n$  unknown:
  - solvable problem, 148–9
  - theorem, 149–52
- Newton's method, 73–5, 175–6
- Nonbasic variables, 143
- Nonsolvable problem, 26, 32–3, 70–1, 80
  - decimal place, 29–32
  - eigenvalue/eigenvector, 116–17
  - linear equations, 98
  - partial differential equations, 219–20
  - repercussions, 27–8

- Notation, 15–17
  - linear equations, 97–8
- Null point, 143–4
- Open-source software, 1–2
- Optimization problems:
  - demo program *extrema*, 188
  - finding extreme values, 181–3
  - finding where function's gradient is zero, 184–8
- Ordinary differential equation of order  $n$ , 191–3
- Ordinary differential equation of the first order, 191–3
- Ordinary differential equations (ODEs), 217
  - difficulties with initial value problem, 196–7
  - initial value problems, 219–20
  - power series method, 220–3
  - two standard problems, 193–6
- Partial differential equations (PDEs):
  - defect of first method, 228–9
  - first solution method, 223–7
  - higher dimensional spaces, 230–1
  - initial value problems, 219–20
  - revised method with comparison computation, 229–30
  - satisfying boundary conditions, 231–2
  - simple problem as example, 227–8
  - terminology, 217–19
- Polynomials, 79–84
  - Bairstow method, 86–90
  - bound for the roots of, 85–6
  - bounding error of rational root approximations, 90–2
  - computing roots of complex polynomial, 238–9
  - finding accurate roots for rational or real, 92–5
  - roots* program, 95
- Power series:
  - definite integrals, 198
  - elementary functions of several variables, 49–52
  - general method of generating, 52–3
  - initial value problem, 198–201
  - ordinary differential equations, 220–3
  - two-point boundary-value problem, 210–13
- Print file, 3–4
- Queue domain, 171–5
- Range, 13–14
  - computing standard functions, 17–18
  - practical, 15
- Rational arithmetic, 137–40
- Rational linear equations, 140–1
- Rational numbers, 97
- Rational operations, 18–20
- Rational root approximation, 90–2
- Ray test, 171–3
- Real elementary functions, 159
- Real numbers, 97
- Real-valued functions, 37–40
- Region of interest, 159
- Remainder formula, 63–4
- Root, 69
- Roots of complex polynomial, 238–9
- Saddle point, 187–8
- Series evaluation, 48–9
- Simple zero, 69
- Simple zero gradient point, 185, 187–8
- Simplex method, 142–5
  - making process foolproof, 145–7
- Software exercises:
  - c\_calc*, 245–6
  - c\_deriv*, 245–6
  - c\_fun*, 245–6
  - c\_integ*, 245–6
  - c\_roots*, 245–6
  - c\_zeros*, 245–6
  - calc* program, 20–3
  - demo* program, 95–6
  - deriv* program, 54
  - difsys*, *difbnd* programs, 214–16
  - eigen*, *c\_eigen*, *r\_eigen* programs, 134
  - equat*, *r\_quat*, *c\_equat* programs, 106
  - fun* program, 39–40
  - integ*, *mulint*, *impint* programs, 67–8
  - linpro*, *I\_equat* programs, 156–7
  - maxmin*, *extrema* programs, 188–9
  - pde* program, 232–3
  - zeros* program, 77–8, 178–80

- Solvable problem, 26–7, 28–9  
  decimal place, 29, 30–2  
  eigenvalue/eigenvector, 113, 114–15, 118  
  elementary complex function, 240–2, 243  
  function's extreme values, 182–3, 185–8  
  handled by *calc*, 32  
  linear equations, 98–100  
  linear programming, 139–40, 148–9  
  linear two-point boundary-value, 214  
  obtaining, 69–72  
  ordinary differential equations, 197, 198  
  partial differential equations, 220  
  polynomials, 79–84  
  several functions are zero, 162, 178  
  two-point boundary-value, 209–10, 211–13
- Square matrix, 97, 104–105
- Standard functions, 17–18
- Tilde-box, 162
- Tilde notation, 5–7
- Topological degree, 166–70
- Turing, Alan, 27
- Two-point boundary-value problem, 193  
  power series, 210–13  
  solvable, 208–10
- Upper triangular matrix, 97
- Vandermonde matrices, 118–22
- Zero, 25–6  
  elementary complex function  $f(z)$ , 239–42  
   $f(x)$ , 69–77  
  finding several functions, 159–78  
  function gradient, 184–8  
  searching for more general region, 176–8