OLIVER ABERTH

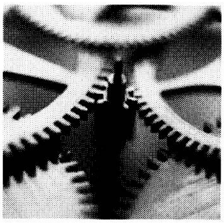
Introduction to Precise Numerical Methods

INCLUDES CD-ROM

2ND EDITION

TP301.6
A147
E.2

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

AP

Academic Press is an imprint of Elsevier

Acquisitions Editor	Tom Singer
Project Manager	Jay Donahue
Marketing Manager	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. ♾

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

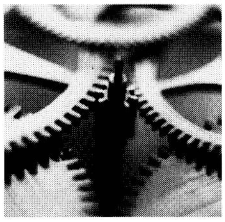
2007000712

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

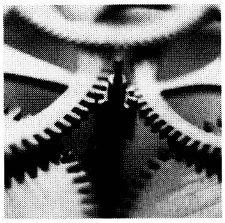
ISBN 13: 978-0-12-373859-2
ISBN 10: 0-12-373859-8

Printed in the United States of America
07 08 09 10 9 8 7 6 5 4 3 2 1

Introduction to Precise Numerical Methods



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.

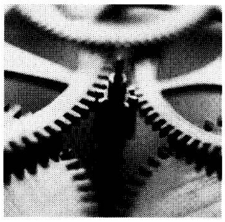
Besides the availability of powerful computers, two other advances permit the easy generation of accurate numerical answers. One is the development of efficient methods for accurately bounding computation errors, stemming from Ramon Moore's invention of interval arithmetic in 1962. The other is the development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



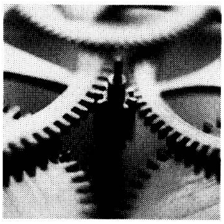
In the writing of this book and the creation of the accompanying software, I have had help from many sources. Two people have made fundamental contributions. Mark J. Schaefer, formerly of Tübingen University, helped write some of the computation programs. His brilliant programming skills were much needed, 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 develope the `pde` program, described in Chapter 15.

Oliver Aberth

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

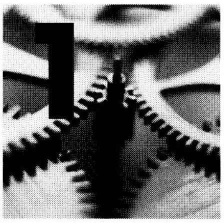
16 Numerical Methods with Complex Functions 235

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

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~
```

The tilde (˜) 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

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

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:

$$111.234$$