SECOND EDITION

FUNDA*M*ENTALS OF

Engineering Numerical Analysis

Parviz Moin

**CAMBRIDGE www.cambridge.org/97805**21884327

This page intentionally left blank

**FUNDAMENTALS OF ENGINEERING NUMERICAL ANALYSIS SECOND EDITION**

Since the original publication of this book, available computer power has increased greatly. Today, scientific computing is playing an ever more prominent role as a tool in scientific discovery and engineering analysis. In this second edition, the key addition is an introduction to the finite element method. This is a widely used technique for solving partial differential equations (PDEs) in complex domains. This text introduces numerical methods and shows how to develop, analyze, and use them. Complete MATLAB programs for all the worked examples are now available at www.cambridge.org/Moin, and more than 30 exercises have been added. This thorough and practical book is intended as a first course in numerical analysis, primarily for new graduate students in engineering and physical science. Along with mastering the fundamentals of numerical methods, students will learn to write their own computer programs using standard numerical methods.

Parviz Moin is the Franklin P. and Caroline M. Johnson Professor of Mechanical Engineering at Stanford University. He is the founder of the Center for Turbu- lence Research and the Stanford Institute for Computational and Mathematical Engineering. He pioneered the use of high-fidelity numerical simulations and mas- sively parallel computers for the study of turbulence physics. Professor Moin is a Fellow of the American Physical Society, American Institute of Aeronautics and Astronautics, and the American Academy of Arts and Sciences. He is a Member of the National Academy of Engineering.

**FUNDAMENTALS OF ENGINEERING NUMERICAL ANALYSIS SECOND EDITION**

PARVIZ MOIN Stanford University

CAMBRIDGE UNIVERSITY PRESS Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, S ̃ao Paulo, Delhi, Dubai, Tokyo, Mexico City

Cambridge University Press 32 Avenue of the Americas, New York, NY 10013-2473, USA

www.cambridge.org Information on this title: www.cambridge.org/9780521711234

C Parviz Moin 2010

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2010

Printed in the United States of America

*A catalog record for this publication is available from the British Library.*

*Library of Congress Cataloging in Publication data*

Moin, Parviz. Fundamentals of engineering numerical analysis / Parviz Moin. – 2nd ed. p. cm. Includes bibliographical references and index. ISBN 978-0-521-88432-7 (hardback) 1. Engineering mathematics. 2. Numerical analysis. I. Title. II. Title: Engineering numerical analysis. TA335.M65 2010 620.001 518–dc22 2010009012

ISBN 978-0-521-88432-7 Hardback ISBN 978-0-521-71123-4 Paperback

Additional resources for this publication at www.cambridge.org/Moin

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party Internet Web sites referred to in this publication and does not guarantee that any content on such Web sites is, or will remain, accurate or appropriate.

Contents

*Preface to the Second Edition page* ix

*Preface to the First Edition* xi

**1 INTERPOLATION** 1 1.1 Lagrange Polynomial Interpolation 1 1.2 Cubic Spline Interpolation 4 *Exercises* 8 *Further Reading* 12

**2 NUMERICAL DIFFERENTIATION – FINITE DIFFERENCES** 13 2.1 Construction of Difference Formulas Using Taylor Series 13 2.2 A General Technique for Construction of Finite Difference

Schemes 15 2.3 An Alternative Measure for the Accuracy of Finite

Differences 17 2.4 Pad ́e Approximations 20 2.5 Non-Uniform Grids 23 *Exercises* 25 *Further Reading* 29

**3 NUMERICAL INTEGRATION** 30 3.1 Trapezoidal and Simpson’s Rules 30 3.2 Error Analysis 31 3.3 Trapezoidal Rule with End-Correction 34 3.4 Romberg Integration and Richardson Extrapolation 35 3.5 Adaptive Quadrature 37 3.6 Gauss Quadrature 40 *Exercises* 44 *Further Reading* 47

v

vi CONTENTS

**4 NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS** 48 4.1 Initial Value Problems 48 4.2 Numerical Stability 50 4.3 Stability Analysis for the Euler Method 52 4.4 Implicit or Backward Euler 55 4.5 Numerical Accuracy Revisited 56 4.6 Trapezoidal Method 58 4.7 Linearization for Implicit Methods 62 4.8 Runge–Kutta Methods 64 4.9 Multi-Step Methods 70 4.10 System of First-Order Ordinary Differential Equations 74 4.11 Boundary Value Problems 78 4.11.1 Shooting Method 79 4.11.2 Direct Methods 82 *Exercises* 84 *Further Reading* 100

**5 NUMERICAL SOLUTION OF PARTIAL DIFFERENTIAL EQUATIONS** 101 5.1 Semi-Discretization 102 5.2 von Neumann Stability Analysis 109 5.3 Modified Wavenumber Analysis 111 5.4 Implicit Time Advancement 116 5.5 Accuracy via Modified Equation 119 5.6 Du Fort–Frankel Method: An Inconsistent Scheme 121 5.7 Multi-Dimensions 124 5.8 Implicit Methods in Higher Dimensions 126 5.9 Approximate Factorization 128 5.9.1 Stability of the Factored Scheme 133 5.9.2 Alternating Direction Implicit Methods 134 5.9.3 Mixed and Fractional Step Methods 136 5.10 Elliptic Partial Differential Equations 137 5.10.1 Iterative Solution Methods 140 5.10.2 The Point Jacobi Method 141 5.10.3 Gauss–Seidel Method 143 5.10.4 Successive Over Relaxation Scheme 144 5.10.5 Multigrid Acceleration 147 *Exercises* 154 *Further Reading* 166

**6 DISCRETE TRANSFORM METHODS** 167 6.1 Fourier Series 167 6.1.1 Discrete Fourier Series 168 6.1.2 Fast Fourier Transform 169 6.1.3 Fourier Transform of a Real Function 170 6.1.4 Discrete Fourier Series in Higher Dimensions 172

CONTENTS vii

6.1.5 Discrete Fourier Transform of a Product of Two

Functions 173 6.1.6 Discrete Sine and Cosine Transforms 175 6.2 Applications of Discrete Fourier Series 176 6.2.1 Direct Solution of Finite Differenced Elliptic Equations 176 6.2.2 Differentiation of a Periodic Function Using Fourier

Spectral Method 180 6.2.3 Numerical Solution of Linear, Constant Coefficient

Differential Equations with Periodic Boundary Conditions 182 6.3 Matrix Operator for Fourier Spectral Numerical

Differentiation 185 6.4 Discrete Chebyshev Transform and Applications 188 6.4.1 Numerical Differentiation Using Chebyshev Polynomials 192 6.4.2 Quadrature Using Chebyshev Polynomials 195 6.4.3 Matrix Form of Chebyshev Collocation Derivative 196 6.5 Method of Weighted Residuals 200 6.6 The Finite Element Method 201

6.6.1 Application of the Finite Element Method to a Boundary

Value Problem 202 6.6.2 Comparison with Finite Difference Method 207 6.6.3 Comparison with a Pad ́e Scheme 209 6.6.4 A Time-Dependent Problem 210 6.7 Application to Complex Domains 213 6.7.1 Constructing the Basis Functions 215 *Exercises* 221 *Further Reading* 226

**A A REVIEW OF LINEAR ALGEBRA** 227 A.1 Vectors, Matrices and Elementary Operations 227 A.2 System of Linear Algebraic Equations 230 A.2.1 Effects of Round-off Error 230 A.3 Operations Counts 231 A.4 Eigenvalues and Eigenvectors 232

*Index* 235

*To Linda*

Preface to the Second Edition

Since the original publication of this book ten years ago, the available computer power has increased by more than 2 orders of magnitude due to massive par- allelism of computer processors and heterogeneous computer clusters. Today, scientific computing is playing an ever more prominent role as a tool in scientific discovery and engineering analysis.

In the second edition an introduction to the finite element method has been added. The finite element method is a widely used technique for solving partial differential equations (PDEs) in complex domains. As in the first edition, numerical solution of PDEs is treated in Chapter 5, and the development there is based on finite differences for spatial derivatives. This development is followed in Chapter 6 by an introduction to more advanced transform methods for solving PDEs: spectral methods and, now, the finite element method. These methods are compared to the finite difference methods in several places throughout Chapter 6.

Hopefully, most of the errors that remained in the 2007 reprint of the book have now been corrected. Several exercises have also been added to all the chapters. In addition, complete MATLAB programs used for all the worked examples are available at www.cambridge.org/Moin. Students should find this new feature helpful in attempting the exercises, as similar computer programs are used in many of them. Working out the exercises is critical to learning numerical analysis, especially using this book. The intention for including this feature is for students to spend less time writing and debugging computer programs and more time digesting the underlying concepts.

I thank all the students and teaching assistants who have provided valuable feedback to me on the teaching of numerical analysis and the contents of this book. In particular, I am grateful to Dr. Ali Mani who took a special interest in this book and made significant technical contributions to the new edition. Special thanks are due to Nick Henderson for compiling the examples programs and Drs. Erich Elsen and Lawrence Cheung for their due diligence and help in

ix

x PREFACE TO THE SECOND EDITION

the preparation of this edition. Prof. Jon Freund suggested the addition of the finite element section and gave me a draft of his notes on the subject to get me started.

Parviz Moin *Stanford, California March 2010*

Preface to the First Edition

With the advent of faster computers, numerical simulation of physical phenom- ena is becoming more practical and more common. Computational prototyping is becoming a significant part of the design process for engineering systems. With ever-increasing computer performance the outlook is even brighter, and computer simulations are expected to replace expensive physical testing of de- sign prototypes.

This book is an outgrowth of my lecture notes for a course in computational mathematics taught to first-year engineering graduate students at Stanford. The course is the third in a sequence of three quarter-courses in computational mathematics. The students are expected to have completed the first two courses in the sequence: numerical linear algebra and elementary partial differential equations. Although familiarity with linear algebra in some depth is essential, mastery of the analytical tools for the solution of partial differential equations (PDEs) is not; only familiarity with PDEs as governing equations for physical systems is desirable. There is a long tradition at Stanford of emphasizing that engineering students learn numerical analysis (as opposed to learning to run canned computer codes). I believe it is important for students to be educated about the fundamentals of numerical methods. My first lesson in numerics in- cludes a warning to the students not to believe, at first glance, the numerical output spewed out from a computer. They should know what factors affect ac- curacy, stability, and convergence and be able to ask tough questions before accepting the numerical output. In other words, the user of numerical methods should not leave all the “thinking” to the computer program and the person who wrote it. It is also important for computational physicists and engineers to have first-hand experience solving real problems with the computer. They should experience both the power of numerical methods for solving non-trivial problems as well as the frustration of using inadequate methods. Frustrating experiences with a numerical method almost always send a competent numer- ical analyst to the drawing board and force him or her to ask good questions

xi

xii PREFACE TO THE FIRST EDITION

about the choice and parameters of the method, which should have been asked before going to the computer in the first place. The exercises at the end of each chapter are intended to give these important experiences with numerical methods.

Along with mastering the fundamentals of numerical methods, the students are expected to write their own programs to solve problems using standard numerical methods. They are also encouraged to use standard (commercial) software whenever possible. There are several software libraries with well- documented programs for basic computational work. Recently, I have used the *Numerical Recipes* by Press et al. (Cambridge) as an optional supplement to my lectures. *Numerical Recipes* is based on a large software library that is well documented and available on computer disks. Some of the examples in this book refer to specific programs in *Numerical Recipes.*

Students should also have a simple (*x*, *y*) plotting package to display their numerical results. Some students prefer to use MATLAB’s plotting software, some use the plotting capability included with a spreadsheet package, and oth- ers use more sophisticated commercial plotting packages. Standard well-written numerical analysis programs are generally available for almost everything cov- ered in the first four chapters, but this is not the case for partial differential equations, discussed in Chapter 5. The main technical reason for this is the large variety of partial differential equations, which requires essentially tailor- made programs for each application.

No attempt has been made to provide complete coverage of the topics that I have chosen to include in this book. This is not meant to be a reference book; rather it contains the material for a first course in numerical analysis for future practitioners. Most of the material is what I have found useful in my career as a computational physicist/engineer. The coverage is succinct, and it is ex- pected that all the material will be covered sequentially. The book is intended for first-year graduate students in science and engineering or seniors with good post-calculus mathematics backgrounds. The first five chapters can be cov- ered in a one-quarter course, and Chapter 6 can be included in a one-semester course.

Discrete data and numerical interpolation are introduced in Chapter 1, which exposes the reader to the dangers of high-order polynomial interpolation. Cu- bic splines are offered as a good working algorithm for interpolation. Chapter 2 (finite differences) and Chapter 3 (numerical integration) are the foundations of discrete calculus. Here, I emphasize systematic procedures for construct- ing finite difference schemes, including high-order Pad ́e approximations. We also examine alternative, and often more informative, measures of numeri- cal accuracy. In addition to introducing the standard numerical integration techniques and their error analysis, we show in Chapter 3 how knowledge of the form of numerical errors can be used to construct more accurate numeri- cal results (Richardson extrapolation) and to construct adaptive schemes that

PREFACE TO THE FIRST EDITION xiii

obtain the solution to the accuracy specified by the user. Usually, at this point in my lectures, I seize the opportunity, offered by these examples, to stress the value of a detailed knowledge of numerical error and its pay-offs even for the most application-oriented students. Knowledge is quickly transferred to power in constructing novel numerical methods.

Chapter 4 is on numerical solution of ordinary differential equations (ODEs) – the heart of this first course in numerical analysis. A number of new concepts such as stability and stiffness are introduced. The reader begins to experience new tools in the arsenal for solving relatively complex problems that would have been impossible to do analytically. Because so many inter- esting applications are cast in ordinary differential equations, this chapter is particularly interesting for engineers. Different classes of numerical methods are introduced and analyzed even though there are several well-known powerful numerical ODE solver packages available to solve any practical ODE without having to know their inner workings. The reason for this extensive coverage of a virtually solved problem is that the same algorithms are used for solution of partial differential equations when canned programs for general PDEs are not available and the user is forced to write his or her own programs. Thus, it is essential to learn about the properties of numerical methods for ODEs in order to develop good programs for PDEs.

Chapter 5 discusses numerical solution of partial differential equations and relies heavily on the analysis of initial value problems introduced for ODEs. In fact by using the modified wavenumber analysis, we can cast into ODEs the discretized initial value problems in PDEs, and the knowledge of ODE properties becomes very useful and no longer of just academic value. Once again the knowledge of numerical errors is used to solve a difficult problem of dealing with large matrices in multi-dimensional PDEs by the approximate factorization technique. Dealing with large matrices is also a focus of numerical techniques for elliptic partial differential equations, which are dealt with by introducing the foundations of iterative solvers.

Demand for high accuracy is increasing as computational engineering ma- tures. Today’s engineers and physicists are less interested in qualitative features of numerical solutions and more concerned with numerical accuracy. A branch of numerical analysis deals with spectral methods, which offer highly accu- rate numerical methods for solution of partial differential equations. Chapter 6 covers aspects of Fourier analysis and introduces transform methods for partial differential equations.

My early work in numerical analysis was influenced greatly by discus- sions with Joel Ferziger and subsequently by the works of Harvard Lomax at NASA–Ames. Thanks are due to all my teaching assistants who helped me develop the course upon which this book is based; in particular, I thank Jon Freund and Arthur Kravchenko who provided valuable assistance in prepara- tion of this book. I am especially grateful to Albert Honein for his substantial

xiv PREFACE TO THE FIRST EDITION

help in preparing this book in its final form and for his many contributions as my teaching assistant in several courses in computational mathematics at Stanford.

Parviz Moin *Stanford, California July 2000*

1 Interpolation

Often we want to fit a smooth curve through a set of data points. Applications might be differentiation or integration or simply estimating the value of the function between two adjacent data points. With interpolation we actually pass a curve *through* the data. If data are from a crude experiment characterized by some uncertainty, it is best to use the method of least squares, which does not require the approximating function to pass through all the data points.

**1.1 Lagrange Polynomial Interpolation**

Suppose we have a set of *n* + 1 (not necessarily equally spaced) data (*xi*, *yi*). We can construct a polynomial of degree *n* that passes through the data:

*P*(*x*) = *a*0 + *a*1*x* + *a*2*x*2 +···+ *anxn.*

The *n* + 1 coefficients of *P* are determined by forcing *P* to pass through the data. This leads to *n* + 1 equations in the *n* + 1 unknowns, *a*0, *a*1,..., *an*:

*yi* = *P*(*xi*) = *a*0 + *a*1*xi* + *a*2*xi* 2+···+ *anxi ni* = 0*,* 1*,* 2*,...,n.*

This procedure for finding the coefficients of the polynomial is not very attractive. It involves solving a system of algebraic equations that is generally ill- conditioned (see Appendix) for large*n*. In practice we will define the polynomial in an explicit way (as opposed to solving a system of equations). Consider the following polynomial of degree *n* associated with each point *xj*:

*L j*(*x*) = *αj*(*x* − *x*0)(*x* − *x*1)···(*x* − *x j*−1)(*x* − *x j*+1)···(*x* − *xn*)*,*

where *αj* is a constant to be determined. In the product notation, *Lj* is written as follows

*L j*(*x*) = *αj*

∏*ni*=0 *i* = *j*

(*x* − *xi*)*.*

1

2 INTERPOLATION

If *x* is equal to any of the data points except *xj*, then *Lj*(*xi*) = 0 for *i* = *j*. For *x* = *xj*,

*L j*(*x j*) = *αj*

∏*ni*=0 *i* = *j*

(*x j* − *xi*)*.*

We now define *α j* to be

*α j* =

⎡⎢⎢⎣

(*x* ⎤−1 ∏*nj* − *xi*)⎥⎥⎦*. i*=0 *i* = *j*

Then, *L j* will have the following important property:

*L j*(*xi*) =

{0 *xi* = *x j*

1 *xi* = *x j.* (1.1)

Next we form a linear combination of these polynomials with the data as weights:

*P*(*x*) =

∑*nj*=0

*yjL j*(*x*)*.* (1.2)

This is a polynomial of degree*n* because it is a linear combination of polynomi- als of degree *n.* It is called a *Lagrange polynomial.* It is the desired interpolating polynomial because by construction, it passes through all the data points. For example, at *x* = *xi*

*P*(*xi*) = *y*0*L*0(*xi*) + *y*1*L*1(*xi*) +···+ *yiLi*(*xi*) +···+ *ynLn*(*xi*)*.*

Since *Li*(*xk*) is equal to zero except for *k* = *i*, and *Li*(*xi*) = 1,

*P*(*xi*) = *yi.*

Note that polynomial interpolation is unique. That is, there is only one poly- nomial of degree *n* that passes through a set of *n* + 1 points\*. The Lagrange polynomial is just a compact, numerically better behaved way of expressing the polynomial whose coefficients could have also been obtained from solving a system of algebraic equations.

For a large set of data points (say greater than 10), polynomial interpolation for uniformly spaced data can be very dangerous. Although the polynomial is fixed (tied down) at the data points, it can wander wildly between them, which can lead to large errors for derivatives or interpolated values.

∗ The uniqueness argument goes like this: suppose there are two polynomials of degree *n*, *Z*1 and *Z*2 that pass through the same data points, *x*0, *x*1,..., *xn*. Let *Z* = *Z*1 – *Z*2. *Z* is a polynomial of degree *n* with *n* + 1 zeros, *x*0, *x*1,..., *xn*, which is impossible unless *Z* is identically zero.

1.1 LAGRANGE POLYNOMIAL INTERPOLATION 3

**EXAMPLE 1.1 Lagrange Interpolation**

Consider the following data, which are obtained from a smooth function also known as Runge’s function, *y* = (1 + 25*x*2)−1:

*xi* −1.00 −0.80 −0.60 −0.40 −0.20 0.00 0.20 0.40 0.60 0.80 1.00 *yi* 0.038 0.058 0.100 0.200 0.500 1.00 0.500 0.200 0.100 0.058 0.038

We wish to fit a smooth curve through the data using the Lagrange polyno- mial interpolation, for which the value at any point *x* is simply

*P* (*x*) =

∑*nj*=0

*y j*

∏*ni*=0 *i* =*j*

*x* − *xi x j* − *xi .*

For example at the point *(x* = 0.7), the interpolated value is

*P* (*.*7) = 0*.*038 (−1*.*0 (0*.*7 + + 0*.*8)(−1*.*0 0*.*8)(0*.*7 + + 0*.*6) 0*.*6) ···(0*.*7 − 0*.*8)(0*.*7 − 1*.*0)

···(−1*.*0 − 0*.*8)(−1*.*0 − 1*.*0) +0*.*058 (−0*.*8 (0*.*7 + + 1*.*0)(−0*.*8 1*.*0)(0*.*7 + + 0*.*6)···(0*.*7 − 0*.*8)(0*.*7 − 1*.*0)

0*.*6)···(−0*.*8 − 0*.*8)(−0*.*8 − 1*.*0) + ··· +0*.*038(0*.*7 (1*.*0 + + 1*.*0)(0*.*7 1*.*0)(1*.*0 + + 0*.*8)···(0*.*7 0*.*6)···(1*.*0 − − 0*.*6)(0*.*7 0*.*6)(1*.*0 − − 0*.*8)

0*.*8) = −0*.*226*.*

Evaluating the interpolating polynomial at a large number of intermediate points, we may plot the resulting polynomial curve passing through the data points (see Figure 1.1). It is clear that the Lagrange polynomial behaves very poorly between some of the data points, especially near the ends of the interval. The problem does not go away by simply having more data points in the interval and thereby tying down the function further. For example, if instead of eleven points we had twenty-one uniformly spaced data points in the same interval, the overshoots at the ends would have peaked at nearly 60 rather than at 1.9 as they did for eleven points. However, as shown in the following example, the problem can be somewhat alleviated if the data points are non-uniformly spaced with finer spacings near the ends of the interval.

**2.0 Lagrange Polynomial 1.5**

**Expected Behavior Data Points**

**) x(fx 1.0**

**0.5**

**0**

**-0.5-1.0 -0.5 0 0.5 1.0 Figure 1.1** Lagrange polynomial interpolation of Runge’s function using eleven equally spaced data points.

4 INTERPOLATION

**EXAMPLE 1.2 Lagrange Interpolation With Non-equally Spaced Data**

Consider the following data which are again extracted from the Runge’s func- tion of Example 1.1. The same number of points are used as in Example 1.1, but the data points *xi* are now more finely spaced near the ends (at the expense of coarser resolution near the center).

*xi* −1.00 −0.95 −0.81 −0.59 −0.31 0.00 0.31 0.59 0.81 0.95 1.00 *yi* 0.038 0.042 0.058 0.104 0.295 1.00 0.295 0.104 0.058 0.042 0.038

The interpolation polynomial and the expected curve, which in this case (as in Example 1.1) is simply the Runge’s function, are plotted in Figure 1.2. It is apparent that the magnitudes of the overshoots at the ends of the inter- val have been reduced; however, the overall accuracy of the scheme is still unacceptable.

**1.2**

**1.0**

**Lagrange Interpolation Expected Behavior Data Points**

**) x(f0.8 0.6**

**0.4**

**0.2**

**0 -1.0 -0.5 0 x 0.5 1.0 Figure 1.2** Lagrange polynomial interpolation of Runge’s function using eleven non- equally spaced data points. The data toward the ends of the interval are more finely spaced.

The wandering problem can also be severely curtailed by*piecewise Lagrange* interpolation. Instead of fitting a single polynomial of degree *n* to all the data, one fits lower order polynomials to sections of it. This is used in many practical applications and is the basis for some numerical methods. The main problem with piecewise Lagrange interpolation is that it has discontinuous slopes at the boundaries of the segments, which causes difficulties when evaluating the derivatives at the data points. Interpolation with cubic splines circumvents this difficulty.

**1.2 Cubic Spline Interpolation**

Interpolation with cubic splines is essentially equivalent to passing a flexible plastic ruler through the data points. You can actually hammer a few nails partially into a board and pretend that they are a set of data points; the nails can then hold a plastic ruler that is bent to touch all the nails. Between the nails, the ruler acts as the interpolating function. From mechanics the equation governing

1.2 CUBIC SPLINE INTERPOLATION 5

gx

5

4) x("32100 1 2 3 4 5 **Figure 1.3** A schematic showing the linearity of *g* in between the data points. Also note that with such a construction, *g* is continuous at the data points.

the position of the curve *y*(*x*) traced by the ruler is

*Cy*(*iv*) = *G,*

where *C* depends on the material properties and *G* represents the applied force necessary to pass the spline through the data. The force is applied only at the data points; between the data points the force is zero. Therefore, the spline is piecewise cubic between the data. As will be shown below, the spline interpolant and its *first two derivatives are continuous at the data points.*

Let *gi*(*x*) be the cubic in the interval *xi* ≤ *x* ≤ *xi*+1 and let *g*(*x*) denote the collection of all the cubics for the entire range of *x.* Since *g* is piecewise cubic its second derivative, *g* , is piecewise linear. For the interval *xi* ≤ *x* ≤ *xi*+1, we can write the equation for the corresponding straight line as

*g i* (*x*) = *g* (*xi*) *xx i* − − *xi*+1

*xi*+1 + *g* (*xi*+1) *xi*+1 *x* − *xi*

− *xi .* (1.3) Note that by construction, in (1.3) we have enforced the continuity of the second derivative at the data points. That is, as shown in Figure 1.3, straight lines from the adjoining intervals meet at the data points.

Integrating (1.3) twice we obtain

*g i*(*x*) = *g* (*xi*)

*xi* − *xi*+1

(*x* − *xi*+1)2

2 + *g* (*xi*+1) *xi*+1 − *xi*

(*x* − *xi*)2

2 + *C*1 (1.4) and*gi*(*x*) = *g* (*xi*)

*xi* − *xi*+1

(*x* − *xi*+1)3

6 + *g* (*xi*+1) *xi*+1 − *xi*

(*x* − 6 *xi*)3

+ *C*1*x* + *C*2*.* (1.5)

The undetermined constants *C*1 and *C*2 are obtained by matching the functional values at the end points:

*gi*(*xi*) = *f* (*xi*) ≡ *yi gi*(*xi*+1) = *f* (*xi*+1) ≡ *yi*+1*,*

which give two equations for the two unknowns, *C*1 and *C*2. Substituting for *C*1

6 INTERPOLATION

and *C*2 in (1.5) leads to the spline equation used for interpolation:

*gi*(*x*) = *g* 6

(*xi*)

[(*xi*+1 − *x*)3

*i* ] − *i*(*xi*+1 − *x*)+ *g* (*x*6

*i*+1)

[(*x* − *xi*)3

*i* ] − *i*(*x* − *xi*)+ *f* (*xi*)*xi*+1 − *x*

*i* + *f* (*xi*+1)*x* − *xi*

*i ,* (1.6) where *xi* ≤ *x* ≤ *xi*+1 and *i* = *xi*+1 − *xi*. In (1.6) *g* (*xi*) and *g* (*xi*+1) are still unknowns. To obtain *g* (*xi*), we use the remaining matching condition, which is the continuity of the first derivatives:

*g i*(*xi*) = *g i*−1(*xi*)*.* The desired system of equations for *g* (*xi*) is then obtained by differentiating *gi*(*x*) and *gi*−1(*x*) from (1.6) and equating the two derivatives at *x* = *xi*. This leads to

6 *i*−1

*g* (*xi*−1) + *i*−1 3 + *i*

*g* (*xi*) + 6 *ig* (*xi*+1) = *f* (*xi*+1) − *f* (*xi*)

*i* − *f* (*xi*) − *f* (*xi*−1)

*i*−1 *i* = 1*,*2*,*3*,..., N* − 1*.* (1.7) These are *N –* 1 equations for the *N* + 1 unknowns *g* (*x*0)*,g* (*x*1)*,..., g* (*xN*). The equations are in tridiagonal form and diagonally dominant, and therefore they can be solved very efficiently. The remaining equations are obtained from the prescription of some “end conditions.” Typical conditions are:

a) Free run-out (natural spline):

*g* (*x*0) = *g* (*xN*) = 0*.*

This is the most commonly used condition. It can be shown that with this condition, the spline is the smoothest interpolant in the sense that the integral of*g* 2 over the whole interval is smaller than any other function interpolating the data. b) Parabolic run-out:

*g* (*x*0) = *g* (*x*1) *g* (*xN*−1) = *g* (*xN*)*.*

In this case, the interpolating polynomials in the first and last intervals are parabolas rather than cubics (see Exercise 3). c) Combination of (a) and (b):*g* (*x*0) = *αg* (*x*1)

*g* (*xN*−1) = *βg* (*xN*)*,*

where *α* and *β* are constants chosen by the user.

1.2 CUBIC SPLINE INTERPOLATION 7

d) Periodic:

*g* (*x*0) = *g* (*xN*−1) *g* (*x*1) = *g* (*xN*)*.*

This condition is suitable for interpolating in one period of a known periodic signal.

The general procedure for spline interpolation is first to solve the system of equations (1.7) with the appropriate end conditions for *g* (*xi*). The result is then used in (1.6), providing the interpolating function *gi*(*x*) for the interval *xi* ≤ *x* ≤ *xi*+1. In general, spline interpolation is preferred over Lagrange polynomial interpolation; it is easy to implement and usually leads to smooth curves.

**EXAMPLE 1.3 Cubic Spline Interpolation**

We will now interpolate the data in Example 1.1 with a cubic spline. We solve the tridiagonal system derived in (1.7). Since the data are uniformly spaced, this equation takes a particularly simple form for *g* (*xi*):

16*g* (*xi*−1) + 23*g* (*xi*) + 16*g* (*xi*+1) = *yi*+1 − 2*yi* + *yi*−1

2 *i* = 1*,*2*,..., n*− 1*.*

For this example, we will use the free run-out condition *g* (*x*0) = *g* (*xn*) = 0*.* The cubic spline is evaluated at several *x* points using (1.6) and the *g* (*xi*) values obtained from the solution of this tridiagonal system. The subroutine spline in *Numerical Recipes* has been used in the calculation. The equiv- alent function in MATLAB is also called spline. The result is presented in Figure 1.4. Spline representation appears to be very smooth and is virtually indistinguishable from Runge’s function.**1.25**

**1.00 Cubic Spline Data Points**

**) x(f0.75 0.50 0.25 0 -1.0 -0.5 0 0.5 1.0 x**

**Figure 1.4** Cubic spline interpolation of Runge’s function using the equally spaced data of Example 1.1.

Clearly spline interpolation is much more accurate than Lagrange inter- polation. Of course, the computer program for spline is longer and a bit more complicated than that for Lagrange interpolation. However, once such programs are written for general use, then the time taken to develop the program, or the “human cost,” no longer enters into consideration.

8 INTERPOLATION

An interesting version of spline interpolation, called tension spline, can be used if the spline fit wiggles too much. The idea is to apply some tension or pull from both ends of the flexible ruler discussed at the beginning of this section. Mathematically, this also leads to a tridiagonal system of equations for *g i* , but the coefficients are more complicated. In the limit of very large tension, all the wiggles are removed, but the spline is reduced to a simple straight line interpolation (see Exercise 6).

**EXERCISES**

1. Write a computer program for Lagrange interpolation (you may want to use the *Numerical Recipes* subroutine polint or interp1 of MATLAB). Test your program by verifying that *P*(0*.*7) = −0*.*226 in Example 1.1. (a) Using the data of Example 1.1, find the interpolated value at *x* = 0.9. (b) Use Runge’s function to generate a table of 21 equally spaced data points. Interpolate these data using a Lagrange polynomial of order 20. Plot this polynomial and comment on the comparison between your result and the plot of Example 1.1. 2. Derive an expression for the derivative of a Lagrange polynomial of order *n* at

a point *x* between the data points. 3. Show that if parabolic run-out conditions are used for cubic spline interpolation, then the interpolating polynomials in the first and last intervals are indeed parabolas. 4. An operationally simpler spline is the so-called quadratic spline. Interpolation

is carried out by piecewise quadratics. (a) What are the suitable joint conditions for quadratic spline? (b) Show how the coefficients of the spline are obtained. What are suitable end

conditions? (c) Compare the required computational efforts for quadratic and cubic

splines. 5. Consider a set of *n* + 1 data points (*x*0*, f*0)*,...,*(*xn, fn*), equally spaced with *xi*+1 − *xi* = *h*. Discuss how cubic splines can be used to obtain a numerical approximation for the first derivative *f* at these data points. Give a detailed account of the required steps. You should obtain formulas for the numerical derivative at the data points *x*0,..., *xn* and explain how to calculate the terms in the formulas. 6. Tension splines can be used if the interpolating spline wiggles too much. In this case, the equation governing the position of the plastic ruler in between the data points is

*y*(*iv*) − *σ*2*y* = 0

where *σ* is the tension parameter. If we denote *gi*(*x*) as the interpolating tension spline in the interval *xi* ≤ *x* ≤ *xi*+1, then *g i* (*x*) − *σ*2*gi*(*x*) is a straight line in

EXERCISES 9

this interval, which can be written in the following convenient forms:

*g i* (*x*) − *σ*2*gi*(*x*) = [*g* (*xi*) − *σ* 2*f* (*xi*)] *x* − *xi*+1 *xi* − *xi*+1 + [*g* (*xi*+1) − *σ*2*f* (*xi*+1)] *x* − *xi*

*xi*+1 − *xi .*

(a) Verify that for *σ* = 0, the cubic spline is recovered, and *σ* → ∞ leads to

linear interpolation. (b) Derive the equation for tension spline interpolation, i.e., the expression

for *gi*(*x*). 7. The tuition for 12 units at *St. Anford* University has been increasing from

1998 to 2008 as shown in the table below:

**Year Tuition per year** 1998 $21,300 1999 $23,057 2000 $24,441 2001 $25,917 2002 $27,204 2003 $28,564 2004 $29,847 2005 $31,200 2006 $32,994 2007 $34,800 2008 $36,030

(a) Plot the given data points and intuitively interpolate (draw) a smooth curve

through them. (b) Interpolate the data with the Lagrange polynomial. Plot the polynomial and the data points. Use the polynomial to predict the tuition in 2010. This is an extrapolation problem; discuss the utility of Lagrange polynomials for extrapolation. (c) Repeat (b) with a cubic spline interpolation and compare your results. 8. The concentration of a certain toxin in a system of lakes downwind of an industrial area has been monitored very accurately at intervals from 1993 to 2007 as shown in the table below. It is believed that the concentration has varied smoothly between these data points.

**Year Toxin Concentration** 1993 12.0 1995 12.7 1997 13.0 1999 15.2 2001 18.2 2003 19.8 2005 24.1 2007 28.1 2009 ???

10 INTERPOLATION

(a) Interpolate the data with the Lagrange polynomial. Plot the polynomial and the data points. Use the polynomial to predict the condition of the lakes in 2009. Discuss this prediction. (b) Interpolation may also be used to fill “holes” in the data. Say the data from 1997 and 1999 disappeared. Predict these values using the Lagrange poly- nomial fitted through the other known data points. (c) Repeat (b) with a cubic spline interpolation. Compare and discuss your

results. 9. Consider a piecewise Lagrange polynomial that interpolates between three points at a time. Let a typical set of consecutive three points be *xi*−1*, xi*, and *xi*+1. Derive differentiation formulas for the first and second derivatives at *xi*. Simplify these expressions for uniformly spaced data with = *xi*+1 − *xi*. You have just derived finite difference formulas for discrete data, which are discussed in the next chapter. 10. Consider a function *f* defined on a set of *N* + 1 discrete points

*x*0 *< x*1 *<* ··· *< xN.*

We want to derive an (*N* + 1) × (*N* + 1) matrix, *D* (with elements *dij*), which when multiplied by the vector of the values of *f* on the grid results in the deriva- tive of *f* at the grid points. Consider the Lagrange polynomial interpolation of *f* in (1.2):

*P*(*x*) =

∑*Nj*=0

*yj L j*(*x*)*.*

We can differentiate this expression to obtain *P* . We seek a matrix *D* such that

*D****f*** = ***P*** *N* where, points. Note ***P*** *N* is that a vector the derivative whose elements approximation are the derivative given by of *D****f*** *P*(*x*) at the data is exact for all polynomials of degree *N* or less. We define *D* such that it gives the exact derivatives for all such polynomials at the *N* + 1 grid points. That is, we want

*D* } *Lk*(*x* {{ *j*) }

*δkj*

= *L k*(*x j*) *j, k* = 0*,* 1*,* 2*,..., N*

where *δkj* is Kronecker delta which is equal to one for *k* = *j* and zero for *k* = *j*. Show that this implies that

*djk* = *ddx Lk*∣∣∣∣*x*=*x j ,* (1) where *djk* are the elements of *D.* Evaluate the right-hand side of (1) and show that

*djk* = *L k*(*x j*) = *αk*

∏*Nl* = *l*=0 *j,k*

(*x j* − *xl*) = *αk*

*αj*(*x j* − *xk*) for *j* = *k,* (2)

EXERCISES 11

and

*djj* = *L j*(*x j*) =

∑*Nl*=0 *l* =*j*

1 *x j* − *xl* for *j* = *k* (3)

where, *αj* is defined in Section 1.1. (HINT: Take the logarithm *Lk*(*x*)*.*) 11. In this problem, we want to develop the two-dimensional spline interpolation procedure, which has applications in many areas such as image processing, weather maps, and topography analysis.

Consider *f* (*x, y*) defined on [0*,* 4] × [0*,*4] given at the following points:

*f* (0*,*0) = 0*.*0006 *f* (1*,*0) = 0*.*2904 *f* (2*,*0) = 0*.*5648 *f* (3*,* 0) = 0*.*2751 *f* (0*,*1) = 0*.*2499 *f* (1*,*1) = 1*.*7995 *f* (2*,*1) = 2*.*8357 *f* (3*,* 1) = 1*.*2861 *f* (0*,*2) = 0*.*4916 *f* (1*,*2) = 2*.*4900 *f* (2*,*2) = 3*.*8781 *f* (3*,* 2) = 1*.*8796 *f* (0*,*3) = 0*.*2423 *f* (1*,*3) = 0*.*9809 *f* (2*,*3) = 1*.*6072 *f* (3*,* 3) = 0*.*8686.

Furthermore, assume that *f* has periodic boundary conditions. In other words, the value of *f* and all of its derivatives are the same at (*x, y*) and (*x* + 4*k, y* + 4*l*) for all integer values of *k* and *l*. Let’s assume that we are interested in the values of the function in a subregion of the domain defined by 1 ≤ *x* ≤ 2 and 1 ≤ *y* ≤ 2 (the area shown in the figure). In the first step, we focus on interpolating *f* at a given point. For example, through the following steps we can obtain estimates for *f* (1*.*5*,*1*.*5). (a) Use a contour plot routine (such as Matlab’s contour) over the given data

and obtain a rough estimate for *f* (1*.*5*,*1*.*5). (b) Let *g*(*x, y*) denote the cubic spline interpolation of *f*. In the first step use one-dimensional splines in the *x*-direction. Compute *gxx* = *∂*2*g/∂x*2 at the data points. Plot *g*(*x,i*) for 0 ≤ *x* ≤ 4 and *i* = 0*,*1*,*2*,*3 which is indicated by the solid lines in the figure. *Hint*: After computing *gxx* you can use (1.6) to compute the function in between the data points. (c) From part (b) obtain the values of *g*(1*.*5*,i*) for *i* = 0*,*1*,* 2*,*3. Now use a one-dimensional spline in the *y*-direction to obtain *g*(1*.*5*, y*). Plot *g*(1*.*5*, y*) for 1 ≤ *y* ≤ 2. What is the value of *g*(1*.*5*,*1*.*5)?

We can use the same method to interpolate the data at any other point in the domain. However, repeating the same procedure for each point is not very cost effective, particularly if the system is large. A more effective approach is to obtain two-dimensional polynomials for each subregion of the domain. In this case these polynomials will be of the form:

*P*(*x, y*) = *a*00 + *a*10*x* + *a*01*y* + *a*20*x*2 + *a*11*xy* + *a*02*y*2 + *a*30*x*3

+*a*21*x*2*y* + *a*12*xy*2 + *a*03*y*3 + *a*31*x*3*y* + *a*22*x*2*y*2 +*a*13*xy*3 + *a*32*x*3*y*2 + *a*23*x*2*y*3 + *a*33*x*3*y*3*.*

12 INTERPOLATION

(d) Use one-dimensional splines in the *y*-direction to obtain cubic polynomial expressions for *g*(1*, y*) and *g*(2*, y*) for 1 ≤ *y* ≤ 2 (the dashed lines in the figure). What are the numerical values of *gyy*(1*,*1), *gyy*(1*,*2), *gyy*(2*,*1), and *gyy*(2*,*2)? (e) In part (b) you obtained the *gxx* values at the grid points. Now treat these values as input data (as your new *f* ) and repeat part (d). Obtain cubic polynomial expressions for *gxx*(1*, y*) and *gxx*(2*, y*) for 1 ≤ *y* ≤ 2. What are the values of *gxxyy*(1*,*1), *gxxyy*(1*,*2), *gxxyy*(2*,* 1), and *gxxyy*(2*,* 2)? (f) For a given *y*0 between 1 and 2, you have *g*(1*, y*0) and *g*(2*, y*0) from part (d) and *gxx*(1*, y*0) and *gxx*(2*, y*0) from part (e). Using this information, what will be the spline polynomial expression of *g*(*x, y*0) for 1 ≤ *x* ≤ 2? If you substitute expressions obtained in parts (d) and (e) and do all of the expansions, you will obtain a polynomial of the form presented above. What is *a*33? (You do not need to calculate all of the coefficients.)

4

3*y* 2100 1 2 *x*

3 4

(g) From the expression obtained in part (f ) compute *g*(1*.*5*,*1*.*5) and check if

you have the same answer as in part (c).

**FURTHER READING** Dahlquist, G., and Bj ̈orck, A.  ̊*Numerical Methods.* Prentice-Hall, 1974, Chapters 4

and 7. Ferziger, J. H. *Numerical Methods for Engineering Application*, Second Edition.

Wiley, 1998, Chapter 2. Forsythe, G. E., Malcolm, M. A., and Moler, C. B. *Computer Methods for Mathe-*

*matical Computations.* Prentice-Hall, 1977, Chapter 4. Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes: The Art of Scientific Computing*, Third Edition. Cambridge University Press, 2007, Chapter 3.

2 Numerical Differentiation – Finite Differences

In the next two chapters we develop a set of tools for discrete calculus. This chapter deals with the technique of finite differences for numerical differen- tiation of discrete data. We develop and discuss formulas for calculating the derivative of a smooth function, but only as defined on a discrete set of grid points *x*0*, x*1*,..., xN*. The data may already be tabulated or a table may have been generated from a complicated function or a process. We will focus on finite difference techniques for obtaining numerical values of the derivatives at the grid points. In Chapter 6 another more elaborate technique for numerical differentiation is introduced. Since we have learned from calculus how to differ- entiate any function, no matter how complicated, finite differences are seldom used for approximating the derivatives of explicit functions. This is in contrast to integration, where we frequently have to look up integrals in tables, and often solutions are not known. As will be seen in Chapters 4 and 5, the main appli- cation of finite differences is for obtaining numerical solution of differential equations.**2.1 Construction of Difference Formulas Using Taylor Series**

Finite difference formulas can be easily derived from Taylor series expansions. Let us begin with the simplest approximation for the derivative of *f* (*x*) at the point *xj*, we use the Taylor series:

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

2 *f* (*x j*) +···*.* (2.1) Rearrangement leads to

*f* (*x j*) = *f* (*x j*+1) − *f* (*x j*)

*x j* − *x j*

2 *f* (*x j*) +··· (2.2)

where *x j* = *x j*+1 − *x j* is the mesh size. The first term on the right-hand side of (2.2) is a finite difference approximation to the derivative. The next term is

13

14 NUMERICAL DIFFERENTIATION – FINITE DIFFERENCES

the *leading error* term. In this book, we also use *h* to indicate the mesh size. When the grid points are uniformly spaced, no subscript will be attached to *h* or *x*.

Formula (2.2) is usually recast in the following form for uniform mesh spacing, *h*

*f j* = *f j*+1 − *f j*

*h* + *O*(*h*)*,* (2.3) which is referred to as the first-order *forward difference.* This is the same expression used to define the derivative in calculus, except that in calculus the definition involves the limit, *h* → 0; but here, *h* is finite.

The exponent of *h* in *O*(*hα*) is the order of accuracy of the method. It is a useful measure of accuracy because it gives an indication of how rapidly the accuracy can be improved with refinement of the grid spacing. For example, with a first-order scheme, such as in (2.3), if we reduce the mesh size by a factor of 2, the error (called the *truncation error*) is reduced by approximately a factor of 2. Notice that when we talk about the truncation error of a finite difference scheme, we always refer to the leading error term with the implication that the higher order terms in the Taylor series expansion are much smaller than the leading term. That is, for sufficiently small *h* the higher powers of *h*, which appear as coefficients of the other terms, get smaller. Of course, one should not be concerned with the actual value of*h* in dimensional units; for example, *h* can be in tens of kilometers in atmospheric dynamics problems, which may lead to the concern that the higher order terms that involve higher powers of *h* become larger. This apparent dilemma is quickly overcome by non-dimensionalizing the dependent variable *x* in (2.1). Let us non-dimensionalize *x* with the domain length *L* = *xN* − *x*0. *L* is actually cancelled out in the non-dimensionalization of (2.1), but now we would be certain that the non-dimensional increment *x j*+1 − *x j* is always less than 1, and hence, its higher powers get smaller.

Let us now consider some other popular finite difference formulas. By ex- panding *f j*−1 about *xj*, we can get

*f j* = *f j* − *f j*−1

*h* + *O*(*h*)*,* (2.4) which is also a first-order scheme, called the first-order *backward difference* formula. Higher order schemes (more accurate) can be derived by Taylor series of the function *f* at different points about the point *xj*. For example, the widely used *central difference* formula can be obtained from subtraction of two Taylor series expansions; assuming uniformly spaced data we have

*f j*+1 = *f j* + *hf j* + *h*22 *f j* + *h*36 *f j* +··· (2.5) *f j*−1 = *f j* − *hf j* + *h*22 *f j* − *h*36 *f j* +···*,* (2.6)

2.2 CONSTRUCTION OF FINITE DIFFERENCE SCHEMES 15

which leads to

*f j* = *f j*+1 − 2*h f j*−1

− *h*6 2*f j* +···*.* (2.7)

This is, of course, a second-order formula. That is, if we refine the mesh by a factor of 2, we expect the truncation error to reduce by a factor of 4. In general, we can obtain higher accuracy if we include more points. Here is a fourth-order formula:

*f j* = *f j*−2 − 8 *f j*−1 + 8 *f j*+1 − 12*h f j*+2

+ *O*(*h*4)*.* (2.8)

The main difficulty with higher order formulas occurs near boundaries of the domain. They require the functional values at points outside the domain, which are not available. For example, if the values of the function *f* are known at points *x*0*, x*1*,..., xN* and the derivative of *f* at *x*1 is required, formula (2.8) would require the value of *f* at *x*−1 (in addition to *x*0*, x*1*, x*2*,* and *x*3) which is not available. In practice, to alleviate this problem, we utilize lower order or non- central formulas near boundaries. Similar formulas can be derived for second- or higher order derivatives. For example, the second-order central difference formula for the second derivative is derived by adding (2.5) and (2.6), the two *f j* terms are cancelled, and after a minor rearrangement, we get

*f j* = *f j*+1 − 2 *f j* + *h*2 *f j*−1

+ *O*(*h*2)*.* (2.9)

**2.2 A General Technique for Construction of Finite Difference Schemes**

A finite difference formula is characterized by the points at which the functional values are used and its order of accuracy. For example, the scheme in (2.9) uses the functional values at *j –* 1, *j*, and *j* + 1, and it is second-order accurate. Given a set of points to be used in a formula, called a stencil, it is desirable to construct the formula with the highest order accuracy that involves those points. There is a general procedure for constructing difference schemes that satisfies this objective; it is best described by an actual example. Suppose we want to construct the *most accurate* difference scheme that involves the functional values at points *j*, *j* + 1, and *j* + 2. In other words, given the restriction on the points involved, we ask for the highest order of accuracy that can be achieved. The desired finite difference formula can be written as

*f j* +

∑2*k*=0

*ak f j*+*k* = *O*(?)*,* (2.10)

where*ak* are the coefficients from the linear combination of Taylor series. These coefficients are to be determined so as to maximize the order of the scheme,

16 NUMERICAL DIFFERENTIATION – FINITE DIFFERENCES

which at this point is displayed by a question mark. We take the linear combi- nation of the Taylor series for the terms in formula (2.10) using a convenient table shown below. The table displays the first four terms in the Taylor series expansion of the functional values in the first column.

**TAYLOR TABLE**

***fj f j f j f j***

*f j* 0 1 0 0 *a*0*fj a*0 0 0 0 *a*1*fj*+1 *a*1 *a*1*h a*1 *h*2 2*a*1 *h*6 3*a*2*fj*+2 *a*2 2*ha*2 *a*2 (2*h*)2 2

*a*2 (2*h*)6

3

The left-hand side of (2.10) is the sum of the elements in the first column of the table; the first four terms of its right-hand side are the sum of the rows in the next four columns of the table, respectively. Thus, (2.10) can be constructed by summing the bottom four rows in the table:

*f j* +

∑2*k*=0

*ak f j*+*k* = (*a*0 + *a*1 + *a*2) *f j* + (1 + *a*1*h* + 2*ha*2) *f j*

+

(*a*1*h*22 + *a*2(2*h*)2

2

)

*f j* +

(*a*1*h*36 + *a*2(2*h*)3

6

)

*f j* +···*.* (2.11)

To get the highest accuracy, we must set as many of the low-order terms to zero as possible. We have three free coefficients; therefore, we can set the coefficients of the first three terms to zero:

*a*0 + *a*1 + *a*2 = 0

*a*1*h* + 2*ha*2 = −1 *a*1*h*2*/*2 + 2*a*2*h*2 = 0*.*

Solving these equations leads to

*a*1 = −*h* 2*a*2 = 12*h a*0 = 32*h.* Thus, the resulting (second-order) formula is obtained by substituting these values for the coefficients in (2.10), after a minor rearrangement we obtain

*f j* = −3 *f j* + 4 *f j*+1 − 2*h f j*+2

+ *O*(*h*2)*.* (2.12)

The leading order truncation error is the first term on the right-hand side of (2.11) that we could not set to zero; substituting for *a*1 and *a*2, it becomes

*h*23 *f j .*

2.3 AN ALTERNATIVE MEASURE FOR THE ACCURACY OF FINITE DIFFERENCES 17

Thus, the best we can do is a second-order formula, given the restriction that the formula is to involve the functional values at *j*, *j* + 1, and *j* + 2. It is interesting to note that the magnitude of the truncation error of this formula is twice that of the second-order central difference scheme (2.7).

**EXAMPLE 2.1 Accuracy of Finite Difference Schemes**

We will consider three different finite difference schemes and investigate their accuracy by varying the grid spacing, *h.* The first derivative of a known function *f* will be approximated and compared with the exact derivative. We take

*f*(*x*) = sin*x*

*x*3 *.*

The specific schemes under consideration are the first-, second-, and fourth-order formulas given by (2.3), (2.7), and (2.8). These are numerically evaluated at *x* = 4, and the absolute values of the differences from the exact solution are plotted as a function of *h* in Figure 2.1. Since the approximation errors are proportional to powers of *h*, it is instructive to use a log–log plot to reveal the order of accuracy of the schemes. For each scheme, the curve representing the log |error| vs. log *h* is expected to be a straight line with its slope equal to the order of the scheme. The slopes of the curves in Figure 2.1 verify the order of each method.

1st order

10

2nd order 4th order 10r orre101010101010 10 10 10 100

**Figure 2.1** Truncation error vs. grid spacing for three finite difference schemes.

**2.3 An Alternative Measure for the Accuracy of Finite Differences**

Order of accuracy is the usual indicator of the accuracy of finite difference formulas; it tells us how mesh refinement improves the accuracy. For example,

18 NUMERICAL DIFFERENTIATION – FINITE DIFFERENCES

mesh refinement by a factor of 2 improves the accuracy of a second-order finite difference scheme by fourfold, and for a fourth-order scheme by a factor of 16. Another method for measuring the order of accuracy that is sometimes more informative is the modified wavenumber approach. Here, we ask how well does a finite difference scheme differentiate a certain class of functions, namely sinusoidal functions. Sinusoidal functions are representative because Fourier series are often used to represent arbitrary functions. Of course, more points are required to adequately represent high-frequency sinusoidal functions and to differentiate them accurately. Given a set of points, or grid resolution, we are interested in knowing how well a finite difference scheme can differentiate the more challenging high-frequency sinusoidal functions. We expect that most differencing schemes would do well for the low-frequency, slowly varying func- tions. The solution of non-linear differential equations usually contains several frequencies and the modified wavenumber approach allows one to assess how well different components of the solution are represented.

To illustrate the procedure, consider a pure harmonic function of period *L*,

*f* (*x*) = *eikx,*

where *k* is the wavenumber (or frequency) and can take on any of the following values

*k* = 2*πL n, n* = 0*,* 1*,* 2*,..., N/*2*.* With these values of *k*, each harmonic function would go through an integer number of periods in the domain. The exact derivative is

*f* = *ikf.* (2.13)

We now ask how accurately the second-order central finite difference scheme, for example, computes the derivative of *f* for different values of *k*. Let us discretize a portion of the *x* axis of length *L* with a uniform mesh,

*x j* = *N Lj, j* = 0*,* 1*,* 2*,..., N* − 1*.* On this grid, *eikx* ranges from a constant for *n* = 0, to a highly oscillatory function of period equal to two mesh widths for *n* = *N/*2. The finite difference approximation for the derivative is

*δfδx*

∣∣∣∣*j* = *f j*+1 − 2*h f j*−1

*,*

where *h* = *L/N* is the mesh size and *δ* denotes the discrete differentiation operator. Substituting for *f j* = *eikx j*, we obtain

*δfδx*

∣∣∣∣*j* = *ei*2*πn*( *j*+1)*/N* 2*h* − *ei*2*πn*(*j*−1)*/N*

= *ei*2*πn/N* − 2*h e*−*i*2*πn/N*

*f j.*

2.3 AN ALTERNATIVE MEASURE FOR THE ACCURACY OF FINITE DIFFERENCES 19

**h h k 3**

**Exact 2nd O Central 4th O Central 4th O Padé 2‚ k 100 1 2 3 Figure 2.2** The modified wavenumbers for three finite difference schemes. *h*is the grid spacing. The Pad ́e scheme is introduced in the next section.

Thus,

*δfδx*

∣∣∣∣*j* = *i* sin(2*πn/N*)

*h f j* = *ik f j*

where

*k* = sin(2*πn/N*)

*h .* (2.14) The numerical approximation to the derivative is in the same form as the exact derivative in (2.13) except that*k*is replaced with*k* . In analogy with (2.13), *k* is called the modified wavenumber for the second-order central difference scheme. In an analogous manner, one can derive modified wavenumbers for any finite difference formula. A measure of accuracy of a finite difference scheme is provided by comparing the modified wavenumber *k* with *k.* This comparison for three schemes is provided in Figure 2.2.

Note that the modified wavenumber in (2.14) (which is shown by the dash line in Figure 2.2) is in good agreement with the exact wavenumber at small val- ues of *k.* This is expected because for small values of *k*, *f* is slowly varying and the finite difference scheme is sufficiently accurate for numerical differentiation. For higher values of *k*, however, *f* varies rapidly in the domain, and the finite difference scheme provides a poor approximation for its derivative. Although more accurate finite difference schemes provide better approximations at higher wavenumbers, the accuracy is always better for low wavenumbers compared to that for high wavenumbers. Similarly, we can assess the accuracy of any formula for a higher derivative using the modified wavenumber approach. For example, since the exact second derivative of the harmonic function is −*k*2 exp(*ikx*), one can compare the modified wavenumber of a finite difference scheme for the sec- ond derivative, now labeled*k* 2, with*k*2. As for the first derivative, a typical*k* 2*h*2

20 NUMERICAL DIFFERENTIATION – FINITE DIFFERENCES

vs.*kh* diagram shows better accuracy for small wavenumbers (see Exercise 6). It also turns out that the second-derivative finite difference formulas usually show better accuracy at the high wavenumbers than the first-derivative formulas.

**2.4 Pad ́e Approximations**

The Taylor series procedure for obtaining the most accurate finite difference formula, given the functional values at certain points, can be generalized by inclusion of the derivatives at the neighboring grid points in the formula. For example, we can ask for the most accurate formula that includes *f j, f j*+1, and *f j*−1 in addition to the functional values *f j*, *f j*+1, and *f j*−1. That is, instead of (2.10), we would write

*f j* + *a*0 *f j* + *a*1 *f j*+1 + *a*2 *f j*−1 + *a*3 *f j*+1 + *a*4 *f j*−1 = *O*(?) (2.15) and our task is then to find the five coefficients *a*0*, a*1*,..., a*4 to maximize the order of this approximation. Before worrying about how to use (2.15) for numerical differentiation, let us find the coefficients. We follow the Taylor table procedure for the functional values as well as derivatives appearing in (2.15). The Taylor table is

**TAYLOR TABLE FOR A PAD ́E SCHEME**

***fj f j f j f j f*(*iv*)**

***j f*(*v*)**

***j***

*f j* 0 1 0 0 0 0

*a*0*f j a*0 0 0 0 0 0

*a*1*fj*+1 *a*1 *a*1*h a*1 *h*22 *a*1 *h*36 *a*1 *h*424 *a*1 *h*5

120

*a*2*fj*−1 *a*2 −*a*2*h a*2 *h*22 −*a*2 *h*36 *a*2 *h*424 −*a*2 *h*5

120

*a*3*f j*+1 0 *a*3 *a*3*h a*3 *h*22 *a*3 *h*36 *a*3 *h*424

*a*4*f j*−1 0 *a*4 −*a*4*h a*4 *h*22 −*a*4 *h*36 *a*4 *h*424

As before, we now sum all the rows and set as many of the lower order terms to zero as possible. We have five coefficients and can set the sum of the entries in columns 2 to 6 to zero. The linear equations for the coefficients are

*a*0 + *a*1 + *a*2 = 0 *a*1*h* − *a*2*h* + *a*3 + *a*4 = −1 *a*1*h*22 + *a*2*h*22 + *a*3*h* − *a*4*h* = 0 *a*1*h*3 − *a*2*h*3 + *a*3 + *a*4 = 0 *a*1*h*4 + *a*2*h*4 + *a*3 − *a*4 = 0*.*

2.4 PAD ́E APPROXIMATIONS 21

The solution of this system is

*a*0 = 0 *a*1 = − 4*h* 3*a*2 = 34*h a*3 = *a*4 = 14*.*

Substitution into column 7 and (2.15) and some rearrangement leads to the following Pad ́e formula for numerical differentiation:

*f j*+1 + *f j*−1 + 4 *f j* = *h*3( *f j*+1 − *f j*−1) + *h*430 *f j v,* (2.16)

where *j* = 1*,* 2*,* 3*,..., n* − 1.

This is a tridiagonal system of equations for *f j*. There are*n–*1 equations for *n* + 1 unknowns. To get the additional equations, special treatment is required near the boundaries. Usually, lower order one-sided difference formulas are used to approximate *f* 0 and *f n*. For example, the following third-order formulas provide the additional equations that would complete the set given by (2.16)

*f* 0 + 2 *f* 1 = 1*h*

(−52 *f*0 + 2 *f*1 + 12 *f*2)

(2.17) *f n* + 2 *f n*−1 = 1*h*

(52 *fn* − 2 *fn*−1 − 12 *fn*−2)*.*

In matrix form, (2.16) and (2.17) are written as ⎡⎢⎢⎢⎢⎢⎢⎢⎢⎢⎢⎢⎣1 2 0 0 0 ··· 0 1 4 1 0 0 ··· 0 0 1 4 1 0 ··· 0 ... ... ... ... ... ... ... ... ... ... ... ... ... ... 0 0 0 *...* 1 4 1 0 0 0 0 ··· 2 1⎤⎥⎥⎥⎥⎥⎥⎥⎥⎥⎥⎥⎦ ⎡⎢⎢⎢⎢⎢⎢⎢⎢⎢⎢⎢⎣

⎤⎥⎥⎥⎥⎥⎥⎥⎥⎥⎥⎥⎦

⎡⎢⎢⎢⎢⎢⎢⎢⎢⎢⎢⎢⎣

−52 *f*0 + 2 *f*1 + 3( *f*2 − *f*0)

3( *f*3 − ...12 *f*2

⎤*f*1) ... 3( *fn* − *fn*−2) *f* 0*f* 1*f* ...2... *f n*−1 *f n*

= *h*

1⎥⎥⎥⎥⎥⎥⎥⎥⎥⎥⎥⎦*.* 52 *fn* − 2 *fn*−1 − 12 *fn*−2

(2.18)

In choosing the boundary schemes, we consider two factors. First, in order to avoid writing a special code to solve the system of equations, the bandwidth of the matrix should not be increased. For example, the boundary scheme in (2.18) preserves the tridiagonal structure of the matrix which allows one to use a standard tridiagonal solver. Second, the boundary stencil should not be wider than the interior stencil. For example, if the interior stencil at *x*1 involves only the functional and derivative values at *x*0*, x*1, and *x*2, the boundary stencil should not include *x*3. This constraint is derived from certain considerations in numer- ical solution of differential boundary value problems using finite differences (Chapter 4). The same constraint also applies to high-order standard non-Pad ́e type schemes. For this reason, the order of the boundary scheme is usually lower

22 NUMERICAL DIFFERENTIATION – FINITE DIFFERENCES

than that of the interior scheme. However, there is substantial evidence from numerical tests that the additional errors due to a lower order boundary scheme are confined to the points near the boundaries.

**EXAMPLE 2.2 Pad ́e Differentiation Using a Lower Order Boundary Scheme**

We will use the fourth-order Pad ́e scheme (2.16) and the third-order bound- ary schemes given by (2.17) to differentiate

*f*(*x*) = sin 5*x* 0 ≤ *x* ≤ 3*.*

Fifteen uniformly spaced points are used. The result is plotted in Figure 2.3. Although relatively few grid points are used, the Pad ́e scheme is remarkably accurate. Note that the main discrepancies are near boundaries where lower order schemes are used.

4th Order Pade Differentiation 7.5

Computed Derivative Exact Derivative

5.0

e vitavireD2.5 0

-2.5-5.0

0 1 x

2 3

**Figure 2.3** Computed derivative of the function in Example 2.2 using a fourth-order Pad ́e scheme and exact derivative. The symbols mark the uniformly spaced grid points.

Note that despite its high order of accuracy, the Pad ́e scheme (2.16) is compact; that is, it requires information only from the neighboring points, *j* + 1 and *j* – 1. Furthermore, as can be seen from Figure 2.1, this scheme has a more accurate modified wavenumber than the standard fourth-order scheme given by (2.8). Pad ́e schemes are global in the sense that to obtain the derivative at a point, the functional values at all the points are required; one either gets the derivatives at all the points or none at all.

Pad ́e schemes can also be easily constructed for higher derivatives. For example, for the three-point central stencil the following fourth-order formula

2.5 NON-UNIFORM GRIDS 23

can be derived using the Taylor table approach:

112 *f i*−1 + 1012 *f i* + 112 *f i*+1 = *fi*+1 − 2 *fi* + *fi*−1

*h*2 *.* (2.19)

**2.5 Non-Uniform Grids**

Often the function *f* varies rapidly in a part of the domain, and it has a mild variation elsewhere. In computationally intensive applications, it is considered wasteful to use a fine grid capable of resolving the rapid variations of *f* ev- erywhere in the domain. One should use a non-uniform grid spacing. In some problems, such as boundary layers in fluid flow problems, the regions of rapid variations are known a priori, and grid points can be clustered where needed. There are also (adaptive) techniques that estimate the grid requirements as the solution progresses and place additional grid points in the regions of rapid variations. For now, we will just concern ourselves with finite differencing on non-uniformly spaced meshes.

Typical finite difference formulas for the first and second derivatives are

*f j* = *f j*+1 − *f j*−1

*x j*+1 − *x j*−1 (2.20)

and

*f j* = 2[ *f j*−1

*h j*(*h j* + *h j*+1) − *f j*

*h jh j*+1 + *f j*+1

*h j*+1(*h j* + *h j*+1)]*,* (2.21)

where*h j* = *x j* − *x j*−1. Finite difference formulas for non-uniform meshes gen- erally have a lower order of accuracy than their counterparts with the same sten- cil but defined for uniform meshes. For example, (2.21) is strictly a first-order approximation whereas its counterpart on a uniform mesh (2.9) is second-order accurate. The lower accuracy is due to reduced cancellations in Taylor series expansions because of the lack of symmetry in the meshes.

An alternative to the cumbersome derivation of finite difference formulas on non-uniform meshes is to use a coordinate transformation. One may transform the independent variable to another coordinate system that is chosen to account for local variations of the function. Uniform mesh spacing in the new coordinate system would correspond to non-uniform mesh spacing in the original (physical) coordinate (see Figure 2.4). For example, the transformation

*ζ* = cos−1 *x*

transforms 0 ≤ *x* ≤ 1 to 0 ≤ *ζ* ≤ *π*2. Uniform spacing in *ζ*, given by

*ζj* = *π*2*N j j* = 0*,* 1*,* 2*,..., N,*

24 NUMERICAL DIFFERENTIATION – FINITE DIFFERENCES

xxj-1 jxj+1

ζ j-1 ζ j ζ

j+1

**Figure 2.4** Uniform mesh spacing in*ζ* corresponds to non-uniform mesh spacing in *x*.

corresponds to a very fine mesh spacing near *x* = 1 and a coarse mesh near *x* = 0. In general, for the transformation

*ζ* = *g*(*x*)

we use the chain rule to transform the derivatives to the new coordinate system

*dfdx* = *dζdx*

*dfdζ* = *g dfdζ* (2.22)

*d*2*f dx*2 = *dx*

*d*[*g dfdζ*

]

= *g dfdζ* + (*g* )2*dζ d*2*f*

2*.* (2.23)

Finite difference approximations for uniform meshes are then used to approx- imate *df/dζ* and *d*2*f/dζ* 2.

**EXAMPLE 2.3 Calculation of Derivatives on a Non-uniform Mesh**

Let *f* be a certain function defined on the grid points

*x j* = tanh−1 *ζ j* where*ζ j* = 0*.*9(2*jN* )− 1*, j* = 0*,..., N.*

The value of *f* at *x j* is denoted by *f j*. The *x* mesh is non-uniform and was constructed to have clustered points in the middle of the domain where *f* is supposed to exhibit rapid variations. The *x* mesh is shown versus the *ζ* mesh in Figure 2.5 for *N* = 18.

From (2.22), the first derivative of *f* at *x j* is

*d f dx*

∣∣∣∣*x j*

= *g* (*x j*) *d f*∣*dζ* ∣∣∣*ζj*

*.*

The central difference approximation to

*d fdζ*

∣∣∣∣*ζj*

EXERCISES 25

1.47

−1.47

−0.9 −0.7 −0.5 −0.3 −0.1 0.1 0.3 0.5 0.7 0.9 ζ

**Figure 2.5** The non-uniform *x* mesh versus the uniform *ζ* mesh in Example 2.3.

is simply (*f j*+1 − *f j*−1)*/*(2 *ζ*). In order to see this, let *y*1(*x*) describe *f* as a function of *x*. Then *f* as a function of *ζ* is given by *f* = *y*1(*x*) = *y*1(*g*−1(*ζ*)) = *y*2(*ζ*), where *y*2 is the composition of *y*1 and *g*−1. Thus

*d f dζ*

0.87

0.55

0.31 0.10 −0.10 −0.31

−0.55

−0.87

∣∣∣∣*ζ j*

≈ *y*2(*ζ j*+1) 2 − *ζ y*2(*ζ j*−1)

= *y*1(*x j*+1) − *y*1(*x j*−1)

2 *ζ* = *f j*+1 − *f j*−1

2 *ζ*

and

*d f dx*

∣∣∣∣*x j*

≈ sech2(*x j*) *f j*+1 2 − *ζ f j*−1

*.*

An expression for the second derivative of *f* is obtained similarly.

These numerical derivatives are valid for *j* = 1*,..., N* − 1. Derivatives at *j* = 0 and *N* are obtained by using one-sided difference approximations to *d f/dζ* and*d* 2*f/dζ* 2.

**EXERCISES**

1. Consider the central finite difference operator *δ/δx* defined by

*δun δx* = *un*+1 2*h* − *un*−1

*.*

(a) In calculus we have *duv*

*dx* = *u dx dv*+ *vdudx .* Does the following analogous finite difference expression hold?

*δ*(*unvn*)

*δx* = *un δvn*

*dx* + *vn δuδx n*

*.*

26 NUMERICAL DIFFERENTIATION – FINITE DIFFERENCES

(b) Show that

*δ*(*unvn*)

*δx* = ̄*un δvn*

*δx* + ̄*vn δun δx* where an overbar indicates average over the nearest neighbors,

̄*un* = 12(*un*+1 + *un*−1)*.*

(c) Show that

*φδψδx* = *δδx*  ̄*φψ* − *ψ δφδx .*

(d) Derive a finite difference formula for the second-derivative operator that is obtained from two applications of the first-derivative finite difference operator. Compare the leading error term of this formula and the popular second-derivative formula

*un*+1 − 2*un* + *un*−1

*h*2 *.* Use both schemes to calculate the second derivative of sin 5*x* at *x* = 1.5. Plot the absolute values of the errors as a function of *h* on a log–log plot similar to Figure 2.1. Use 10−4 ≤ *h* ≤ 100. Discuss your plot. 2. Find the most accurate formula for the first derivative at *xi* utilizing known values of *f* at *xi*−1*, xi, xi*+1, and *xi*+2. The points are uniformly spaced. Give the leading error term and state the order of the method. 3. Verify that the modified wavenumber for the fourth-order Pad ́e scheme for the

first derivative is

*k* = 3 sin(*k* )

(2 + cos(*k* ))*.*

4. A general Pad ́e type boundary scheme (at *i* = 0) for the first derivative which

does not alter the tridiagonal structure of the matrix in (2.16) can be written as

*f* 0 + *αf* 1 = 1*h* (*af*0 + *bf*1 + *cf*2 + *df*3)*.*

(a) Show that requiring this scheme to be at least third-order accurate would

constrain the coefficients to *a* = −11 + 2*α*

6 *, b* = 6 − *α*

2 *, c* = 2*α* − 3

2 *, d* = 2 − *α*

6 *.* Which value of *α* would you choose and why? (b) Find all the coefficients such that the scheme would be fourth-order

accurate. 5. Modified wavenumbers for non-central finite difference schemes are complex. Derive the modified wavenumber for the down-wind scheme given by (2.12). Plot its real and imaginary parts separately and discuss your results.

EXERCISES 27

6. *Modified wavenumber for second-derivative operators.*

Recall that the second derivative of *f* = exp(*ikx*) is −*k*2*f* . Application of a finite difference operator for second derivative to *f* would lead to −*k* 2*f,* where *k* 2 is the ‘modified wavenumber’ for the second-derivative. The modi- fied wavenumber method for assessing the accuracy of second-derivative finite difference formulas is then to compare the corresponding *k* 2 with *k*2 in a plot such as in Figure 2.2 (but now, *k* 2*h*2 and *k*2*h*2 vs*.kh,* 0 ≤ *kh* ≤ *π*). (a) Use the modified wavenumber analysis to assess the accuracy of the central

difference formula

*f j* = *f j*+1 − 2 *f j* + *h*2 *f j*−1

*.*

(b) Use Taylor series to show that the Pad ́e formula given by (2.19) is fourth-

order accurate. (c) Use the modified wavenumber analysis to compare the schemes in (a) and (b). (Hint: To derive modified wavenumbers for Pad ́e type schemes, replace *f j* with −*k* 2 exp(*ikx j*), etc.) (d) Show that *k* 2*h*2 − *k*2*h*2 = *O*(*k*6*h*6) for the fourth-order Pad ́e scheme as

*kh* → 0*.* Show also that the *kh*→0

lim *k* 2*k*2 = 1. 7. *Pad ́e operators.*

(a) Show that the fourth-order Pad ́e operator for second derivative can formally

be operator written for as the 1+ second 121*Dh*2

2*D*2*,* derivative.

where *D*2 is the second-order central difference

(b) Show that the fourth-order Pad ́e operator for the first derivative can be

written for the first as 1+ derivative.

16*Dh*0

2*D*2*,*where D0 is the second-order central difference operator

These formulations are useful when using Pad ́e schemes to solve boundary value problems (see the next problem). 8. In numerical solution of boundary value problems in differential equations, we can sometimes use the physics of the problem not only to enforce boundary conditions but also to maintain high-order accuracy near the boundary. For example, we may know the heat flux through a surface or displacement of a beam specified at one end. We can use this information to produce better estimates of the derivatives near the boundary.

Suppose we want to numerically solve the following boundary value prob- lem with Neumann boundary conditions:

*d*2*y dx*2 + *y* = *x*3*,* 0 ≤ *x* ≤ 1 *y* (0) = *y* (1) = 0*.*

We discretize the domain using grid points *xi* = (*i* − 0*.*5)*h,i* = 1*,..., N*. Note that there are no grid points on the boundaries as shown in the figure below. In this problem, *yi* is the numerical estimate of *y* at *xi*. By using a finite difference scheme, we and transform the ODE can estimate into a linear *y* system *i* in terms of linear combinations of *yi*’s

of equations.

28 NUMERICAL DIFFERENTIATION – FINITE DIFFERENCES

Use the fourth-order Pad ́e formula (2.19) for the interior points. (a) For in the the following left boundary, form:

derive a third-order Pad ́e scheme to approximate *y* 0 *y* 1 + *b*2*y* 2 = *a*1*y*1 + *a*2*y*2 + *a*3*y*3 + *a*4*y b* + *O*(*h*3)*,*

(b) where Repeat *y* the *b* = previous *y* (0), which step for is known the right from boundary.

the boundary condition at *x* = 0.

(c) Using the finite difference formulae derived above, we can write the fol-

lowing linear relation:

**A**⎡⎢⎢⎢⎢⎣

*y* ...1⎤⎥⎥⎥⎥⎦ = **B** ⎡⎢⎢⎢⎢⎣ ⎤...⎥⎥⎥⎥⎦*. y N*

What are the elements of the matrices **A** and **B** operating on the interior and boundary nodes? (d) Use this relationship to transform the ODE into a system with *yi*’s as unknowns. Use *N* = 24 and solve this system. Do you actually have to invert *A*? Plot the exact and numerical solutions. Discuss your result. How are the Neumann boundary conditions enforced into the discretized boundary value problem? 9. Consider the function:

*f* (*x*) = sin((4 − *x*)(4 + *x*))*,* 0 ≤ *x* ≤ 8*.*

Use a uniform grid with *N* + 1 points, where *N* = 32, to numerically compute the second derivative of *f* as explained below: (a) Derive a finite difference scheme for (1.7) in the text.

*f j* using the cubic spline formula

(b) Use Taylor series to find the order of accuracy of this scheme. (c) Solve the spline formula resulting tridiagonal applies only to the system interior for points. *f j* . Remember that the cubic To account for the bound- ary points, derive a first-order one-sided scheme. For example, for the left boundary, the exact and construct numerical a first-order solutions. scheme Discuss for your *f* 0 using *f*0, *f*1, and results.

*f*2. Plot

(d) Use the fourth-order Pad ́e scheme for first-order one-sided schemes derived *f* in *j* given in (2.19) in the text. Use the the previous step for the bound- ary points. Solve the resulting tridiagonal system and plot the exact and numerical solutions. Discuss your results.

*y*...1...*yN*

FURTHER READING 29

(e) Investigate the accuracy of both schemes at *x* = 4 by varying the grid spacing *h.* That is, for each scheme plot log |error| vs. log(*h*), where error is the difference between the exact and numerical solution. Verify the order of each method by calculating the slopes of the curves. 10. *Nonuniform mesh*.

Consider the function *f* (*x*) = 1 − *x*8 and a grid defined as follows:

⎧⎨⎩

*j* = 0*,* 1*,*2*,..., N ξj* = −1 + *x j* = *a* 1tanh(*ξ*2*jN j* tanh−1[*a*]) 0 *< a <* 1*.* The parameter *a* can be used to adjust the spacing of the grid points, with large *a* placing more points near the boundaries. For this problem take *a* = 0.98 and *N* = 32. (a) Compute and plot the derivative of *f* with the central difference formula (2.20) and the coordinate transformation method described in Section 2.5 and compare with the exact derivative in −1 ≤ *x <* 1. How would the results change with *a* = 0.9? (b) Repeat part (a) with the transformation: ⎧⎨*j* ⎩

*ξx j j* = = = 0*,* cos(*ξπ N j*1*,*2*,..., j*)*.*

*N*

Which transformation would you choose, the one in part (a) or this one? (c) How many uniformly spaced grid points would be required to achieve the same accuracy as the transformation method in (a)? The maximum error in the derivative over the domain for the uniform case should be less than or equal to the maximum error over the domain for the transformed case.

**FURTHER READING** Dahlquist, G., and Bj ̈orck, ̊A. *Numerical Methods*. Prentice-Hall, 1974, Chapter 7. Lapidus, L., and Pinder, George F. *Numerical Solution of Partial Differential*

*Equations in Science and Engineering.* Wiley, 1982, Chapter 2.

3 Numerical Integration

Generally, numerical methods for integration or quadrature are needed more in practice than finite difference formulae for differentiation. The reason is that while differentiation is always possible to do analytically (even though it might sometimes be tedious) some integrals are difficult or impossible to do analytically. Therefore, we often refer to tables to evaluate non-trivial integrals. In this chapter we will introduce numerical methods that are used for evaluation of definite integrals that cannot be found in the tables; that is, they are impossible or too tedious to do analytically. Some of the elementary methods that are introduced can also be used to evaluate integrals where the integrand is only defined on a discrete grid or in tabular form.

Throughout the chapter, we will discuss methods for evaluation of the defi- nite integral of the function *f* in the interval [*a*, *b*],

*I* =

∫ *ba f* (*x*)*dx.* We will assume that the functional values are known on a set of discrete points, *x*0 = *a, x*1*, x*2*,...,xn* = *b*. If *f* is known analytically, the user or the algorithm would determine the location of the discrete points *x j*. On the other hand if the data on *f* are available only in tabular form, then the locations of the grid points are fixed a priori and only a limited class of methods are applicable.

**3.1 Trapezoidal and Simpson’s Rules**

For one interval, *xi* ≤ *x* ≤ *xi*+1, the trapezoidal rule is given by

∫ *xi*+1

*xi f* (*x*)*dx* ≈ 2 *x*( *fi* + *fi*+1) (3.1) where *x* = *xi*+1 − *xi*. The geometrical foundation of this formula is that the function *f* in the interval is approximated by a straight line passing through the end points, and the area under the curve in the interval is approximated by

30

3.2 ERROR ANALYSIS 31

f(x)

xj xj+1

**Figure 3.1** Trapezoidal rule; approximating *f* by a straight line between *xj* and *xj*+1.

the area of the resulting trapezoid (see Figure 3.1). For the entire interval [*a*, *b*] the trapezoidal rule is obtained by adding the integrals over all sub-intervals:

*I* ≈ *h*

⎛⎝12 *f*0 + 12 *fn* +

*f j*⎞⎠*,* (3.2)

where *uniform spacing x* = *h* is assumed.

If we approximate *f* in each interval by a parabola rather than a straight line, then the resulting quadrature formula is known as *Simpson’s rule.* To uniquely define a parabola as a fitting function, it must pass through three points (or two intervals). Thus, ∫ Simpson’s formula for the integral from *xj* to *x j*+2 is given by

*x j*+2 *x j f* (*x*)*dx* ≈ 3

*x*∑*n*−1*j*=1 [ *f* (*x j*) + 4 *f* (*x j*+1) + *f* (*x j*+2)]*.* (3.3)

Similarly, Simpson’s rule for the entire domain with uniform mesh spacing,

*x* = *h* is given by

*I* ≈ *h*3

⎛⎜⎜⎝ *f*0 + *fn* + 4

*f j*⎞⎟⎟⎠*.* (3.4)

Note that in order to use Simpson’s rule for the entire interval of integration, the total number of points (*n* + 1) must be odd (even number of panels).

Before we discuss the accuracy of these formulae, notice that they both can be written in the compact form:

*I* =

∑*n*−1*j*=1 *j*=odd

*f j* + 2

∑*n*−2*j*=2 *j*=even

∫ *ba f* (*x*)*dx* ≈

∑*ni*=0

*wi f* (*xi*) (3.5)

where *wi* are the weights. For example, and *wi* = *h* for *i* = 1*,*2*,...,n* − 1.

for the trapezoidal rule *w*0 = *wn* = *h*2 **3.2 Error Analysis**

We will now establish the accuracy of these formulas using Taylor series ex- pansions. It turns out that it is easier to build our analysis around the so-called

32 NUMERICAL INTEGRATION

f(x)

xj yj

xj+1

**Figure 3.2** Rectangle rule; approximating *f* in the interval between *x j* and *x j*+1 by its value at the midpoint.

rectangle (or midpoint) rule of integration; the order of accuracy of the trape- zoidal and Simpson rules are then easily derived from that of the rectangle rule.

Consider the rectangle rule (Figure 3.2) for the interval [*xi,xi*+1],

∫ *xi*+1

*xi f* (*x*)*dx* ≈ *hi f* (*yi*)*,* (3.6)

where *yi* = (*xi* + *xi*+1)*/*2 is the midpoint of the interval [*xi, xi*+1] and *hi* is its width. Let’s replace the integrand with its Taylor series about *yi*

*f* (*x*) = *f* (*yi*) + (*x* − *yi*) *f* (*yi*) + 12(*x* − *yi*)2 *f* (*yi*)

+ 16(*x* − *yi*)3 *f* (*yi*) +···*.* Substitution in (3.6) leads to

∫ *xi*+1

*xi f* (*x*)*dx* = *hi f* (*yi*) + 12(*x* − *yi*)2∣∣∣∣*xi*+1

*xi f* (*yi*) + 16(*x* − *yi*)3∣∣∣∣*xi*+1

*xi f* (*yi*) +···*.*

All the terms with even powers of (*x* − *yi*) vanish, and we obtain

∫ *xi*+1

*xi f* (*x*)*dx* = *hi f* (*yi*) + *h*3*i*24 *f* (*yi*) + 19201

*hi* 5*f* (*iv*)(*yi*) +···*.* (3.7)

Thus, for one interval, the rectangle rule is third-order accurate.

Now let us perform an error analysis for the trapezoidal rule. Consider the Taylor series expansions for the functional values appearing on the right-hand side of (3.1):

*f* (*xi*) = *f* (*yi*) − 12*hi f* (*yi*) + 18*hi* 2*f i* (*yi*) − 481*hi* 3*f* (*yi*) +··· *f* (*xi*+1) = *f* (*yi*) + 12*hi f* (*yi*) + 18*hi* 2*f i* (*yi*) + 481*hi* 3*f* (*yi*) +···*.*

3.2 ERROR ANALYSIS 33

Adding these two expressions and dividing by 2 yields,

*f* (*xi*) + 2 *f* (*xi*+1)

= *f* (*yi*) + 18*h*2*i f* (*yi*) + 3841*h*4*i f* (*iv*)(*yi*) +···*.* Now we can use this expression to solve for *f* (*yi*) and then substitute it into (3.7)

∫ *xi*+1

*xi f* (*x*)*dx* = *hi f* (*xi*) + *f* (*xi*+1)

2 − 121*h*3*i f* (*yi*) − 4801*h*5*i f* (*iv*) (*yi*) +···*.* (3.8) Thus, for one interval the trapezoidal rule is also third-order accurate, and its leading truncation error is twice in magnitude but has the opposite sign of the truncation error of the rectangle rule. This is a bit surprising since we would expect approximating a function in an interval by a straight line (which is the basis of the trapezoidal method) to be more accurate than approximating it by a horizontal line passing through the function at the midpoint of the interval. Apparently, error cancellations in evaluating the integral lead to higher accuracy for the rectangle rule.

To obtain the order of accuracy for approximating the integral for the *entire domain*, we can sum both sides of (3.8); assuming uniform spacing, i.e.,*hi* = , we will have*I* =

∫ *ba f* (*x*)*dx* =

∫ *xi*+1

*xi f* (*x*)*dx*

= 2

∑*n*−1⎛*i*=0

⎝ *f* (*a*) + *f* (*b*) + 2

*f j*⎞∑*n*−1⎠ − *j*=1

3 ∑*n*−112 *i*=0

*f* (*yi*)

5 *n*−1− 480

∑*f* (*iv*) (*yi*) +···*. i*=0

(3.9)

Now, we will apply the mean value theorem of integral calculus to the summations. The mean value theorem states that for sufficiently smooth *f* there exists a point ̄*x* in the interval [*a*, *b*] such that

∑*n*−1*i*=0

*f* (*yi*) = *nf* ( ̄*x*)*.*

Similarly, there is a point *ξ* in [*a*, *b*], such that

∑*n*−1*i*=0

*f* (*iv*)(*yi*) = *nf* (*iv*)(*ξ*)*.*

Noting that *n* = (*b* − *a*)*/* and using the results of the mean value theorem in

34 NUMERICAL INTEGRATION

(3.9), we obtain

∫ *bI* =

*f j*⎤⎦

− (*b* − *a*)

⎡⎣ *f* (*a*) + *f* (*b*) + 2 *a f* (*x*)*dx* = 2

∑*n*−1*j*=1 4 480 *f* (*iv*) (*ξ*) +···*.* (3.10) Thus, the trapezoidal rule for the *entire interval* is *second-order* accurate. One can easily show that the Simpson’s formula for one panel [*xi,xi*+2] can be written as

*S*( *f* ) = 23 *R*( *f* ) + 13*T*( *f* )*,* where *R*( *f* ) and *T*( *f* ) denote rectangle and trapezoidal rules, respectively, applied to the function *f*. Note that the midpoint of the interval [*xi,xi*+2] is *xi*+1. Using (3.7) and (3.8) (modified for the interval [*xi,xi*+2]) and the mean value theorem, we see that Simpson’s rule is *fourth-order* accurate for the entire interval [*a*, *b*].

**3.3 Trapezoidal Rule with End-Correction**

This rule is easily derived by simply substituting in (3.8) for *f* (*yi*), the second- order central difference formula, *f* (*yi*) = ( *f i*+1 − *f i* )*/hi* + *O*(*h*2*i* ):

*Ii* = *hi* 2

12 *f* ( ̄*x*) − (*b* − *a*)

*fi* + *fi*+1

2 − 121*hi* 3*f i*+1 − *hi f i*

+ *O*(*h*5*i*

)*.*

Once again, to get a simple global integration formula, we will assume constant step size, *hi* = *h* = const, and sum over the entire interval

*I* = *h*2

∑*n*−1( *fi* + *i*=0

*fi*+1) − *h*212

∑*n*−1( *f i*+1 − *i*=0

*f i* ) + *O*(*h*4)*.*

Cancellations in the second summation on the right-hand side lead to

*I* = *h*2

∑*n*−1*i*=0

( *fi* + *fi*+1) − 12*h*2( *f* (*b*) − *f* (*a*)) + *O*(*h*4)*.* (3.11)

Thus, the trapezoidal rule with end-correction is *fourth-order* accurate and can be readily applied without much additional work, provided that the derivatives of the integrand at the end points are known.

**EXAMPLE 3.1 Quadrature**

Consider the integral ∫ *π*1

sin 2*x*3 *x*

*dx.*

3.4 ROMBERG INTEGRATION AND RICHARDSON EXTRAPOLATION 35

We will numerically evaluate this integral using the trapezoidal rule (3.2), Simpson’s rule (3.4), and trapezoidal rule with end-correction (3.11). This integral has an analytical solution in terms of Si(*x*), sine integrals (see *Hand- book of Mathematical Functions*, by Abramowitz & Stegun, p. 231), and may be numerically evaluated to an arbitrary degree of accuracy for use as an ‘exact’ solution, allowing us to evaluate our quadrature techniques. The results of the numerical calculations as well as percent errors*†* for the quadra- ture techniques are presented below for *n* = 8 and *n* = 32 panels in the inte- gration. The ‘exact’ solution is *I* = 0.1985572988. . . .

***n*** = **8 Result % Error** Trapezoidal 0.204304 2.894303 Simpson 0.198834 0.139596 End-Correct. 0.198476 0.040948

***n*** = **32 Result % Error** Trapezoidal 0.198921 0.183286 Simpson 0.198559 0.000661 End-Correct. 0.198557 0.000167

We see that the higher order Simpson’s rule and trapezoidal with end- correction outperform the plain trapezoidal rule.

**3.4 Romberg Integration and Richardson Extrapolation**

Richardson extrapolation is a powerful technique for obtaining an accurate numerical solution of a quantity (e.g., integral, derivative, etc.) by combining two or more less accurate solutions. The essential ingredient for application of the technique is knowledge of the form of the truncation error of the basic numerical method used. We shall demonstrate an application of the Richardson extrapolation by using it to improve the accuracy of the integral

*I* =

∫ *ba f* (*x*)*dx*

with the trapezoidal rule as the basic numerical method. This algorithm is known as the *Romberg integration.*

*†* The percent error (% error) is the absolute value of the truncation error divided by the exact

solution and multiplied by 100:

%error =

∣∣∣∣exact solution − numerical solution exact solution

∣∣∣∣ × 100*.*

36 NUMERICAL INTEGRATION

From our error analysis for the trapezoidal rule (3.10), we have

*I* = *h*2

⎡⎣ *f* (*a*) + *f* (*b*) + 2

*f j*⎤⎦ + *c*1*h*2 + *c*2*h*4 + *c*3*h*6 +···*.* (3.12)

Let the trapezoidal approximation with uniform mesh of size *h* be denoted by *I* ̃1

*I* ̃1 = *h*2

∑*n*−1*j*=1

⎡⎣ *f* (*a*) + *f* (*b*) + 2

*f j*⎤⎦*.* (3.13)

The exact integral and the trapezoidal expression are related by

*I* ̃1 = *I* − *c*1*h*2 − *c*2*h*4 − *c*3*h*6 −···*.* (3.14)

Now, suppose we evaluate call this estimate *I* ̃2

the integral with half the step size *h*1 = *h/*2. Let’s

*I* ̃2 = *I* − *c*1*h*24 − *c*2*h*416 − *c*3*h*664 −···*.* (3.15)

We can eliminate *O*(*h*2) terms by taking a linear combination of (3.14) and (3.15) to obtain

*I* ̃12 = 4 *I* ̃2 − *I* ̃1

3 = *I* + 14*c*2*h*4 + 165*c*3*h*6 +···*.* (3.16)

This is a *fourth-order* approximation for *I*. In fact, (3.16) is a rediscovery of Simpson’s rule. We have combined two estimates of *I* to obtain a more accurate estimate; this procedure is called the Richardson extrapolation and can be repeated to obtain still higher accuracy.

Let’s evaluate *I* with *h*2 = *h*1*/*2 = *h/*4; we obtain

*I* ̃3 = *I* − *c*1*h*216 − *c*2 *h*4

256 − *c*3 ∑*n*−1*j*=1 *h*6 4096 −···*.* (3.17)

To get another fourth-order estimate, we will combine *I* ̃3 with *I* ̃2:

*I* ̃23 = 4 *I* ̃3 − *I* ̃2

3 = *I* + 641*c*2*h*4 + 10245

*c*3*h*6 +···*.* (3.18)

Now that we have two fourth-order estimates, we can combine them and elim- inate the *O*(*h*4) terms. Elimination of the *O*(*h*4) terms between (3.16) and (3.18) results in a *sixth-order accurate* formula. This process can be continued indefinitely. The essence of the Romberg integration algorithm just described is illustrated in the following diagram. In typical Romberg integration subrou- tines, the user specifies an error tolerance, and the algorithm uses the Richardson

3.5 ADAPTIVE QUADRATURE 37

extrapolation as many times as necessary to achieve it.

*O*(*h*2) *O*(*h*4) *O*(*h*6) ~*I*1

~*I*2~*I*3Eqn. (3.18)

**EXAMPLE 3.2 Romberg Integration**

We will numerically evaluate the integral from Example 3.1 using the Romberg integration. The basis for our integration will be the trapezoidal rule. The integration will be set to automatically stop when the solution varies less than 0.1% between levels – we may thus specify how accurate we wish our solution to be. The table below shows the Romberg integration in progress. The first column indicates the number of panels used to compute the integral using the trapezoidal rule.

2 4 8 16  *̃I ̃I ̃I ̃I*2 3 4 1 = = = = 0*.*220713 0*.*204304 0*.*200009 0*.*278173

0.201560 0.198834 0.198653 0.198578 0.198560 0.198559

The % error of this calculation is 0.00074. We see that using only a second- order method as a basis we are able to generate an *O*(*h*8) method and a 0.00074% error at the cost of only 17 function evaluations.

**3.5 Adaptive Quadrature**

Often it is wasteful to use the same mesh size everywhere in the interval of integration [*a*, *b*]. The major cost of numerical integration is the number of function evaluations required, which is obviously related to the number of mesh points used. Thus, to reduce the computational effort, one should use a fine mesh only in regions of rapid functional variation and a coarser mesh where the integrand is varying slowly. Adaptive quadrature techniques automatically determine panel sizes in various regions so that the computed result meets some prescribed accuracy requirement supplied by the user. That is, with the minimum number of function evaluations, we would like a numerical estimate *I*  ̃of the integral such that

∣∣∣∣ *I*  ̃−

Eqn. (3.16)

∫ *ba f* (*x*)*dx*∣∣∣∣≤ *ε*

where *ε* is the error tolerance provided by the user.

38 NUMERICAL INTEGRATION

To demonstrate the technique, we will use Simpson’s rule as the base method. Let’s divide the interval [*a*, *b*] into subintervals [*xi, xi*+1]. Divide this interval into two panels and use Simpson’s rule to obtain

*Si* = *hi*6

[

*f* (*xi*) + 4 *f*

(*xi* + *hi*2

)

]+ *f* (*xi* + *hi*)*.*

Now, divide the interval into four panels, and obtain another estimate for the integral

*Si* (2)

= *hi*12

[

*f* (*xi*) + 4 *f*

(*xi* + *hi*4

)

+ 2 *f*

(*xi* + *hi*2

)

+ 4 *f*

(*xi* + 3*hi*4

)

]+ *f* (*xi* + *hi*)*.*

The basic idea, as will be shown, is to we *Si* (2)

will , and use obtain *Si* (2)

an estimate for the accuracy compare of *Si* (2)

the . If two the approximations, accuracy is acceptable, *Si* and

for the interval and start working on the next interval; otherwise, the method further subdivides the interval. Let *Ii* denote the exact integral in [*xi,xi*+1]. From our error analysis we know that Simpson’s rule is *locally* fifth- order accurate,

*Ii* − *Si* = *ch*5*i f* (*iv*) (*xi* + *hi*2

)

+··· (3.19)

and for the refined interval, we simply add the two truncation errors

*Ii* − *Si* (2)

= *c*

(*hi*2

)5 [

*f* (*iv*) (*xi* + *hi*4

)

+ *f* (*iv*) (*xi* + 3*hi*4

)]

+···*.*

Each of the terms in the bracket can be expanded in Taylor series about the point (*xi* + *hi/*2):

*f* (*iv*) (*xi* + *hi*4

)

= *f* (*iv*) (*xi* + *hi*2

)

− *hi*4 *f* (*v*) (*xi* + *hi*2

)

+···

*f* (*iv*) (*xi* + 3*hi*4

)

= *f* (*iv*) (*xi* + *hi*2

)

+ *hi*4 *f* (*v*) (*xi* + *hi*2

)

+···*.*

Thus,

*Ii* − *Si* (2)

= 2*c*

(*hi*2

)5 [

*f* (*iv*) (*xi* + *hi*2

)]

+···*.* (3.20)

Subtracting (3.19) from (3.20), *Ii* drops out and we obtain

*Si* (2)

− *Si* = 1516*ch*5*i f* (*iv*) (*xi* + *hi*2

)

+···*.*

This is the key result, it states that 1*/*15 of the difference between *Si* the error in and *Si* (2)

. The *Si* (2)

, as given by (3.20), is about good news is that this difference

3.5 ADAPTIVE QUADRATURE 39

can be computed; it is simply the difference between two numerical estimates of the integral that we have already computed.

If the user-specified error tolerance for the entire interval is *ε*, the weighted tolerance for the interval [*xi,xi*+1] is*hi*

*b* − *aε.*

Thus, the adaptive algorithm proceeds as follows: If

115

∣∣∣*Si* (2)

− *Si*∣∣∣ ≤ *hi*

*b* − *aε,* (3.21)

then *Si* (2)

is sufficiently accurate for the interval [*xi,xi*+1], and we move on to the next interval. If condition (3.21) is not satisfied, the interval will be subdivided further.

This is the essence of adaptive quadrature programs. Similar methodology can be devised when other base methods such as the trapezoidal rule are used (Exercise 14). As with the Richardson extrapolation, the knowledge of the truncation error can be used to obtain estimates for the accuracy of the numerical solution *without* knowing the exact solution.

**EXAMPLE 3.3 Adaptive Quadrature**

Consider the function

*f* (*x*) = 10*e*−50|*x*| − (*x* − 0*.*5)0*.*01

2 + 0*.*001 + 5 sin(5*x*)*.*

The integral

*I* =

∫ 1−1 *f*(*x*)*dx*

has the exact value of −0.56681975015. When evaluated using the adap- tive quadrature routine QUANC8*†* (quad1 in MATLAB), with various error tolerances *ε*, the following values are obtained.

***ε* Integral** 10−2 −0.45280954 10−3 −0.53238036 10−4 −0.56779547 10−5 −0.56681371 10−6 −0.56681977 10−7 −0.56681974

*†* G. E. Forsythe, M. A. Malcolm, and C. B. Moler (1977), *Computer Methods for Mathematical Computations.* Englewood Cliffs, N.J.: Prentice Hall. QUANC8 is available on the World Wide Web; check, for example, http://www.netlib.org/.

40 NUMERICAL INTEGRATION

10

50−5quadrature points *f*(*x*)

−10−1 −0.5 0 *x* 0.5 1 **Figure 3.3** Distribution of adaptive quadrature points for the function in Example 3.3.

The quadrature points for the case *ε* = 10−5 are shown along with the function *f*(*x*) in Figure 3.3. Note how the algorithm puts more points in regions where greater resolution was needed for evaluation of the integral.

**3.6 Gauss Quadrature**

Recall that any quadrature formula can be written as

*I* =

∫ *ba f* (*x*)*dx* =

∑*ni*=0

*wi f* (*xi*)*.* (3.22)

If the function *f* is given analytically, we have two important choices to make. We have to select the location of the points *xi* and the weights *wi*. The main concept underpinning Gauss quadrature is to make these choices for optimal accuracy; the criterion for accuracy being the highest degree polynomial that can be integrated exactly. You can easily verify that the trapezoidal rule in- tegrates a straight line exactly and Simpson’s rule integrates a cubic exactly (see Exercise 5). As we will show below, Gauss quadrature integrates a polyno- mial of degree 2*n* + 1 exactly using only *n* + 1 points, which is a remarkable achievement!

Let *f* be a polynomial of degree 2*n* + 1. Suppose we represent *f* by an *n*th- order Lagrange polynomial, *P.* Let *x*0*, x*1*, x*2*,...,xn* be the points on the *x*-axis where the function *f* is evaluated. Using Lagrange interpolation, we have:

*P*(*x*) =

∑*nj*=0

*f* (*x j*)*L*(*n*)

*j* (*x*)*.* (3.23)

This representation is exact if *f* were a polynomial of degree*n.* Let *F* be a poly- nomial of degree *n* + 1 with *x*0*, x*1*,...,xn* as its roots,

*F*(*x*) = (*x* − *x*0)(*x* − *x*1)(*x* − *x*2)···(*x* − *xn*)*.*

3.6 GAUSS QUADRATURE 41

The difference *f* (*x*) − *P*(*x*) is a polynomial of degree 2*n* + 1 that vanishes at *x*0*,x*1*,...,xn* because *P* was constructed to pass through *f* (*x*0)*, f* (*x*1)*,..., f* (*xn*) at the points*x*0*,x*1*,..., xn*. Thus, we can write the difference *f* (*x*) − *P*(*x*) in the following form:

*f* (*x*) − *P*(*x*) = *F*(*x*)

∑*nl*=0

*qlxl.*

Integrating this equation results in

∫

*f* (*x*)*dx* =∫

*P*(*x*)*dx* +

∫

*F*(*x*)

∑*nl*=0

*qlxldx.*

Suppose we demand ∫

that *F*(*x*)*xαdx* = 0 *α* = 0*,*1*,*2*,*3*,...,n.* (3.24)

In principle we can choose *x*0*,x*1*,x*2*,..., xn* such that these *n* + 1 conditions are satisfied. Choosing the abscissa in this manner leads to the following ex- pression for the integral:

∫

*f* (*x*)*dx* =

∫

*P*(*x*)*dx* =

∑*nj*=0

*f* (*x j*)*wj,*

where

*wj* =

∫

*L*(*n*)

*j* (*x*)*dx* (3.25)

are the weights.

According to (3.24), *F* is a polynomial of degree *n* + 1 that is orthogonal to all polynomials of degree less than or equal to *n.* Points *x*0*,x*1*,..., xn* are the zeros of this polynomial. These polynomials are called Legendre polynomials when *x* varies between –1 and 1. They are orthonormal, that is

∫ 1−1 *Fn* (*x*) *Fm* (*x*)*dx* = *δnm* where

*δnm* =

{0 if *m* = *n*

1 if *m* = *n,*

and *Fn* is the Legendre polynomial of degree *n.* Their zeros are documented in mathematical tables (see *Handbook of Mathematical Functions*, by Abramowitz & Stegun) or in canned programs (see for example, *Numerical Recipes* by Press et al. or MATLAB). Having the zeros, the weights *wj* can be readily computed, and they are also documented in the Gauss quadrature tables or obtained from canned programs. Many numerical analysis software libraries contain Gauss quadrature integration subroutines.

42 NUMERICAL INTEGRATION

Note that we can always transform the interval *a* ≤ *x* ≤ *b* into −1 ≤ *ξ* ≤ 1 by the transformation

*x* = *b* + 2 *a*

+ *b* − 2 *a*

*ξ.*

Typically, to use Gauss–Legendre quadrature tables to evaluate the integral

∫ *ba f* (*x*)*dx,* one first changes the independent variable to *ξ* and obtains the weights *wi* and the points on the abscissa, *ξ*0*,ξ*2*,...,ξn* from the tables (for the chosen *n*). The integral is then approximated by

*b* − *a* 2

∑*n*(*b* + 2 *a*

+ *b* − 2 *a ξj*)*f*

*wj.* (3.26) *j*=0

Note that in the tables in Abramowitz & Stegun,*n* denotes the number of points, not *n* + 1.

**EXAMPLE 3.4 Integration Using Gauss–Legendre Quadrature**

Consider the integral

∫ 81

log *x x*

*dx.*

The exact value is 12

(log 8)2 = 2*.*1620386*.* Suppose we evaluate this inte- gral with five points using the Gauss–Legendre quadrature. The subroutine gauleg in *Numerical Recipes* (gauss leg in MATLAB) gives the following points and weights in the interval, 1 ≤ *x* ≤ 8:

***i xi wi*** 1 1.3283706 0.8292441 2 2.6153574 1.6752003 3 4.5000000 1.9911112 4 6.3846426 1.6752003 5 7.6716294 0.8292441

Substituting these values into (3.26) results in the numerical estimate for the integral, *I* ≈ 2*.*165234*.* The corresponding error is *ε* = 0*.*0032 (0*.*15%) which is much better than the performance of the Simpson’s rule with nine points (eight panels), i.e., *ε* = 0*.*013 (0*.*6%)*.* Gauss quadrature with nine points would result in *ε* = 0*.*000011 (0*.*05%)*.*

There are several Gauss quadrature procedures corresponding to other or- thogonal polynomials. These polynomials are distinguished by the weight

3.6 GAUSS QUADRATURE 43

functions, *W*, used in their statement of orthogonality: ∫ *ba Pm* (*x*)*Pn* (*x*)*W*(*x*)*dx* = *δmn* (3.27) and the range [*a*, *b*] over which the functions are orthogonal. For example, Hermite polynomials are ∫ +∞

orthogonal according to −∞ *e*−*x*2*Hm* (*x*)*Hn* (*x*)*dx* = *δmn.* The Gauss–Hermite quadrature can be used to evaluate integrals of the form

*I* =

∫ +∞

−∞ *e*−*x*2 *f* (*x*)*dx* ≈

∑*ni*=0

*wi f* (*xi*)*.* (3.28)

This should lead to accurate results provided that *f* grows slower than *ex*2 as |*x*| approaches infinity.

**EXAMPLE 3.5 Gauss Quadrature Based on Hermite Polynomials**

Consider the integral

*I* =

∫ +∞

−∞ *e*−*x*2 cos*xdx.* The exact value is 1.38038845. Suppose we use the Gauss–Hermite quadra- ture to evaluate the integral using seven nodes. A call to the gauher FORTRAN subroutine in *Numerical Recipes* (gauss her in MATLAB) produces the following abscissa and weights:

***i xi wi*** 1 2.6519613 0.0009718 2 1.6735517 0.0545156 3 0.8162879 0.4256073 4 0.0000000 0.8102646 5 −0.8162879 0.4256073 6 −1.6735517 0.0545156 7 −2.6519613 0.0009718

Note that the weights rapidly vanish at higher values of |*x*|, this is probably why no more points are needed beyond |*x*| = 2.652. Substituting these val- ues into (3.28) results in *I* ≈ 1.38038850, which is in excellent agreement with the exact value.

Although Gauss quadrature is very powerful, it may not be cost effective for solution improvement. One improves the accuracy by adding additional points, which would involve additional function evaluations. Function evaluations are the major portion of the computational cost in numerical integration. In the case of Gauss quadrature, the new grid points generally do not include the old ones and therefore one needs to perform a complete new set of function evaluations.

44 NUMERICAL INTEGRATION

In contrast, adaptive techniques and the Romberg integration do not discard the previous function evaluations but use them to improve the solution accuracy when additional points are added.**EXERCISES**

1. What is the relation between the fourth-order central Pad ́e scheme for differ- entiation and Simpson’s rule for integration? How can you use Simpson’s rule to *Hint*: derive Start the with fourth-order ∫ *xxi*−1 *i*+1

Pad ́e scheme? *f* (*x*)*dx.* 2. Show that

*N*−1∑*i*=1

*ui δx*

*δv*∣∣∣∣*i* = −

∑*N*−1*vi i*=1

∣∣∣∣*i* + boundary terms*.*

What are the boundary terms? Compare this discrete expression to the rule of integration by parts. 3. Using the error analysis for the trapezoidal and rectangle rules, show that Simpson’s rule for integration over the entire interval is fourth-order accurate. 4. Explain why in Example 3.1, the trapezoidal rule with end-correction is slightly

more accurate than the Simpson’s rule. 5. Explain why the rectangle and trapezoidal rules can integrate a straight line

exactly and the Simpson’s rule can integrate a cubic exactly. 6. A common problem of mathematical physics is that of solving the Fredholm

integral equation

*f* (*x*) = *φ*(*x*) +

*δuδx*

∫ *ba K*(*x,t*)*φ*(*t*) *dt,* where the functions *f* (*x*) and *K*(*x,t*) are given and the problem is to obtain *φ*(*x*). (a) Describe a numerical method for solving this equation. (b) Solve the following equation

*φ*(*x*) = *πx*2 +

∫ *π*0 3(0*.*5 sin 3*x* − *tx*2)*φ*(*t*)*dt.* Compare to the exact solution *φ*(*x*) = sin 3*x*. 7. Describe a method for solving the Volterra integral equation

*f* (*x*) = *φ*(*x*) +

∫ *xa K*(*x,t*)*φ*(*t*)*dt.* Note that the upper limit of the integral is *x.* What is *φ*(*a*)? 8. Consider the integral

∫ 10

[ √*x* 100 + *.*01 + (*x* − 0*.*3)1

2 + *.*001 − *π*]*dx.*

EXERCISES 45

(a) Numerically evaluate this integral using the trapezoidal rule with *n* panels of uniform length *h.* Make a log–log plot of the error (%) vs. *n* and discuss the accuracy of the method. Take *n* = 8, 16, 32,*...*. (b) Repeat part (a) using the Simpson’s rule and the trapezoidal rule with

end-correction. (c) Evaluate the integral using an adaptive method with various error toler- ances (you may want to use the *Numerical Recipes* subroutine odeint or MATLAB’s functionquad8). How are the*x*points for function evaluations distributed? Plot the integrand showing the positions of its evaluations on 9. the *x* axis.

Simpson’s rule was used to find the value of the integral *I* results for two different step sizes are given in the table below

= ∫ 0 1*f* (*x*)*dx*. The

*h I* 0.2 12.045 0.1 11.801 Use this information to find a more accurate value of the integral *I*. 10. Use the Richardson extrapolation to compute *f* (1*.*0) and *f* (5*.*0) to five place

accuracy with *f* = (*x* + 0*.*5)−2 *.* Use the central difference formula

*f* (*x*) ≈ *f* (*x* + *h*) − *f* (*x* − *h*)

2*h* and take the initial step size,*ho* = 0.5. Comment on the reason for the difference in the convergence rates for the two derivatives. 11. Use the Gauss quadrature to integrate:

*I* =

∫ +∞

−∞ *e*−*x*2 cos*αx dx* for *α* = 5*.*The exact solution is *I* = √*πe*−*α*2*/*4*.* The example worked out in the text corresponds to *α* = 1*.* For the present case of *α* = 5, discuss the number of function evaluations required to get the same level of accuracy as in the example. 12. Evaluate:

*I* =

∫ 20

*e*√−*xx dx*

(a) To deal with the singularity of the integrand at x = 0, try the change of

variable *x* = *t*2. (b) Use the midpoint rule to avoid the singularity at x = 0. Compare the two

methods in terms of accuracy and cost. 13. It has been suggested that to evaluate:

*I* =

∫ ∞0 *e*−*x*2*dx*

46 NUMERICAL INTEGRATION

(a) One can truncate the integration range to a finite interval, [0, *R*], such that the integrand is “sufficiently small” at *R* (and bounded by a monotonically decreasing function in the interval [*R,*∞]). Evaluate using *R* = 4. (b) Change the independent the finite value, *I* =

domain √2 *π.*

variable to *t* in *t.* Compare your = results 1+*x* 1and compute the integral over in (a) and (b) with the exact

14. Describe in detail an adaptive quadrature method that uses the trapezoidal rule 15. as We its would basic like integration to calculate scheme. ∫ 0 *π*sin(*x*)*dx*:

Show in detail the error estimate.

(a) Develop a quadrature method based on the cubic spline interpolation. (b) Use this method to calculate the integral using 4, 8, 16, 32 intervals. Show the error versus number of points in a log–log plot. What is the order of accuracy of the method? 16. In this problem, we compare the performance of different integration strategies.

We would like to integrate:

*I* =

∫ +∞

−∞ *f* (*x*) *dx, f* (*x*) = *e*−*x*2 cos(2*x*)*.*

(a) Use nodes. the Compare Gauss–Hermite your answer quadrature with the to exact evaluate value the of the integral integral using (√*π/e*). eight

(b) Use the transformation *ξ* = tanh(*ax*) to map your domain into a finite interval. Reformulate the integral in the new domain. What is the value of the integrand in the limit of *ξ* = ±1? (c) Use 17 points to discretize the *ξ* domain uniformly. Plot *f* (*x*) and show the corresponding location of these points with *a* = 2 and *a* = 0.4. Which value of *a* is more appropriate for mapping? (d) Numerically evaluate the integral obtained in part (b) using the trapezoidal rule with 17 points for *a* = 0.4. What is the error of the integral? Compare your results with the result of Simpson’s rule. Explain why the trapezoidal rule performs better than Simpson’s rule in this case. *Hint*: Plot the inte- grand as a function of *ξ* and note its behavior at the boundaries. 17. Combine a more accurate Simpson’s method rule with for the the trapezoidal integral of rule ∫ values of *f* (*xi* − *h*)*, f* (*xi*)*, f* (*xi* + *h*)*, f* (*xi* with *xx*− *i*−*h i*+*h*

*h*)*, f* end correction (*x*)*dx*. You and *f* (*xi* to obtain may use the + *h*)*.* What is the ∫ *ba* order of accuracy of your scheme? What will be the global scheme for *f* (*x*) *dx* based on this method? 18. *Romberg integration*.

In (3.16), we showed that the extrapolated value *I* ̃12 is fourth-order accurate. This was derived assuming that the coefficients*ci*’s in (3.14) and (3.15) were the same. Strictly speaking, this assumption making this assumption we can show that is *I* ̃12 not is fourth-order correct; however, accurate. even In without (3.15) replace *ci*’s with *c i*, *ci* = *c i*.

FURTHER READING 47

(a) Show that the coefficients c1, c2, . . . , are as follows:

*c*1 = −(*b* 12

− *a*)

∑*n*−1*i*=0

*f* (2) *n* (*yi*)

*, c*2 = −(*b* 480

− *a*)

∑*n*−1*i*=0

*f* (4) *n* (*yi*)

*,...,*

where *yi* is the midpoint of the interval [*xi, xi*+1] with width *h*. (b) Similarly, find expressions (2*n* − 1)*,* where the *zj*’s are for the *c* midpoints 1*, c* 2*,...,* of in intervals terms of *z j, j* = 0*,...,* with width *h/*2. That is, *z*2*i* = *yi* − (*h/*4) *, z*2*i*+1 = *yi* + (*h/*4) *, i* = 0*,...,n* − 1*.* (c) Show indeed that fourth-order *c* 1 = *c*1 + accurate. *α*1*h*2*c*2 +···*,* What and hence the extrapolation formula is

is *α*1? *Hint*: Use Taylor series to expand *f* (*z*2*i*+1) and *f* (*z*2*i*) about *yi* and sub- stitute in the expression for *c* 1*.*

**FURTHER READING**

Abramowitz, M., and Stegun, I. *Handbook of Mathematical Functions with For- mulas, Graphs,* Dahlquist, G., and *and* Bj ̈orck, *Mathematical* A.  ̊*Tables*. Dover, 1972.

*Numerical Methods.* Prentice-Hall, 1974, Chapter 7. Ferziger, J. H. *Numerical Methods for Engineering Application*, Second Edition.

Wiley, 1998, Chapter 3. Forsythe, G. E., Malcolm, M. A., and Moler, C. B. *Computer Methods for Mathe-*

*matical Computations.* Prentice-Hall, 1977, Chapter 5. Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes: The Art of Scientific Computing*, Third Edition. Cambridge University Press, 2007, Chapter 4.

4 Numerical Solution of Ordinary Differential Equations

In this chapter we shall consider numerical solution of ordinary differential equations, ODEs. Here we will experience the real power of numerical analysis for engineering applications, as we will be able to tackle some real problems. We will consider both single and systems of differential equations. Since high- order ODEs can be converted to a system of first-order differential equations, our concentration will be on first-order ODEs. The extension to systems will be straightforward. We will consider all classes of ordinary differential equa- tions: *initial*, *boundary* and *eigenvalue* problems. However, we will emphasize techniques for initial value problems because they are used extensively as the basis of methods for the other types of differential equations. The material in this chapter constitutes the core of this first course in numerical analysis; as we shall see in Chapter 5, numerical methods for partial differential equations are rooted in the methods for ODEs.

**4.1 Initial Value Problems**

Consider the first-order ordinary differential equation

*dydt* = *f* (*y,t*) *y*(0) = *y*0*.* (4.1)

We would like to find *y*(*t*) for 0 *< t* ≤ *t f*. The aim of all numerical methods for solution of this differential equation is to obtain the solution at time *tn*+1 = *tn* + *t*, given the solution for 0 ≤ *t* ≤ *tn*. This process, of course, continues; i.e., once *yn*+1 = *y*(*tn*+1) is obtained, then *yn*+2 is calculated and so on until the final time, *t f*.

We begin by considering the so-called Taylor series methods. Let’s expand the solution at *tn*+1 about the solution at *tn*

*yn*+1 = *yn* + *hy n* + *h*22 *y n* + *h*36 *y n* +··· (4.2)

48

4.1 INITIAL VALUE PROBLEMS 49

where *h* = *t*. From the differential equation (4.1), we have

*y n* = *f* ( *yn,tn*) which can be substituted in the second term in (4.2). We can, in principle, stop at this point, drop the higher order terms in (4.2), and get a second-order approximation to *yn*+1 using *yn*. To get higher order approximations to *yn*+1, we need to evaluate the higher order derivatives in (4.2) in terms of the known quantities at *t* = *tn*. We will use the chain rule to obtain

*y* = *dy dt* = *dfdt* = *∂f∂t* + *∂f∂y*

*dydt*

= *ft* + *f fy*

*y* = *∂∂t* [ *ft* + *f fy*] + *∂∂y*[ *ft* + *f fy*] *f*

= *ftt* + 2 *f fyt* + *ft fy* + *f f y* 2+ *f* 2*fyy.*

Since *f* is a known function of *y* and *t*, all the above partial derivatives can, in principle, be computed. However, it is clear that the number of terms increases rapidly, and the method is not very practical for higher than third order.

The method based on the first two terms in the expansion is called the *Euler method*:

*yn*+1 = *yn* + *hf* ( *yn,tn*)*.* (4.3)

In using the Euler method, one simply starts from the initial condition, *y*0, and marches forward using this formula to obtain *y*1*, y*2*,...*. We will study the properties of this method extensively as it is a very simple method to analyze. From the Taylor series expansion it is apparent that the Euler method is second- order accurate for one time step. That is, if the exact solution is known at time step *n*, the numerical solution at time step *n* + 1 is second-order accurate. However, as with the quadrature formulas, in multi-step calculations, the errors accumulate, and the global error for advancing from the initial condition to the final time *t f* is only *first-order* accurate.

Among the more accurate methods that we will discuss are the *Runge–Kutta* formulas. With explicit Runge–Kutta methods the solution at time step *tn*+1 is obtained in terms of *yn, f* ( *yn,tn*), and *f* ( *y,t*) evaluated at the intermediate steps between *tn* and *tn*+1 = *tn* + *t* (not including *tn*+1). The higher accuracy is achieved because more information about *f* is provided due to the interme- diate evaluations of *f* . This is in contrast to the Taylor series method where we provided more information about *f* through the higher derivatives of *f* at *tn*.

50 NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

Higher accuracy can also be obtained by providing information about *f* at times *t < tn*. That is, the corresponding formulas involve *yn*−1*, yn*−2*,...,* and *fn*−1*, fn*−2*,...*. These methods are called *multi-step* methods.

We will also distinguish between *explicit* and *implicit* methods. The preced- ing methods were all explicit. The formulas that involve *f* ( *y,t*) evaluated at *yn*+1*, tn*+1 belong to the class of implicit methods. Since *f* may be a non-linear function of *y*, to obtain the solution at each time step, implicit methods usually require solution of non-linear algebraic equations. Although the computational cost per time step is higher, implicit methods offer the advantage of numerical stability, which we shall discuss next.

**4.2 Numerical Stability**

So far, in the previous chapters, we have been concerned only with the accuracy of numerical methods and the work required to implement them. In this section the concept of numerical stability in numerical analysis is introduced, which is a more critical property of numerical methods for solving differential equations. It is quite possible for the numerical solution to a differential equation to grow unbounded even though its exact solution is well behaved. Of course, there are cases for which the exact solution grows unbounded, but for our discussion of stability, we shall concentrate only on the cases in which the exact solution is bounded. Given a differential equation

*y* = *f* ( *y,t*) (4.1)

and a numerical method, in stability analysis we seek the conditions in terms of the parameters of the numerical method (mainly the step size *h*) for which the numerical solution remains bounded. In this context we have three classes of numerical methods:

*Stable numerical scheme*: Numerical solution does not grow unbounded (blow up) with any choice of parameters such as the step size. We will have to see what the cost is for such robustness. *Unstable numerical scheme*: Numerical solution blows up with any choice of parameters. Clearly, no matter how accurate they may be, such numerical schemes would not be useful. *Conditionally stable numerical scheme*: With certain choices of parameters the numerical solution remains bounded. Hopefully, the cost of the calculation does not become prohibitively large.

We would apply the so-called *stability analysis* to a numerical method to deter- mine its stability properties, i.e., to determine to which of the above categories the method belongs. The analysis is performed for a simpler equation than (4.1), which hopefully retains some of the features of the general equation. Consider

4.2 NUMERICAL STABILITY 51

the two-dimensional Taylor series expansion of *f* ( *y,t*): *f* ( *y,t*) = *f* ( *y*0*,t*0) + (*t* − *t*0)*∂f∂t* ( *y*0*,t*0) + ( *y*−*y*0)*∂f∂y*( *y*0*,t*0)

+ 2!

1[(*t* − *t*0)2*∂∂t*2*f*

2 + 2(*t* − *t*0)( *y* − *y*0) *∂t∂y ∂*2*f*

+ ( *y* − *y*0)2*∂∂y*2*f*

2]

+···*.*

Collecting only the linear terms and substituting in (4.1), we formally get

*y* = *λy* + *α*1 + *α*2*t* +··· (4.4) where *λ,α*1*,α*2 are constants. For example,

*λ* = *∂f∂y*( *y*0*,t*0)*.* Discarding the non-linear terms (those involving higher powers of ( *y* − *y*0), (*t* − *t*0) or their product) on the right-hand side of (4.4) yields the linearization of (4.1) about ( *y*0*,t*0). For convenience and feasibility of analytical treatment, stability analysis is usually performed on the *model problem*, consisting of only the first term on the right-hand side of (4.4),

*y* = *λy,* (4.5)

instead of the general problem (4.1). Here, *λ* is a constant. It turns out that the inhomogeneous terms in the linearized equation (4.4) do not significantly affect the results of the stability analysis (see Exercise 10). Note that the model equation has an exponential solution, which is the most dangerous part of the full solution of (4.1).

In our treatment of (4.5), we will allow *λ* to be complex

*λ* = *λR* + *iλI* with the real part *λR* ≤ 0 to ensure that the solution does not grow with *t*. This generalization will allow us to readily apply the results of our analysis to systems of ordinary differential equations and partial differential equations. To illustrate this point, consider the second-order differential equation

*y* + *ω*2*y* = 0*.*

The exact solution is sinusoidal

*y* = *c*1 cos*ωt* + *c*2 sin*ωt.* We can convert this second-order equation to two first-order equations

[ *y*1*y*2

]

=

[ 0−*ω*2 10][ *y*1*y*2

]*.*

The eigenvalues of the 2 × 2 matrix *A*,

*A* =

[ 0−*ω*2 10]*,*

52 NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

are *λ* = ±*iω*. Diagonalizing *A* with the matrix of its eigenvectors *S*,

*A* = *S S*−1*,*

leads to the uncoupled set of equations

***z*** = ***z****,*

where

***z*** = *S*−1

( *y*1*y*2)

and is the diagonal matrix with eigenvalues of *A* on the diagonal. The differ- ential equations for the components of ***z*** are

*z* 1 = *iωz*1 *z* 2 = −*iωz*2*.*

This simple example illustrates that higher order linear differential equations or systems of first-order linear differential equations can reduce to uncoupled ordinary differential equations of the form of (4.5) with complex coefficients. The imaginary part of the coefficient results in oscillatory solutions of the form *e*±*iωt*, and the real part dictates whether the solution grows or decays. For our stability analysis *we will be concerned only with cases where λ has a zero or negative real part*.**4.3 Stability Analysis for the Euler Method**

Applying the Euler method (4.3),

*yn*+1 = *yn* + *hf* ( *yn,tn*)*,*

to the model problem (4.5) leads to

*yn*+1 = *yn* + *λhyn*

= *yn*(1 + *λh*)*.*

Thus, the solution at time step *n* can be written as

*yn* = *y*0(1 + *λh*)*n.* (4.6)

For complex *λ*, we have*yn* = *y*0(1 + *λRh* + *iλIh*)*n* = *y*0*σn,*

where *σ* = (1 + *λRh* + *iλIh*) is called the amplification factor. The numerical solution is stable (i.e., remains bounded as *n* becomes large) if

|*σ*| ≤ 1*.* (4.7)

Region of stability for the exact solution

Im(λh)

Re(λh)

**Figure 4.1** Stability diagram for the exact solution in the *λRh*− *λI h* plane.

Note that for *λR* ≤ 0 (which is the only case we consider) the exact solution, *y*0*eλt*, decays. That is, in the (*λRh* − *λIh*) plane, the region of stability of the *exact solution* is the left-hand plane as illustrated in Figure 4.1.

However, only a portion of this plane is the region of stability for the Euler method. This portion is inside the circle

|*σ*|2 = (1 + *λRh*)2 + *λ*2*Ih*2 = 1*.* (4.8)

For any value of *λh* in the left-hand plane and outside this circle the numerical solution blows up while the exact solution decays (see Figure 4.2). Thus, the Euler method is *conditionally stable*. To have a stable numerical solution, we must reduce the step size *h* so that *λh* falls within the circle. If *λ* is *real* (and negative), then the maximum step size for stability is 2*/*|*λ*|. That is, to get a stable solution, we must limit the step size to

*h* ≤ |*λ*|2*.* (4.9)

Note that for real (and negative) *λ*, (4.7) is enforced for *λh* as low as −2. The main consequence of this limitation on*h*is that it would require more time steps, and hence more work, to reach the final time of integration, *t f* . The circle (4.8)

λRegion of stability Ih for Explicit Euler

−2.0 λRh

**Figure 4.2** Stability diagram for the explicit Euler method.

4.3 STABILITY ANALYSIS FOR THE EULER METHOD 53

54 NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

is only tangent to the imaginary axis. Therefore, the Euler method is always unstable (irrespective of the step size) for *purely imaginary λ*. If *λ* is real and the numerical solution is unstable, then we must have

|1 + *λh*| *>* 1*,*

which means that (1 + *λh*) is negative with magnitude greater than 1. Since

*yn* = (1 + *λh*)*ny*0*,*

the numerical solution exhibits oscillations with change of sign at every time step. This oscillatory behavior of the numerical solution is usually a good indi- cation of numerical instability.

**EXAMPLE 4.1 Explicit Euler**

We will solve the following ODE using the Euler method:

*y* + 0*.*5*y* = 0 *y*(0) = 1 0 ≤ *t* ≤ 20*.*

Here *λ* is real and negative. The stability analysis of this section indicates that the Euler method should be stable for *h* ≤ 4. The solution is advanced by

*yn*+1 = *yn* − 0*.*5*hyn*

and the results for stable (*h* = 1.0) and unstable (*h* = 4.2) solutions are pre- sented in Figure 4.3. We see that the solution with *h* = 4.2 is indeed unstable. Also note the oscillatory behavior of the solution before blow-up.

1.5

Explicit Euler, *h* = 1 Explicit Euler, *h* = 4.2 Exact

1

0.5*y* 0

−0.5−1

−1.50 2 4 6 8 10 12 14 16 18 20 *t*

**Figure 4.3** Numerical solution of the ODE in Example 4.1 using the Euler method.

4.4 IMPLICIT OR BACKWARD EULER 55

**4.4 Implicit or Backward Euler**

The implicit Euler scheme is given by the following formula:

*yn*+1 = *yn* + *hf* ( *yn*+1*,tn*+1)*.* (4.10)

Note that in contrast to the explicit Euler, the implicit Euler does not allow us to easily obtain the solution at the next time step. If *f* is non-linear, we must solve a non-linear algebraic equation at each time step to obtain *yn*+1, which usually requires an iterative algorithm. Therefore, the computational cost per time step for this scheme is, apparently, much higher than that for the explicit Euler. However, as we shall see below, the implicit Euler method has a much better stability property. Moreover, Section 4.7 will show that at each step, the requirement for an iterative algorithm may be avoided by the linearization technique.

Applying the backward Euler scheme to the model equation (4.5), we obtain

*yn*+1 = *yn* + *λhyn*+1*.*

Solving for *yn*+1 produces

*yn*+1 = (1 − *λh*)−1*yn*

or

*yn* = *σ ny*0*,*

where

*σ* = 1 − 1

*λh.* Considering complex *λ*, we have

*σ* = (1 − *λRh*) 1

− *iλIh.* The denominator is a complex number and can be written as the product of its modulus and phase factor,

*σ* = *Ae*1*iθ ,* where

*A* =

√(1 − *λRh*)2 + *λ*2*Ih*2*, θ* = −tan−1 1 − *λIh*

*λRh.* For stability, the modulus of *σ* must be less than or equal to 1; i.e.,

|*σ*| = |*e*−*iθA* |

= *A* 1≤ 1*.*

56 NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

This is always true because*λR* is negative and hence *A >* 1. Thus, the backward Euler scheme is unconditionally stable. Unconditional stability is the usual characteristic of implicit methods. However, the price is higher computational cost per time step for having to solve a non-linear equation.

It should be pointed out that one can construct conditionally stable implicit methods. Obviously, such methods are not very popular because of the higher cost per step without the benefit of unconditional stability. Also note that nu- merical stability does not necessarily imply accuracy. A method can be stable but inaccurate. From the stability point of view, our objective is to use the max- imum step size *h* to reach the final destination at time *t* = *t f*. Large time steps translate to a lower number of function evaluations and lower computational cost. Large time steps may not be optimum for acceptable accuracy, but are strived for from the stability point of view.

**EXAMPLE 4.2 Implicit (Backward) Euler**

We now solve the ODE of Example 4.1 using the implicit Euler method. The stability analysis for the implicit Euler indicated that the numerical solution should be unconditionally stable. The solution is advanced by

*yn*+1 = 1 + *y*0*.*5*h*

*n*

and the results for *h* = 1.0 and *h* = 4.2 are presented in Figure 4.4. Both solutions are now seen to be stable, as expected. The solution with *h* = 1.0 is more accurate. Note that the usual difficulty in obtaining the solution at each time step inherent with implicit methods is not encountered here because the differential equation in this example is linear.

**1.0**

**Implicit Euler (h = 1.0) Exact ) t(y0.5**

**t**

**Implicit Euler (h = 4.2)**

**0**

**0 5 10 15 20**

**Figure 4.4** Numerical solution of the ODE in Example 4.2 using the implicit Euler method.

**4.5 Numerical Accuracy Revisited**

We have shown that the numerical solution to the model problem

*y* = *λy* (4.5)

4.5 NUMERICAL ACCURACY REVISITED 57

is of the form

*yn* = *y*0*σn.* (4.11)

The exact solution is

*y*(*t*) = *y*0*eλt* = *y*0*eλnh* = *y*0(*eλh*)*n.* (4.12)

In analogy with the modified wavenumber approach of Chapter 2, one can often determine the order of accuracy of a method by comparing the numerical and exact solutions for a model problem, i.e., (4.11) and (4.12). That is, we compare the amplification factor *σ* with

*eλh* = 1 + *λh* + *λ*2*h*2

2 + *λ*3*h*3

6 +···*.* For example, the amplification factor of the explicit Euler is

*σ* = 1 + *λh,*

and the amplification factor for the backward Euler is

*σ* = 1

(1 − *λh*) = 1 + *λh* + *λ*2*h*2 + *λ*3*h*3 +···*.* Thus, both methods are able to reproduce only up to the *λh* term in the exponential expansion. Each method is second-order accurate for one time step, but globally first order. From now on, we will call a method *α*th order if its amplification factor matches all the terms up to and including the *λαhα/α*! term in the exponential expansion. The order of accuracy derived in this manner from the linear analysis (i.e., from application to (4.5)) should be viewed as the upper limit on the order of accuracy. A method may have a lower order of accuracy for non-linear equations.

Often the order of accuracy by itself is not very informative. In particular, in problems with oscillatory solutions, one is interested in the phase and amplitude errors separately. To understand this type of error analysis, we will consider the model equation with pure imaginary *λ*:

*y* = *iωy y*(0) = 1*.*

The exact solution is *eiωt*, which is oscillatory. The frequency of oscillations is *ω* and its amplitude is 1. The numerical solution with the explicit Euler is

*yn* = *σny*0

where *σ* = 1 + *iωh*. It is clear that the amplitude of the numerical solution,

|*σ*| = √1 + *w*2*h*2

is greater than 1, which reconfirms that the Euler method is unstable for purely imaginary *λ.σ* is a complex number and can be written as

*σ* = |*σ*|*eiθ,*

58 NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

y

Phase lag

Amplitude error

t

Exact Solution Numerical Solution

**Figure 4.5** A schematic showing the amplitude and phase errors in the numerical solution.

where

*θ* = tan−1 *ωh* = tan −1Im(*σ*)

Re(*σ*)*.*

A measure of the phase error (PE) (see Figure 4.5) is obtained from comparison with the phase of the exact solution

PE = *ωh* − *θ* = *ωh* − tan−1 *ωh.*

Using the power series for tan–1,

tan−1 *ωh* = *ωh* − (*ωh*)3 3

+ (*ωh*)5 5

− (*ωh*)7 7

+···

we have

PE =(*ωh*)3 3

+···*,* (4.13)

which corresponds to a phase lag. This is the phase error encountered at each step. The phase error after *n* time steps is *n*PE.

**4.6 Trapezoidal Method**

The formal solution to the differential equation (4.1) with the condition *y*(*tn*) = *yn* is

*y*(*t*) = *yn* +

∫ *t*

*tn f* ( *y,t* )*dt .* At *t* = *tn*+1

*yn*+1 = *yn* +

∫ *tn*+1

*tn f* ( *y,t* )*dt .*

4.6 TRAPEZOIDAL METHOD 59

Approximating the integral with the trapezoidal method leads to

*yn*+1 = *yn* + *h*2[ *f* ( *yn*+1*,tn*+1) + *f* ( *yn,tn*)]*.* (4.14) This is the trapezoidal method for the solution of ordinary differential equations. When applied to certain partial differential equations it is often called the Crank– Nicolson method. Clearly the trapezoidal method is an implicit scheme.

Applying the trapezoidal method to the model equation yields

*yn*+1 − *yn* = *h*2[*λyn*+1 + *λyn*] or

*yn*+1 = 1 + *λh*2 1 − *λh*2

*yn.*

Expanding the amplification factor *σ* leads to

*σ* = 1 + *λh*2 1 − *λh*2

= 1 + *λh* + *λ*22 *h*2

+ *λ*34 *h*3

+···

which indicates that the method is *second-order* accurate. The extra accuracy is obtained at virtually no extra cost over the backward Euler method.

Now, we will examine the stability properties of the trapezoidal method by computing the modulus of *σ* for complex *λ* = *λR* + *iλI*. The amplification factor becomes

*σ* = 1 + *λRh*2 1 − *λRh*2 + *i λ*2 *I h*− *i λ*2

*I h.*

Both the numerator and denominator are complex and can be written as *Aeiθ* and *Beiα,* respectively, where

*A* =

√(1 + *λRh*2

)2

+ *λ*2*Ih*2

4

and

*B* =

√(1 − *λ*2

*Rh*)2

+ *λ*2*I*4 *h*2

*.*

Thus,

*σ* = *BAei*(*θ*−*α*) or

|*σ*| = *BA.*

60 NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

Since we are only interested in cases where *λR <* 0, and for these cases *A < B*, it follows that

|*σ*| *<* 1*.*

Thus, the trapezoidal method is unconditionally stable, which is expected since it is an *implicit* method. Note, however, that for real and negative *λ*,

*h*→∞lim *σ* = −1*,* which implies that for large time steps, the numerical solution *σny*0 oscillates between *y*0 and –*y*0 from one time step to the next, but the solution will not blow up.

Let us examine the accuracy of the trapezoidal method for oscillatory solu- tions, *λ* = *iω.* In this case (*λR* = 0)*, A* = *B,* and

|*σ*| = 1*.*

Thus, there is *no amplitude error* associated with the trapezoidal method. Since

*σ* = *e*2*iθ θ* = tan−1 (*ωh*2

)*,*

the phase error is given by

PE = *ωh* − 2 tan−1 *ωh*2 = *ωh* − 2

[*ωh*2 − (*ωh*)24 3

] +···= (*ωh*)12 3

+···

which is about four times better than that for the explicit Euler but of the same order of accuracy.**EXAMPLE 4.3 A Second-Order Equation**

We now consider the second-order equation

*y* + *ω*2*y* = 0 *t >* 0 *y*(0) = *yo y* (0) = 0*,*

and investigate the numerical solutions by the explicit Euler, implicit Eu- ler, and trapezoidal methods. In Section 4.2 it was demonstrated how this equation could be reduced to a coupled pair of first-order equations:

*y* 1 = *y*2 *y* 2 = −*ω*2*y*1*.*

In matrix form we have [ *y*1*y*2

]

=

[ 0 1

−*ω*2 0][ *y*1*y*2

]*.*

These equations were then decoupled, giving

*z* 1 = *iωz*1 *z* 2 = −*iωz*2*.*

The stability of the numerical solution depends upon the eigenvalues *iω* and −*iω* that decouple the system. We see that here the eigenvalues are

4.6 TRAPEZOIDAL METHOD 61

2.5

0 1 2 3 4 5 6 *t*

Explicit Euler

2

Implicit Euler Trapezoidal 1.5Exact

1

0.5*y* 0

−0.5−1

−1.5−2

−2.5**Figure 4.6** Numerical solution of the ODE in Example 4.3.

imaginary and therefore predict the Euler solution to be unconditionally unstable. We have also seen that both backward Euler and trapezoidal meth- ods are unconditionally stable. We will show this to be the case by numerical simulation of the equations. Solution advancement proceeds as follows. For explicit Euler:

[ *y*1*y*2

]*n*+1

[ 1 *h*

−*ω*2*h* 1][ *y*1*y*2 ]=

*. n*For implicit Euler:

[ 1 −*h ω*2*h* 1

][ *y* 1*y* 2

]*n*+1

[ *y*1*y*2

]*n.*

For trapezoidal:[ 1 *ω*2 *h*2 − 1

*h*2 = ][ *y*1*y*2

]*n*+1

[ 1 −*ω*2 *h*2 *h*1

2 ][ *y*1*y*2 ]=

*. n*

Numerical results are plotted in Figure 4.6 for *yo* = 1, *ω* = 4, and time step *h* = 0*.*15.

We see that the explicit Euler rapidly blows up as expected. The implicit Euler is stable, but decays very rapidly. The trapezoidal method performs the best and has zero amplitude error as predicted in the analysis of Section 4.6; however, its phase error is evident and is increasing as the solution proceeds.

Although the numerical methods used in the previous example were intro- duced in the context of a single differential equation, their application to a sys- tem was a straightforward generalization of the corresponding single equation formulas. It is also important to emphasize that the decoupling of the equations using eigenvalues and eigenvectors was performed solely for the purpose of sta- bility analysis. The equations are never decoupled in actual numerical solutions.

62 NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

**4.7 Linearization for Implicit Methods**

As pointed out in Section 4.4, the difficulty with implicit methods is that, in general, at each time step, they require solving a non-linear algebraic equa- tion, which often require an iterative solution procedure such as the Newton– Raphson method. For non-linear initial value problems, iteration can be avoided by the *linearization technique*. Consider the ordinary differential equation:

*y* = *f* ( *y,t*)*.* (4.1)

Applying the trapezoidal method to this equation yields

*yn*+1 = *yn* + *h*2 [ *f* (*yn*+1*,tn*+1) + *f* (*yn,tn*)] + *O*(*h*3)*.* (4.15) To solve for*yn*+1would require solving a non-linear algebraic equation, and non- linear equations are usually solved by iterative methods. However, by realizing that (4.15) is already an approximate equation (to *O*(*h*3)), it would not make sense to find its solution exactly or to within round-off error. Therefore, we will attempt to solve the non-linear equation (4.15) to *O*(*h*3), which, hopefully, will not require iterations.

Consider the Taylor series expansion of *f* ( *yn*+1*,tn*+1): *f* ( *yn*+1*,tn*+1) = *f* ( *yn,tn*+1) + ( *yn*+1 − *yn*) *∂f∂y*

∣∣∣∣( *yn,tn*+1) + 12 *f* ( *yn*+1 − *yn*)2 *∂*2 *f ∂y*2

∣∣∣∣∣( *yn,tn*+1)

+···*.* (4.16)

But from Taylor series expansion for *y* we have

*yn*+1 − *yn* = *O*(*h*)*.* Therefore, replacing *f* ( *yn*+1*,tn*+1) in (4.15) with the first two terms in its Taylor series expansion does not alter the order of accuracy of (4.15), which (for one step) is *O*(*h*3). Making this substitution results in

*yn*+1 = *yn* + *h*2

[*f* ( *yn,tn*+1) + ( *yn*+1−*yn*)*∂f∂y*

∣∣∣∣( *yn,tn*+1)]+ *f* ( *yn,tn*)+*O*(*h*3)*.*

(4.17)

Rearranging and solving for *yn*+1, yields

*yn*+1 = *yn* + *h*2

*f* ( *yn,tn*+1) + *f* ( *yn,tn*)

1 − *h*2

∣∣( *yn,tn*+1) *.* (4.18) Thus, the solution can proceed without iteration while retaining the global second-order accuracy. Clearly, as far as the *linear* stability analysis is con- cerned, the linearized scheme is also unconditionally stable. However, one should caution that in practice, linearization may lead to some loss of total stability for non-linear *f*.

*∂ f∂y*

4.7 LINEARIZATION FOR IMPLICIT METHODS 63

**EXAMPLE 4.4 Linearization**

We consider the non-linear ordinary differential equation

*y* + *y*(1 − *y*) = 0 *y*(0) = 12

and its numerical solution by the trapezoidal method:

*yn*+1 = *yn* + *h*2

[*yn*+1( *yn*+1 − 1) + *yn*( *yn* − 1)]*.*

This, of course, is a non-linear algebraic equation for *yn*+1. Using the lin- earization method developed in this section, where *f* is now *y*( *y* − 1), we arrive at the following linearized trapezoidal method:

*yn*+1 = *yn* + 1 *hy*− *n*( *h*(*yyn n* − − 1)

12)*.*

Since the non-linearity is quadratic, we may also solve the resulting non- linear algebraic equation directly and compare the direct implicit solution with the linearized solution. The direct implicit solution is given by

*yn*+1 =

( 2*h* + 1) −

√( 2*h* + 1)2 − 4( 2*hyn* + 2 *yn*( *yn* − 1)) *.*

These equations were advanced from time *t* = 0 to *t* = 1. The error in the so- lution at *t* = 1 is plotted in Figure 4.7 versus the number of steps taken. The slopes for both the trapezoidal and linearized trapezoidal methods clearly show a second-order dependence upon number of steps, demonstrating that second-order accuracy is maintained with linearization. The directly solved trapezoidal method is slightly more accurate, but this is a problem-specific phenomenon (for example, the linearized trapezoidal solution for *y* + *y* 2 = 0 yields the *exact* solution for any *h* while the accuracy of the direct implicit solution is dependent on *h*).

**10-1**

**10-2**

**Trapezoidal Linearized Trapezoidal 10-3 r orrE1010-4 -5**

**10-6**

**10-7**

**10-8**

**100 101 102 103 N -- Number of Steps**

**Figure 4.7** Error in the solution of the ODE in Example 4.4.

64 NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

**4.8 Runge–Kutta Methods**

We noted in the Taylor series method, in Section 4.1, that the order of accuracy of a method increases by including more terms in the expansion. The additional terms involve various partial derivatives of *f* ( *y,t*), which provide additional information on*f* at *t* = *tn*. Note that the analytical form of *f* is not transparent to a time-stepping procedure, only numerical data at one or more steps are. There are different methods of providing additional information about *f* . Runge– Kutta (RK) methods introduce points between *tn* and *tn*+1 and evaluate *f* at these intermediate points. The additional function evaluations, of course, result in higher cost per time step; but the accuracy is increased, and as it turns out, better stability properties are also obtained.

We begin by describing the general form of (two stage)*second-order*Runge– Kutta formulas for solving

*y* = *f* ( *y,t*)*.* (4.1)

The solution at time step *tn*+1 is obtained from

*yn*+1 = *yn* + *γ*1*k*1 + *γ*2*k*2*,* (4.19)

where the functions *k*1 and *k*2 are defined sequentially

*k*1 = *hf* ( *yn,tn*) (4.20) *k*2 = *hf* ( *yn* + *βk*1*,tn* + *αh*)*,* (4.21)

and *α*, *β*, *γ*1, *γ*2 are constants to be determined. These constants are determined to ensure the highest order of accuracy for the method. To establish the order of accuracy, consider the Taylor series expansion of *y*(*tn*+1) from Section 4.1:

*yn*+1 = *yn* + *hy n* + *h*22 *y n* +···*.* But

*y n* = *f* ( *yn,tn*)*,*

and using the chain rule, we have already obtained

*y* = *ft* + *f fy,*

where *ft* and *fy* are the partial derivatives of *f* with respect to *t* and *y* respec- tively. Thus,

*yn*+1 = *yn* + *hf* ( *yn,tn*) + *h*22 ( *ftn* + *fn fyn*) +···*.* (4.22) To establish the order of accuracy of the Runge–Kutta method as given by (4.19), we must compare its estimate for *yn*+1 to that of the Taylor series formula (4.22). For this comparison to be useful, we must convert the various terms in these