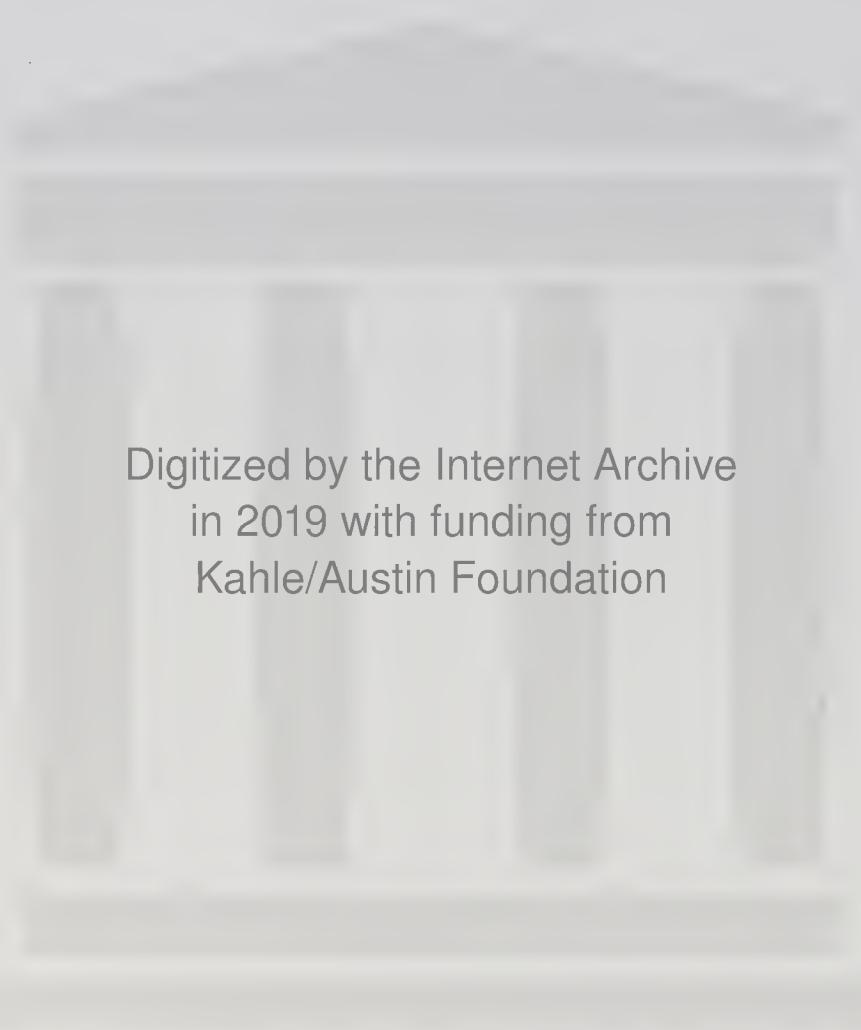


THE NUMERICAL METHOD OF LINES

Integration of Partial Differential Equations

W. E. Schiesser



Digitized by the Internet Archive
in 2019 with funding from
Kahle/Austin Foundation

<https://archive.org/details/numericalmethodo0000schi>

The Numerical _____

Method of Lines _____

The Numerical Method of Lines

**Integration of Partial
Differential Equations**

W.E. Schiesser

Lehigh University
Bethlehem, Pennsylvania
and
Accelerator Division
SSC Laboratory
Dallas, Texas



Academic Press, Inc.

Harcourt Brace Jovanovich, Publishers
San Diego New York Boston
London Sydney Tokyo Toronto

Thomas J. Bata Library
TRENT UNIVERSITY
PETERBOROUGH, ONTARIO

This book is printed on acid-free paper. ☺

Copyright © 1991 by ACADEMIC PRESS, INC.

All Rights Reserved.

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

DISCLAIMER OF WARRANTY AND LIMITS OF LIABILITY:

The author of this book has used his best efforts in preparing this material. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. NO WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, ARE MADE WITH REGARD TO THESE PROGRAMS OR THE DOCUMENTATION CONTAINED IN THIS BOOK, INCLUDING WITHOUT LIMITATION WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. No LIABILITY IS ACCEPTED IN ANY EVENT FOR ANY DAMAGES, INCLUDING INCIDENTAL OR CONSEQUENTIAL DAMAGES, LOST PROFITS, COSTS OF LOST DATA OR PROGRAM MATERIAL, OR OTHERWISE IN CONNECTION WITH OR ARISING OUT OF THE FURNISHING, PERFORMANCE, OR USE OF THE PROGRAMS IN THIS BOOK.

Academic Press, Inc.
San Diego, California 92101

*United Kingdom Edition published by
Academic Press Limited
24-28 Oval Road, London NW1 7DX*

Library of Congress Cataloging-in-Publication Data

Schiesser, W. E.

The numerical method of lines : integration of partial differential equations / W.E. Schiesser.

p. cm.

Includes bibliographical references and index.

ISBN 0-12-624130-9

1. Differential equations, Partial--Numerical solutions.

I. Title.

QA377.S354 1991

515'.353--dc20

90-25389

CIP

PRINTED IN THE UNITED STATES OF AMERICA

91 92 93 94 9 8 7 6 5 4 3 2 1

To Dan
(1964–1985)

Contents

Preface xi

1

What Is the Numerical Method of Lines?

1.1	Why PDEs?	2
1.2	A Typical PDE Derivation	3
1.3	Subscript Notation for PDEs	7
1.4	What Is a Solution to a PDE?	8
1.5	A Brief Introduction to the Numerical Method of Lines	10
1.6	An Error Analysis of the NUMOL Solution	19
1.7	Dirichlet and Neumann Boundary Conditions	24
1.8	Arguments versus COMMON	31
	References	43
	Problems	43

2

Some Applications of the Numerical Method of Lines

2.1	Additional COMMON	45
2.2	Inconsistent Initial and Boundary Conditions	49

2.3	A Nonlinear Problem	54
2.4	A Linear PDE in Cylindrical Coordinates: The Parabolic Fourier Second Law	60
2.5	A Nonlinear PDE in Spherical Coordinates	63
2.6	A PDE Second-Order in Time: The Hyperbolic Wave Equation	70
2.7	Three PDEs Zero-Order in Time: The Elliptic Laplace, Poisson, and Helmholtz Equations	75
2.8	Two Nonlinear PDEs	90
	References	95
	Problems	95

3**Spatial Differentiation**

3.1	Polynomial Approximations	97
3.2	Second-Order Formulas for First Derivatives	98
3.3	Fourth-Order Formulas for First Derivatives	105
3.4	Fourth-Order Formulas for Second Derivatives	111
3.5	First-Order Hyperbolic PDEs	122
	References	141
	Problems	141

4**Initial-Value Integration**

4.1	The ODE Eigenvalue Problem	147
4.2	Stability of Nonlinear ODEs	150
4.3	Stability of the Explicit Euler Method	152
4.4	Stability of the Implicit Euler Method	158
4.5	The BDF Methods	160
4.6	SDRIV2	167
4.7	LSODE, LSODES, and LSODI	171
4.8	DASSL	191
	References	205
	Problems	206

5**Stability of Numerical
Method of Lines Approximations**

5.1	Approximations of the Advection Equation	209
5.2	Approximation of the Heat-Conduction Equation	213

5.3 A System of Nonlinear PDEs with Mapping and Eigenvalue Analysis of the ODE Jacobian Matrix	215
References	244
Problems	244

6

Additional Applications: Multidimensional PDEs and Adaptive Grids

6.1 Classification of PDEs	246
6.2 Burgers' Equation in One Dimension	250
6.3 Burgers' Equation in Two Dimensions	262
6.4 Burgers' Equation in Three Dimensions	273
6.5 Adaptive Grids	287
6.6 Summary and Conclusions	302
References	302
Problems	302

Appendix A The Laplacian Operator in Various Coordinate Systems

305

Appendix B Spatial Differentiation Routines

307

Appendix C Library of ODE and PDE Applications

310

Physics	310
Mathematics	311
Biochemical, Biomedical, and Environmental Systems	313
Separation Systems	314
Kinetics and Reactor Models	315
Heat Transfer	316
Fluid Flow	317
Automatic Control	318

Index

321

Preface

Partial differential equations (PDEs) are among the most widely used forms of mathematics to describe scientific and engineering systems. However, somewhat ironically, writing the PDEs for a particular application is often the easier part of using them; obtaining a solution is the more difficult part. This statement, of course, is rather general and vague, and there are instances when the solution of a PDE model is quite straightforward. However, for most realistic scientific and engineering problems, the PDEs are so numerous and nonlinear as to preclude an analytical solution, and we must therefore resort to numerical methods.

Conceptually, numerical methods are straightforward to understand and use, which, in addition to their generality for the solution of difficult PDE problems, is the basis of their appeal. In practice, the literature describing numerical methods for PDEs can be bewildering and overwhelming, especially for a scientist or engineer who has a problem in PDEs to be solved but has little background in numerical analysis and software for developing a computer solution. The intent of this book is to facilitate getting started in the solution of PDE problems by providing a set of easily understood concepts in numerical methods and the software to implement these methods; the methodology is termed “the numerical method of lines.”

The numerical method of lines is built on the recent advances in the computer solution of ordinary differential equations (ODEs), as extended to PDEs. As we have attempted to demonstrate in this book, it is a remarkably flexible and versatile approach to PDEs, which, at least in

principle, can be applied to all of the major classes of PDEs (elliptic, hyperbolic and parabolic, linear and nonlinear, in one, two, and three spatial dimensions). We have attempted to demonstrate this broad applicability through a series of examples discussed in some detail.

The book does not discuss the theory of the numerical methods; rather, it demonstrates the use of the numerical methods through examples. Also, it is intended only as an introduction. The development of the numerical method of lines has been so rapid during the past 20 years that a detailed discussion of the many applications that have been reported could not be accommodated in one book of reasonable length, so we have not even attempted to survey what has been done. Rather, we hope to provide the reader with enough background and motivation to facilitate further reading of the literature. The software discussed in the book is available from the author as source code on diskettes so that the reader can run any of the applications in the book using only a Fortran 77 compiler, which is probably the best way to fully understand the basic concepts discussed in the book, particularly with regard to the organization and use of the software. Also, a set of some 250 applications of ODEs and PDEs developed by the author are listed in Appendix C. All of these applications are available on diskettes as documented Fortran 77 subroutines.

Many people have contributed to the development of the numerical method of lines during the past 20 years. I have had the privilege of knowing and working with the following people: Fernando Aguirre, Ken Anselmo, Richard Boivin, George Byrne, Ruben Carcagno, Mike Carver, Kathy Chen, Richard Chen, Don Dabdub, Mark Davis, Glenn Dissinger, Bengt Fornberg, Grant Fox, Bill Gear, Alan Hindmarsh, Sam Hu, Mac Hyman, Stan Johnson, David Kahaner, Walter Karplus, Leon Lapidus, Werner Liniger, Al Loeb, Bill Luyben, Neil Madsen, Mike McAslan, Eduardo Meyer, Joe Palen, Linda Petzold, Carl Pirkle, John Rice, Gerry Saidel, Bob Seader, Larry Shampine, Jeng Shih, Cesar Silebi, Fred Stein, Gilbert Stengle, Gilbert Strang, Orhan Tarhan, Skip Thompson, Buddy Watts, Ralph Willoughby, and Jim Zaiser. In particular, I wish to acknowledge the contributions of the following people: Mike Carver for the five-point biased upwind approximations, Alan Hindmarsh for the ODEPACK integrators (LSODE, LSODES, LSODI), Sam Hu for the adaptive grid routines discussed in Chapter 6, David Kahaner for SDRIV2 and DDRIV2, Linda Petzold for DASSL, Larry Shampine and Buddy Watts for RKF45, Cesar Silebi for the humidification column model discussed in Chapter 5, Fred Stein for the reacting sphere model discussed in Chapter 2, and Gilbert Stengle for assistance in the derivation of the second-derivative formulas discussed in Section 3.4. My intention is to appropriately acknowledge the contributors to this book, and I extend sincere apologies for

any oversights. Finally, I welcome any communications from the readers and users of this book; if past experience with such communications is any indication for the future, I will receive totally unexpected questions, suggestions, and applications that are an intellectually rewarding aspect of this work and the basis for new developments in the numerical method of lines.

W. E. SCHIESSER
SSC Laboratory

The SSC Laboratory is operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC02-89ER40486.



What Is the Numerical Method of Lines?

The title of this chapter is a repeat of the same question we have received many times from people who are interested in solving partial differential equations (PDEs). When we have explained the essential features of the method, the responses from the questioners have ranged from “Isn’t that really just a well-established idea” to “I have never heard of this approach.” The purpose of this chapter, and the entire book, is to explain the numerical method of lines (which we shall abbreviate NUMOL to save time with a pronounceable abbreviation), with the goal of providing a computational procedure that can be applied to a broad spectrum of problems in science and engineering. We will assume that the reader might be rather inexperienced in the computer solution of PDEs and is looking for a solution procedure that can be applied with reasonable effort to a PDE problem of current interest; this solution procedure includes the use of the routines discussed in the book (which are available as Fortran 77 source code). We assume that the reader has the usual background in calculus and introductory ordinary differential equations (ODEs), and perhaps possibly for the first time, is confronted with a problem in PDEs. We therefore begin with some basic notions concerning PDEs.

1.1 Why PDEs?

Engineers and scientists spend much of their time acquiring new knowledge and applying their knowledge to the solution of problems. For example, a scientist might study the chemical reactions occurring in the atmosphere that produce air pollution or the circulation patterns in the atmosphere that disperse pollutants. An engineer might use the current knowledge of semiconductors to design a new solid-state device or Newton's equations of motion to design a satellite or space station.

Typically, the acquisition and use of knowledge to understand such complex scientific and engineering problem systems, e.g., the atmosphere or space flight, requires the use of mathematical relations; i.e., physical intuition must be supplemented by *mathematical models (sets of equations)* to predict and explain quantitatively how complex systems behave. Otherwise, we must rely on direct experimentation which may be too expensive or risky (a space station that doesn't function as required, or continuing production and dispersion of pollutants in the atmosphere leading possibly to intolerable health hazards or ecological effects). Accurate mathematical modeling is therefore an indispensable tool for the advance of science and engineering.

Many mathematical forms are used; in particular, differential equations are used in all areas of science and engineering. In this book, we explore computer methods for the NUMOL integration of PDEs that can be applied to a broad spectrum of problems through the use of existing software. The basic concepts of the NUMOL are developed and explained in terms of examples. The emphasis is on applications rather than mathematical properties, and we rely as much on a "let's try this" approach as theory to guide us in putting together NUMOL solutions.

In analyzing the behavior of a problem system, we often want to know how the properties of the system change with time—e.g., how the level of a pollutant changes with time of day. Also, we often wish to know something about the spatial distribution of the property—e.g., how the concentration of the pollutant changes as we move away from the center of the urban area where it is generated. Thus, we have *both time and space as independent variables*. If these independent variables appear in differential equations (*equations containing derivatives*), then we are working with *partial differential equations* (*differential equations with two or more independent variables*). To understand how PDEs naturally arise in time- and space-dependent problems, we consider a classical example.

1.2 A Typical PDE Derivation

We know intuitively that heat flows from regions of higher temperature to regions of lower temperature (e.g., heat is removed from an automobile engine by transferring the heat to the coolant in the radiator). We might wish to know how the temperature in a solid changes with time and position in the solid, perhaps to determine if heat is removed sufficiently fast to avoid thermal damage to the solid (a real concern with automobile engine blocks). If we assume that the transfer of heat is by thermal conduction only, we can analyze this situation with reference to Figure 1.1.

An energy balance on a small volume of the solid of cross-sectional area A and thickness Δx states

Rate of accumulation of energy = rate of heat conduction in

— rate of heat conduction out (1.1)

Equation (1.1) illustrates that mathematical models are typically derived through the use of basic principles; in this case, the *conservation of energy*.

The next step in deriving the model is to state mathematically each of the terms of the basic equation. For the accumulation of energy, we can write

$$A \Delta x \rho C_p \frac{\partial T}{\partial t}$$

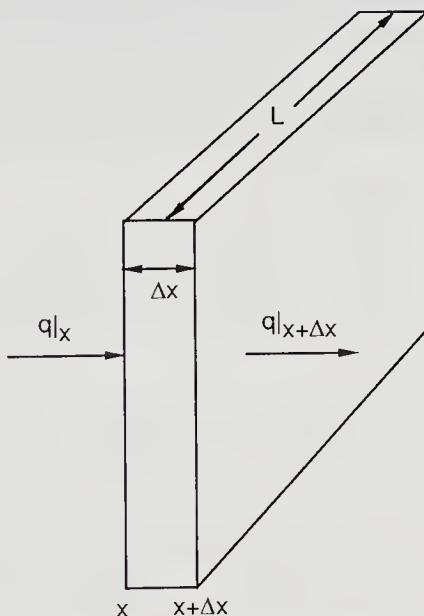


Figure 1.1 Heat conduction in a solid.

where

- T solid temperature ($^{\circ}\text{C}$)
- x position along the solid (cm)
- Δx width of the small volume on which the energy balance is written (cm)
- A cross-sectional area for the small volume (cm^2)
- t time (sec)
- ρ solid density (g/cm^3)
- C_p solid specific heat ($\text{J}/\text{g}^{\circ}\text{C}$)

The units of the accumulation term are

$$(\text{cm}^2)(\text{cm})(\text{g}/\text{cm}^3)(\text{J}/\text{g}^{\circ}\text{C})(^{\circ}\text{C}/\text{sec}) = \text{J/sec}$$

These net units (J/sec) represent the rate of accumulation of energy (heat) within the small volume.

The rate of heat conduction in equation (1.1) is given by $Aq|_x$ where q_x is heat flux at x in joules per square centimeter per second. The units of this term are

$$(\text{cm}^2)(\text{J}/\text{cm}^2 \cdot \text{sec}) = \text{J/sec}$$

and the net units (J/sec) represent the heat flow into the volume by conduction at x .

Similarly, the rate of heat conduction out [equation (1.1)] at $x + \Delta x$ is $Aq|_{x+\Delta x}$, which, of course, also has the net units (joules per second) representing the heat flow out of the volume by conduction at $x + \Delta x$.

Then, by the energy balance of equation (1.1)

$$A \Delta x \rho C_p \frac{\partial T}{\partial t} = Aq|_x - Aq|_{x+\Delta x}$$

If this equation is divided by $A \Delta x \rho C_p$,

$$\frac{\partial T}{\partial t} = - \left(\frac{1}{\rho C_p} \right) \frac{q|_{x+\Delta x} - q|_x}{\Delta x}$$

and in the limit as $\Delta x \rightarrow 0$,

$$\frac{\partial T}{\partial t} = - \left(\frac{1}{\rho C_p} \right) \frac{\partial q}{\partial x} \quad (1.2)$$

Equation (1.2) is the energy balance for the volume in Figure 1.1 (with the limiting thickness Δx equal to zero). Note that it is a PDE since it has two independent variables, x and t . It also has two dependent variables, T and q , and therefore we have an incomplete mathematical model (one

PDE in two dependent variables, T and q) so that a second equation is required relating the two dependent variables.

For this second equation, we use

$$q = -k \frac{\partial T}{\partial x} \quad (1.3)$$

Equation (1.3) is *Fourier's first law* for heat conduction; the minus sign is required so that the heat flux q is in the direction of decreasing temperature (e.g., q is positive when $\partial T/\partial x$ is negative).

Substitution of equation (1.3) (with constant k) in equation (1.2) gives finally

$$\frac{\partial T}{\partial t} = \left(\frac{k}{\rho C_p} \right) \frac{\partial^2 T}{\partial x^2} \quad (1.4)$$

which is *Fourier's second law for heat conduction in Cartesian coordinates*. The group $k/\rho C_p$ is the *thermal diffusivity* for the solid [note that it has the units square centimeters per second as expected from the variables T , t , and x in equation (1.4)].

Equation (1.4) is the basic PDE model for the problem of finding the temperature T of the solid as a function of time t and position x ; however, it is not a complete mathematical model. In addition to the PDE, we must specify *auxiliary conditions* to complete the model.

Auxiliary conditions are required for *each independent variable*. The number of conditions for each independent variable equals the *order of the highest-order derivative for the independent variable*. Thus, for t , $\partial T/\partial t$ is the highest-order derivative in equation (1.4) (a first-order derivative), so one auxiliary condition is required. Since t in equation (1.4) represents time, the auxiliary condition is called an *initial condition* and might be stated as

$$T(x, 0) = T_0(x) \quad (1.5)$$

where $T_0(x)$ is a prescribed initial temperature distribution at $t = 0$ (typically a constant). Note also the terminology $T(x, 0)$, which is temperature as a function of x at $t = 0$.

More generally, an initial condition is specified at only *one value of the independent variable*. In this case, there is no alternative since there is just one initial condition; if, however, two or more initial conditions are required in specifying a PDE problem, they will all be specified at the same value of the initial-value independent variable.

In the same way, x in equation (1.4) requires two auxiliary conditions since it appears in a second-order derivative ($\partial^2 T/\partial x^2$). For example, we

might specify the temperature at two boundaries of the solid in Figure 1.1, say, $x = 0$ and $x = L$:

$$T(0, t) = T_1(t) \quad T(L, t) = T_2(t) \quad (1.6)(1.7)$$

where $T_1(t)$ and $T_2(t)$ are prescribed functions. Since the temperatures are specified at the boundaries of the solid, auxiliary conditions (1.6) and (1.7) are termed *boundary conditions*; note that the boundary conditions can be functions of time t .

More generally, boundary conditions are specified for a dependent variable at *two or more values of the independent variable* (in contrast to an initial condition, which is specified at only one value of the independent variable). We therefore distinguish between initial-value independent variables, e.g., t , and *spatial* or boundary-value independent variables, e.g., x . In most applications of PDEs, the boundary conditions are specified for derivatives in the PDEs that reflect the variation of the dependent variable in space (or, in other words, the variation of the dependent variable with respect to the spatial independent variable), such as $\partial^2 T / \partial x^2$; these derivatives in the PDEs are referred to as *spatial derivatives*. The calculation of the spatial derivatives is a major consideration throughout the remainder of this book.

Boundary conditions of various types are also possible. Equations (1.6) and (1.7) are termed *Dirichlet boundary conditions* since the dependent variable T is specified at $x = 0$ and L . If the derivative of the dependent variable with respect to the boundary-value independent variable is specified, we have a *Neumann boundary condition*. For example,

$$\frac{\partial T(0, t)}{\partial x} = 0, \quad \frac{\partial T(L, t)}{\partial x} = 0 \quad (1.8)(1.9)$$

From equation (1.3), we see that if the temperature gradient $\partial T / \partial x$ is zero, the flux q is also zero. Thus equations (1.8) and (1.9) are zero flux or *insulated* boundary conditions. In general, derivatives with respect to the spatial or boundary-value independent variable must be at least one order lower than the highest-order derivative in the PDE; e.g., since equation (1.4) is second-order in x , the boundary conditions such as equations (1.8) and (1.9) can be no higher than first-order.

Equations (1.4), (1.5), (1.6), and (1.7) [or (1.8) and (1.9) in place of (1.6) and (1.7)] now constitute a complete PDE model (the PDE plus all of the required auxiliary conditions). We can therefore now think about computing a solution. However, before we do, we shall consider an alternative nomenclature that will make expressing PDEs considerably easier and will also facilitate the programming of solutions.

1.3 Subscript Notation for PDEs

If we adopt the convention that a subscript in an independent variable denotes a partial derivative with respect to that independent variable, equation (1.4) can be simply stated as

$$T_t = \alpha T_{xx} \quad (1.10)$$

where $\alpha = k/pC_p$. This concise notation will make the specification of PDE models considerably easier (than expressing partial derivatives in terms of ∂), and the computer programming of the solutions will be directly related to the simplified subscript notation.

As a second example, equations (1.8) and (1.9) can be stated simply as

$$T_x(0, t) = T_x(L, t) = 0 \quad (1.11)(1.12)$$

As a third, somewhat more complicated, example, the PDE problem

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x} \left(k(T) \frac{\partial T}{\partial x} \right) \quad (1.13)$$

$$T(x, 0) = T_0(0) \quad T_\infty(\infty) \quad (1.14)$$

$$\frac{\partial T(0, t)}{\partial x} = 0 \quad (1.15)$$

$$k \frac{\partial T(L, t)}{\partial x} = \epsilon(T_a^4 - T(L, t)^4) \quad (1.16)$$

can be stated with the subscript notation as

$$T_t = (k(T)T_x)_x \quad (1.17)$$

$$T(x, 0) = T_0(0) \quad T_\infty(\infty) \quad (1.18)$$

$$T_x(0, t) = 0 \quad (1.19)$$

$$k T_x(L, t) = \epsilon(T_a^4 - T(L, t)^4) \quad (1.20)$$

Note that boundary condition (1.16) [or (1.20)] is a combination of a Dirichlet boundary condition [since it involves the dependent variable $T(L, t)$] and a Neumann boundary condition [since it also involves the first-order derivative of the dependent variable $\partial T(L, t)/\partial x$]; a combination of Dirichlet and Neumann boundary conditions will be called a *boundary condition of the third type*. Also, in equation (1.13) [or (1.17)], the thermal conductivity k is a function of temperature [$k = k(T)$], so it

cannot be taken outside the second derivative in x as in equation (1.4); [also, $\rho C_p = 1$ in equation (1.13)].

We have formulated a complete PDE model [equations (1.4) or (1.10), (1.5), (1.6), and (1.7)], and we can therefore consider obtaining a solution. However, before we do that, we must first determine rather precisely what we mean by a solution to a PDE. (Computers require attention to every detail, and in this case we cannot compute a solution until we know precisely what we are trying to compute.)

1.4 What Is a Solution to a PDE? ---

Equations (1.4), (1.5), (1.6), and (1.7) relate the dependent variable T to the independent variables x and t . We would therefore expect that if we could find a solution $T(x, t)$, it would simultaneously satisfy equations (1.4) to (1.7). This solution can take one of two forms: (a) a mathematical function or (b) a numerical approximation of the mathematical function. We will now consider both forms of the solution.

To facilitate the discussion, consider the special case of equations (1.4) to (1.7):

$$T_t = T_{xx} \quad (1.21)$$

$$T(x, 0) = \sin(\pi x/L) \quad (1.22)$$

$$T(0, t) = T(L, t) = 0 \quad (1.23)(1.24)$$

A solution to equations (1.21) to (1.24) is a mathematical function that satisfies the PDE [equation (1.21)] and all of its auxiliary conditions [equations (1.22) to (1.24)]. This function is easily derived as

$$T(x, t) = e^{-(\pi^2/L^2)t} \sin(\pi x/L) \quad (1.25)$$

Equation (1.25) can be verified as the solution to equations (1.21) to (1.24) by substitution in these equations. For equation (1.21)

$$T_t - (\pi^2/L^2)e^{-(\pi^2/L^2)t} \sin(\pi x/L)$$

$$T_{xx} - (\pi^2/L^2)e^{-(\pi^2/L^2)t} \sin(\pi x/L)$$

Thus, equation (1.21) is satisfied. The auxiliary conditions, equations (1.22) to (1.24), are easily checked (by inspection), and therefore equation (1.25) is a solution to equations (1.21) to (1.24).

Equation (1.25) is termed (a) an *analytical solution*, (b) an *exact solution*, or (c) a *closed-form solution*. It has several useful properties:

(a) Since it is a mathematical function, it can, in principle, be manipulated to obtain other useful mathematical functions. For example, if we are interested in the heat flux at the surface $x = 0$, then from Fourier's first law, equation (1.3), we obtain

$$\begin{aligned} q(0, t) &= -kT_x(0, t) = -k(\pi/L)e^{-(\pi^2/L^2)t} \cos(\pi 0/L) \\ &= -(k\pi/L)e^{-(\pi^2/L^2)t} \end{aligned} \quad (1.26)$$

If, additionally, we are interested in the total amount of heat $Q_{\text{tot}}(t)$ that crosses the surface $x = 0$ as a function of time t , then

$$Q_{\text{tot}}(t) = \int_0^t q(0, t) dt \quad (1.27)$$

Substitution of equation (1.26) in equation (1.27), followed by an easy integration in t , gives $Q_{\text{tot}}(t)$.

(b) Since equation (1.25) is a mathematical function, it can be studied to elucidate its mathematical properties. For example, this solution is very *smooth* in the sense that derivatives of all orders exist with respect to both x and t (the exponential in t and the sine in x are infinitely differentiable).

(c) Since equation (1.25) is an exact solution, it gives $T(x, t)$ that satisfies equations (1.21) to (1.24) *with no error*. This is quite useful in evaluating numerical solutions (the second type of solution to be discussed next), since numerical solutions contain errors, and we often wish to determine whether the *numerical algorithms* (computational procedures) we use give numerical solutions of acceptable accuracy. Because of these desirable properties of analytical solutions, we generally would like to be able to derive and use them in studying mathematical models based on PDEs. Unfortunately, we are able to derive analytical solutions only under very special circumstances, essentially when the PDE problem is quite simple. In particular, to derive analytical solutions, we require that the mathematical model involve

- (a) Only a few PDEs, e.g., less than three PDEs.
- (b) Only *linear* PDEs (this property will be discussed subsequently).

The net result of these requirements is that most PDE models of practical interest are excluded, i.e., we cannot derive exact solutions and therefore we must compute numerical solutions to the PDEs. The reason for this very limited class of PDE problems that can be solved analytically is much the same as for algebraic equations: In general, we can solve only a few problems in algebraic equations, namely, small sets of linear equations. Therefore, we now proceed to numerical solutions, and in particular, the numerical method of lines.

1.5 A Brief Introduction to the Numerical Method of Lines

If we consider how we might proceed to produce a computer solution of equations (1.21) to (1.24), we quickly realize there is no direct way to tell the computer about a problem in PDEs (working, for example, with a standard compiler like Fortran); computers don't naturally understand partial derivatives. Rather, we must state the problem in PDEs in a format that can then be programmed using a standard compiler; basically, this means replacing the original PDE problem with an equivalent problem in algebra.

For example, if we seek $T(x, t)$ that satisfies equations (1.21) to (1.24), we might consider the variation of T with respect to x to take place along a grid in x where a particular value of x will be specified in terms of an integer index i . Thus, $x(1)$ corresponds to $x = 0$ and $x(N)$ to $x = L$, where N is the total number of grid points in x . A particular value of x is then given by $x(i) = (i - 1) \Delta x$, $i = 1, 2, \dots, N$, where Δx is the grid spacing, i.e., $\Delta x = L/(N - 1)$. Similarly, the variation in T with respect to t could be specified in terms of an integer index j , so $t(1)$ corresponds to $t = 0$ and $t(j - 1) = (j - 1) \Delta t$, $j = 1, 2, \dots$, corresponding to a grid spacing in t of Δt .

With these two subscripts, i and j , we could specify a value of T corresponding to particular values of x and t , $T(i, j) = T((i - 1) \Delta x, (j - 1) \Delta t)$. Then we could replace the partial derivatives in equation (1.21) with *algebraic approximations* evaluated at a general point with indices (i, j) ; this would lead to a set of algebraic equations that approximate equation (1.21). Once the approximating algebraic equations have been defined, they could be solved using any standard linear equation solver [Kahaner (1.2)] to obtain an *approximate numerical solution* to equation (1.21); of course, the auxiliary conditions, equations (1.22) to (1.24), would also have to be included in the algebraic equations. This procedure is the basis for well-known classical *finite difference*, *finite element*, and *finite volume* methods for PDEs. The NUMOL is really just a small departure from this basic approach.

In the NUMOL, we retain the index i to account for variations of T with x , but we treat t as a continuous variable (rather than t evaluated at discrete points corresponding to the index j). Thus, we will replace the partial derivative T_{xx} in equation (1.21) with an algebraic approximation evaluated at point i , but keep the derivative T_t ; this will lead to a system of differential equations in t , and since we now have only one independent variable, t , the differential equations will be ODEs. *This is the essence of the NUMOL!* Since one example is probably worth a thousand words, we will now consider the NUMOL solution of equations (1.21) to (1.24).

Assume that T is stored in a one-dimensional Fortran array, $T(I)$, $I = 1, 2, \dots, N$ (again, subscript I is used to denote a particular value of x). Also, the first derivative, T_x , and the second derivative, T_{xx} , are stored in arrays $TX(I)$ and $TXX(I)$, respectively. We now assume we have a subroutine DSS002¹ that accepts array T as an input and computes the first derivative, TX , as an output over a grid of N points, via the call

```
CALL DSS002(0., L, N, T, TX)
```

where again L is the total length of the PDE problem system in x . Then, the second derivative TXX can be computed by

```
CALL DSS002(0., L, N, TX, TXX)
```

i.e., TXX is the derivative of TX . Once TXX is available, it can be used in equation (1.21).

We must also compute the derivative T_t in equation (1.21), which will be stored in an array $TT(I)$. This can be done with a single Fortran DO loop according to equation (1.21) ($\alpha = 1$)

```
DO 2 I=1,N
      TT(I)=TXX(I)
2      CONTINUE
```

Thus, the N derivatives of T with respect to t are now calculated (and are in array TT). If we have a subroutine that integrates derivatives with respect to an initial-value independent variable, e.g., t , $TT(I)$ can be integrated to $T(I)$, which, in turn, is the input to the first call to DSS002.

However, in following this procedure, we have not included the initial condition, equation (1.22), and boundary conditions, equations (1.23) and (1.24). To do this, we can combine the preceding Fortran statements and include equations (1.22), (1.23), and (1.24) where $L = 1.0$,

```
C...
C...  INITIAL CONDITION (1.22)
      IF(TIME.EQ.0.)THEN
          DO 1 I=1,N
              X(I)=FLOAT(I-1)*DX
              T(I)=SIN(PI*X(I)/L)
1          CONTINUE
      END IF
C...
C...  BOUNDARY CONDITION (1.23)
      T(1)=0.
```

continues

¹ Subroutine DSS002 will subsequently be discussed in detail; it is also included in Fortran 77 on the diskettes available from the author.

```

C...
C...  BOUNDARY CONDITION (1.24)
      T(N)=0.
C...
C...  DERIVATIVE TX
      CALL DSS002(0.,L.,N,T,TX)
C...
C...  DERIVATIVE TXX
      CALL DSS002(0.,L,N,TX,TXX)
C...
C...  EQUATION (1.21)
      DO 2 I=1,N
      TT(I)=TXX(I)
2    CONTINUE

```

$DX = L/(N - 1)$, $PI = 3.1415927$, and N is set to the number of grid points in x . The preceding coding illustrates one of the major advantages of the NUMOL, the close resemblance of the coding to the problem equations [equations (1.21) to (1.24)]; this is in contrast to more conventional approaches to the numerical integration of PDEs in which the algebraic equations bear little resemblance to the original problem equations. This close correspondence of the problem equations and the NUMOL coding significantly enhances the understanding and debugging of the coding.

This essentially completes the NUMOL programming of equations (1.21) to (1.24), assuming that subroutine DSS002 is available and that a subroutine is available for the integration of $TT(I)$ to $T(I)$; this latter subroutine will be called the ODE integrator, and in the following coding, has the name RKF45 [Forsythe (1.1)] (it will also set the value of TIME ($= t$)). We now consider the remaining parts of the program; i.e., we will now put together a complete NUMOL code for the solution of equations (1.21) to (1.24).

The main program PR01P1 is listed in Program 1.1a.

Program 1.1a Main Program for NUMOL Integration of Equations (1.21) to (1.24) by RKF45 and DSS002

```

PROGRAM PR01P1
C...
C...  MAIN PROGRAM FOR THE NUMOL INTEGRATION OF EQUATIONS (1.21) TO
C...  (1.24) USING DIFFERENTIATION IN SPACE SUBROUTINE DSS002 AND ODE
C...  INTEGRATOR RKF45
C...
C...  SET THE NUMBER OF ODES
      PARAMETER (N=51)
C...

```

continues

```

C... ABSOLUTE DIMENSION THE ARRAYS (WORK AND IWORK ARE REAL AND INTEGER
C... WORK ARRAYS REQUIRED BY SUBROUTINE RKF45)
REAL T(N), TT(N), TE(N), DIFF(N), WORK(1000)
INTEGER IWORK(5)
C...
C... EXTERNAL THE DERIVATIVE SUBROUTINE CALLED BY RKF45
EXTERNAL FCN
C...
C... DEFINE THE INPUT/OUTPUT UNIT NUMBERS
NI=5
NO=6
C...
C... OPEN INPUT AND OUTPUT FILES
OPEN(NI,FILE='DATA')
OPEN(NO,FILE='OUTPUT')
C...
C... SET THE INITIAL, FINAL AND PRINT INTERVAL TIMES
TIME=0.
TF=0.5
TP=0.1
C...
C... SET THE PARAMETERS FOR SUBROUTINE RKF45
RELERR=1.0E-06
ABSERR=0.0
IFLAG=1
TOUT=TP
C...
C... CALL SUBROUTINE FCN TO SET THE INITIAL CONDITION
CALL FCN(TIME,T,TT)
C...
C... CALCULATE THE EXACT SOLUTION, AND THE DIFFERENCE BETWEEN THE
C... NUMOL AND EXACT SOLUTIONS
1 DO 3 I=1,N,10
    TE(I)=EXACT(I,TIME)
    DIFF(I)=T(I)-TE(I)
3 CONTINUE
C...
C... PRINT THE INITIAL CONDITION (IFLAG = 1) OR THE SOLUTION AT THE
C... NEXT OUTPUT POINT (IFLAG = 2)
WRITE(NO,2) TIME,(T(I),I=1,N,10),
1           (TE(I),I=1,N,10),
2           (DIFF(I),I=1,N,10)
2 FORMAT(' TIME = ',F6.2,/,15X,' X=0',5X,' X=0.2',5X,' X=0.4',5X,
1                   ' X=0.6',5X,' X=0.8',5X,' X=1',/,,
2                   ' T(X,T)',6F10.6,/,,
3                   ' TE(X,T)',6F10.6,/,,
4                   ' DIFF(X,T)',6F10.6,/)
C...
C... CHECK FOR THE FINAL VALUE OF TIME
IF(TIME.GT.(TF-0.5*TP))THEN
C...
C...     THE SOLUTION IS COMPLETE SO TERMINATE EXECUTION
STOP
ELSE
C...
C...     SET THE NEXT OUTPUT TIME AND CONTINUE THE INTEGRATION
TOUT=TIME+TP
END IF
C...
C... CALL SUBROUTINE RKF45 TO START THE SOLUTION FROM THE INITIAL
C... CONDITION (IFLAG = 1) OR COMPUTE THE SOLUTION TO THE NEXT OUTPUT
C... POINT (IFLAG = 2) AT TOUT
CALL RKF45(FCN,N,T,TIME,TOUT,RELERR,ABSERR,IFLAG,WORK,IWORK)
C...

```

continues

```

C... TEST FOR AN ERROR CONDITION
C... IF(IFLAG.NE.2)THEN
C...
C... PRINT A MESSAGE INDICATING AN ERROR CONDITION AND TERMINATE
C... EXECUTION
C... WRITE(NO,1000) IFLAG
C... STOP
C... ELSE
C...
C... CONTINUE THE INTEGRATION
C... GO TO 1
C... END IF
C...
C... ****
C... FORMATS
C...
1000 FORMAT(1H ,//,' IFLAG = ',I3,//,
1 ' INDICATING AN INTEGRATION ERROR, SO THE CURRENT RUN' ,/,
2 ' IS TERMINATED. PLEASE REFER TO THE DOCUMENTATION FOR' ,/,
3 ' SUBROUTINE' ,//,25X,'RKF45' ,//,
4 ' FOR AN EXPLANATION OF THESE ERROR INDICATORS' ,)
C... END

```

The following features of main program PRO1P1 are noteworthy:

- (1) The number of spatial grid points (and thus, the number of ODEs) is set to $N = 51$ (the NUMOL code was run with $N = 11, 21, 31, 41$, and 51 to demonstrate the effect of the number of grid points on the solution; the results are summarized in a subsequent table).
- (2) The initial value of t (TIME = 0), the final value of t (TF = 0.5), and the interval in t at which the solution is to be printed, the *print interval* (TP = 0.1), are set; note that initial-value problems are open-ended with respect to how long the calculation should proceed, and therefore a final value of the initial value variable must be specified.
- (3) Subroutine RKF45 is called to integrate an ODE at each grid point in x (TT(I) is integrated to T(I), $I = 1, 2, \dots, N$).
- (4) The exact solution to equations (1.21) to (1.24), equation (1.25), is evaluated for each t [TE(I)] and printed along with the NUMOL solution [T(I)], as well as the difference between the two solutions ([DIFF(I)]).
- (5) By looping back to statement 1, the solution is calculated at $t = 0, 0.1, 0.2, \dots, 0.5$.
- (6) After each output value of t (0.1, 0.2, . . . , 0.5), a check is made on the value of the error flag, IFLAG; if subroutine RKF45 has detected that the integration of the ODEs has not proceeded satisfactorily, an error message is printed (via FORMAT 1000) and execution is stopped.

The subroutine defining the ODE derivatives, FCN, and the function for the exact solution, EXACT, are listed in Program 1.1b.

Program 1.1b ODE Subroutine FCN and Function EXACT (Equations (1.21) to (1.24))

```

SUBROUTINE FCN(TIME,T,TT)
C...
C... SUBROUTINE FCN IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... INITIAL CONDITION AND ODE DERIVATIVES IN THE NUMOL SOLUTION OF
C... EQUATIONS (1.21) TO (1.24)
C...
C... DIMENSION THE ARRAYS REQUIRED IN FCN
C... PARAMETER (N=51)
C... REAL T(N), TT(N), X(N), TX(N), TXX(N)
C...
C... TYPE SELECTED VARIABLES AS REAL
C... REAL L
C...
C... CALCULATE PI FOR USE IN DO LOOP 1
C... PI=ACOS(-1.0)
C...
C... LENGTH
C... L=1.0
C...
C... GRID SPACING
C... DX=L/FLOAT(N-1)
C...
C... INITIAL CONDITION (1.22)
C... IF(TIME.EQ.0.)THEN
C...   DO 1 I=1,N
C...     X(I)=FLOAT(I-1)*DX
C...     T(I)=SIN(PI*X(I)/L)
C...   CONTINUE
C... END IF
C...
C... BOUNDARY CONDITION (1.23)
C... T(1)=0.
C...
C... BOUNDARY CONDITION (1.24)
C... T(N)=0.
C...
C... DERIVATIVE TX
C... CALL DSS002(0.,L,N,T,TX)
C...
C... DERIVATIVE TXX
C... CALL DSS002(0.,L,N,TX,TXX)
C...
C... EQUATION (1.21)
C... DO 2 I=1,N
C...   TT(I)=TXX(I)
C... CONTINUE
C... RETURN
C... END

      REAL FUNCTION EXACT(I,T)
C...
C... FUNCTION EXACT COMPUTES THE EXACT SOLUTION TO EQUATIONS (1.21)
C... TO (1.24), I.E., EQUATION (1.25), AT GRID INDEX I AND TIME T
C...
C... NUMBER OF GRID POINTS
C... PARAMETER (N=51)
C...
C... TYPE SELECTED VARIABLES AS REAL
C... REAL L
C...
C... PI FOR USE IN THE EXACT SOLUTION
C... PI=ACOS(-1.)

```

continues

16 1. What Is the Numerical Method of Lines?

```

C...
C... LENGTH
L=1.
C...
C... GRID SPACING
DX=L/FLOAT(N-1)
C...
C... X FOR WHICH THE EXACT SOLUTION IS TO BE COMPUTED
X=FLOAT(I-1)*DX
C...
C... EXACT SOLUTION AT X AND T
EXACT=EXP((-PI**2/L**2)*T)*SIN(PI*X/L)
RETURN
END

```

The programming in subroutine FCN and function EXACT should be essentially self-explanatory from the preceding discussion. Note in particular that FCN accepts the dependent variable vector, T(I), $I = 1, 2, \dots, N$, and the independent variable TIME, and returns the ODE derivative vector, TT(I), all through the arguments of FCN.

Table 1.1 Numerical Solution from Programs 1.1a and 1.1b

TIME =	0.00	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	0.000000	.587785	.951057	.951057	.587785 0.000000
		TE(X,T)	0.000000	.587785	.951057	.951057	.587785 .000000
		DIFF(X,T)	0.000000	0.000000	0.000000	0.000000	0.000000 .000000
TIME =	.10	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	0.000000	.219357	.354927	.354927	.219357 0.000000
		TE(X,T)	0.000000	.219072	.354466	.354466	.219072 .000000
		DIFF(X,T)	0.000000	.000285	.000460	.000460	.000285 .000000
TIME =	.20	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	0.000000	.081862	.132456	.132456	.081862 0.000000
		TE(X,T)	0.000000	.081650	.132112	.132112	.081650 .000000
		DIFF(X,T)	0.000000	.000212	.000343	.000343	.000212 .000000
TIME =	.30	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	0.000000	.030550	.049431	.049431	.030550 0.000000
		TE(X,T)	0.000000	.030432	.049239	.049239	.030432 .000000
		DIFF(X,T)	0.000000	.000119	.000192	.000192	.000119 .000000
TIME =	.40	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	0.000000	.011401	.018447	.018447	.011401 0.000000
		TE(X,T)	0.000000	.011342	.018352	.018352	.011342 .000000
		DIFF(X,T)	0.000000	.000059	.000096	.000096	.000059 .000000
TIME =	.50	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	0.000000	.004255	.006884	.006884	.004255 0.000000
		TE(X,T)	0.000000	.004227	.006840	.006840	.004227 .000000
		DIFF(X,T)	0.000000	.000028	.000045	.000045	.000028 .000000

The output from Programs 1.1a and 1.1b is in Table 1.1. Note also that we have essentially obtained $T(x, t)$, that is, T as a function of x and t that satisfies equations (1.21) to (1.24) within the errors introduced by the NUMOL procedures; we shall examine these errors in some detail in subsequent sections since we must have some assurance that they are acceptably small. The NUMOL and exact solutions are the same at $t = 0$ [as expected, since initial condition (1.22) is the same for both solutions]. The solution is symmetrical about $x = 0.5$ (which suggests that we could achieve some computational efficiency by taking advantage of this symmetry and computing the solution over the interval $0 \leq x \leq 0.5$ or $0.5 \leq x \leq 1$). Also, boundary conditions (1.23) and (1.24) are satisfied. Fortunately, the error decays as the solution decays (note the solution and error at $t = 0.1$, then at $t = 0.5$) so that *the error remains a relatively small part of the solution* (at least to a reasonable level for a scientific or engineering application).

We should also appreciate why $t = 0.5$ is an acceptable final time for this problem. In fact, for every PDE problem we consider in the remainder of this book, we shall have to *select a final value for the initial-value independent variable (t)* since the integration is essentially an open-ended process for initial-value variables (mathematically, these variables can go to infinity, but practically, when we compute a PDE solution, we must specify a finite final value). In the present case, we know from the exact solution, equation (1.25), that t appears in an exponential that decays with increasing t :

$$e^{-(\pi^2/L^2)t}$$

For $L = 1$, $t = 0.5$, this exponential is

$$e^{-(\pi^2/L^2)t} = e^{-(\pi^2/1^2)0.5} = 0.007192$$

Thus, at $t = 0.5$, the solution has decayed to 0.7192% of its initial value (at $t = 0$), so that for practical purposes, the solution is complete at $t = 0.5$. Of course, we could let the integration go to a larger value of t , but we would observe little further change in the solution. In general, then, we will have to select a final value of the initial-value independent variable that gives essentially the complete solution. The fact that the NUMOL solution to equations (1.21) to (1.24) is essentially complete at $t = 0.5$ is indicated in Figure 1.2 (the plot for $N = 51$); Figure 1.2 also clearly demonstrates the symmetry and boundary conditions of the solution.

The coding in Programs 1.1a and 1.1b indicates that the numerical output of Table 1.1 was computed with single-precision Fortran. The

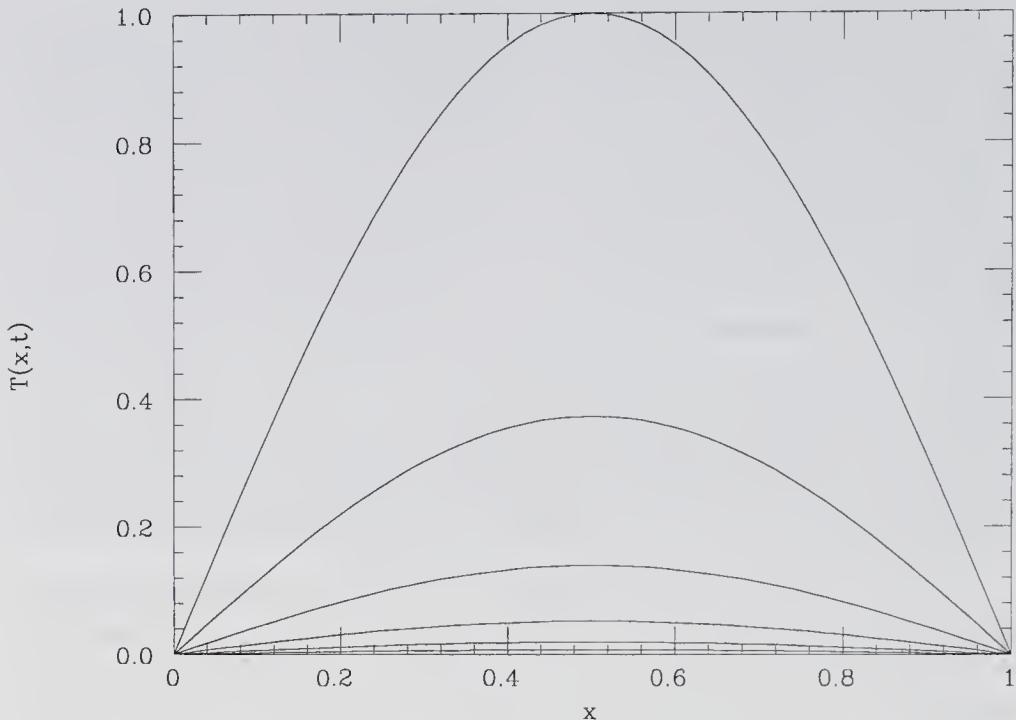


Figure 1.2 Graphical solution from Programs 1.1a and 1.1b.

calculations were performed on a computer with a 64-bit word length in single-precision Fortran, corresponding to approximately 14 decimal digits. For computers with 32-bit words in single-precision Fortran, *double-precision Fortran coding is recommended*. Discussions of the effect of word length on numerical computations are given by Forsythe (1.1) and Kahaner (1.2).

Programs 1.1a and 1.1b illustrate the essential features of a NUMOL code, namely: (a) a main program (driver) to call the ODE integrator and (b) a routine called by the ODE integrator to define the ODE derivatives. This suggests a natural division of the coding: (a) the general-purpose NUMOL coding can be put in the main program, and (b) the problem-dependent coding can be put in the ODE subroutine. This has the major advantage that the general-purpose code can be applied to any problem, and then only an ODE routine must be supplied for a new problem. We have not yet quite achieved this goal of the separation of the code into general and special-purpose sections (e.g., main program PRO1P1 contains some coding specific to the problem), but we will subsequently develop an approach to programming that will give this separation.

1.6 An Error Analysis of the NUMOL Solution

We now consider the errors in the NUMOL solution produced in Section 1.5. In using the NUMOL (or any other approximate numerical procedure), we should keep in mind that the *evaluation of errors is an essential part of the calculation*, and we should be prepared to state how close to the exact solution we think the numerical solution is; in a sense, this is asking the impossible since it implies that we know the exact solution, and if that were the case, we wouldn't be computing a numerical solution. However, the key here is to realize that we can probably only *estimate the error* in the numerical solution. We should, however, strive to make the estimate of the error as reliable and accurate as possible. In this sense, the present problem is quite artificial since we know the exact solution, and therefore have an absolute standard against which to evaluate the numerical solution. In practice, this will almost never be the case. However, by examining the *exact error*, rather than the estimated error, we gain some insight into how well the NUMOL performed in this particular case; certainly if it didn't do reasonably well on this very simple problem, we wouldn't have much to recommend it for more difficult problems based on this initial experience.

The exact error at $x = 0.4$ and $t = 0.1$ for the NUMOL solution as a function of the number of spatial grid points is presented in Table 1.2. The last entry in Table 1.2, $N = 51$, error = 0.000460, comes directly from Table 1.1 at $x = 0.4$, $t = 0.1$. The other entries were produced by running Programs 1.1a and 1.1b for $N = 11, 21, 31$, and 41. The only required changes are in the Fortran statements

PARAMETER (N=51)

in main program PRO1P1 and function FCN, and

```

      WRITE(NO,2)TIME,(T(I),I=1,N,10),
      1      (TE(I),I=1,N,10),
      2      (DIFF(I),I=1,N,10)
    
```

Table 1.2 Exact Error in $T(x = 0.4, t = 0.1)$

Number of Grid Points	Exact Error Computed by Main Program PRO1P1
11	0.011543
21	0.002880
31	0.001279
41	0.000719
51	0.000460

in PRO1P1; for the latter, the increment must be changed in the three implied DO loops, i.e., 2 for $N = 11$, 4 for $N = 21$, . . . , 10 for $N = 51$.

The error in Table 1.2 decreases with increasing N (we might conclude this is as expected, but actually, based on what we have discussed so far, this is not necessarily obvious; in fact, we can say something about how the error varies with N in general only after we have examined what is in subroutine DSS002). A more detailed error analysis will be given in Chapter 3.

The variation of the exact error with the number of spatial grid points is shown in Figure 1.3, which is a plot of the log of the errors in Table 1.2 vs. the log of $DX = 1/(N - 1)$ (the log of the grid spacing). The five points fall on essentially a straight line.

The slope of the line in Figure 1.3, $\Delta \log(\text{error})/\Delta \log(DX)$, can be estimated by using just the first and last points

$$\begin{aligned}\text{Slope} &= \frac{\log_{10}(0.011543) - \log_{10}(0.000460)}{\log_{10}(1/(11-1)) - \log_{10}(1/(51-1))} \\ &= \frac{-1.9377 - (-3.3372)}{-1.0000 - (-1.6990)} = 2.002\end{aligned}$$

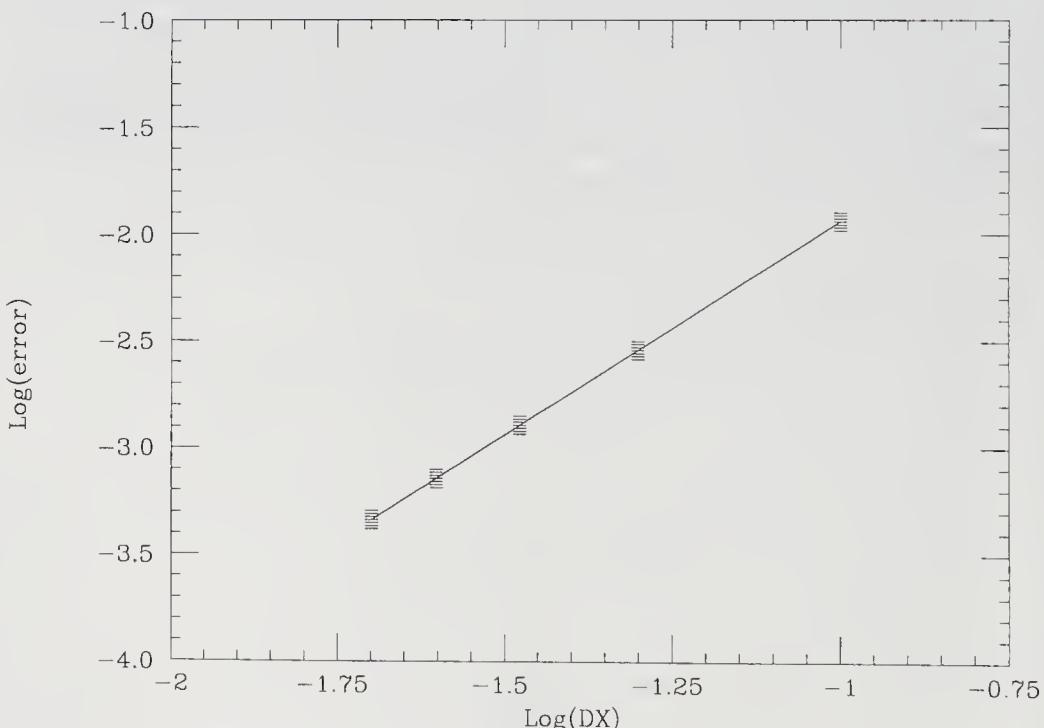


Figure 1.3 Log–log plot of the errors in Table 1.2.

Thus, the error varies as DX to the 2.002 power, or

$$\text{Error} = c(DX^{2.002})$$

This result is stated more generally as follows: the error is *of order* 2.002, or the solution is approximately *second order correct in x*. This can be stated in equation form as

$$\text{Error} = O(DX^{2.002})$$

where “ $O()$ ” is interpreted *as of order*. This type of *order analysis* will be used in subsequent sections for the study of errors in the NUMOL. One implication of the preceding result is that the error in the NUMOL solution of equations (1.21) to (1.24) decreases approximately as the square of the grid spacing, DX . Generally, higher order of an algorithm implies higher accuracy; for example, in Chapter 3, we shall investigate some fourth-and higher-order spatial differentiation formulas that have higher accuracy than the second-order formulas in DSS002 when applied to polynomials.

However, the increased accuracy with greater N as reflected in the data of Table 1.2 is gained at the expense of additional computation, which takes two forms:

(1) Clearly, we are integrating more ODEs as N increases, which requires more computation. This is evident in considering function FCN in Program 1.1b since the number of passes through DO loop 2 is just equal to N . The same is true for subroutine DSS002 (when we examine what is in DSS002 in Chapter 3, we will find that there is also a DO loop with an upper limit of N). Thus, the number of ODE derivatives to be computed will in general be proportional to the number of spatial grid points (and this calculation of the ODE derivatives can be the *major portion* of the NUMOL calculation).

(2) An additional computational effort is often required with increasing numbers of grid points, which is not so obvious as in (1) above. In fact, this additional computation occurred in the execution of Programs 1.1a and 1.1b, and we will now briefly explain what happened (more details will be given when we discuss the stability of the NUMOL in Chapters 4 and 5).

When Programs 1.1a and 1.1b were executed for $N = 11, 21, 31$, and 41 , everything went as expected (the NUMOL solution was computed and printed for $t = 0, 0.1, 0.2, \dots, 0.5$). However, when the execution for $N = 51$ was initiated, the solution for $t = 0, 0.1, 0.2, \dots, 0.4$ was computed and printed, then subroutine RKF45 terminated execution with an error message IFLAG = 4 (see Program 1.1a to confirm that IFLAG is

printed in an error message via FORMAT 1000). On reading the internal documentation comments in RKF45, we found that RKF45 terminated execution because of excessive calls to function FCN (the total number of calls to FCN exceeded the limit set in RKF45); evidently, the solution for $N = 51$ required more calls to FCN than the solutions for $N = 11$ to 41; this is in addition to the greater computation in FCN for $N = 51$ than for the smaller values of N as explained in (1) above.

To confirm that this was indeed the problem, a DATA statement in RKF45 that sets the maximum number of calls to FCN, MAXNFE, (if MAXNFE is exceeded, execution is terminated with the error flag IFLAG = 4)

```
DATA MAXNFE/2000/
```

was replaced with

```
DATA MAXNFE/4000/
```

and the solution for $N = 51$ ran to completion ($t = 0.5$). Additionally, a counter was put in function FCN to accumulate the calls to FCN and print the total number of calls at the end of the solution; this total was 3194 (between 2000 and 4000). Incidentally, the use of such a counter is a good diagnostic procedure in analyzing the performance of a new NUMOL code, and it can easily be done; a statement like NFCN=NFCN+1 is put in function FCN, and the variable NFCN is put in COMMON so that it can be initialized to zero in the main program (PRO1P1) before the first call to FCN, and printed out from the main program after the last call to FCN (when the solution is complete at $t = 0.5$; for the present problem, the final value was NFCN = 3194).

The question that naturally arises is why the run for $N = 51$ requires a relatively large number of calls to FCN. Briefly, the explanation is due to the *stability limit of the integration algorithm* in RKF45. Thus, the *stability of the ODE integrator* can be a major consideration in applying the NUMOL (and will be discussed in Chapter 4). In other words, the ODEs became *stiffer* as N increased, and we must therefore explore what is meant by a system of *stiff ODEs*, and what the implications of *stiffness* are in using the NUMOL. We will conclude in Chapter 4 that an ODE integrator designed specifically for stiff ODEs may be required in using the NUMOL.

To conclude this discussion, we can indicate at least the name and general characteristics of the ODE integrator in RKF45. RKF designates the Runge–Kutta–Fehlberg numerical integration algorithm. The 45 indicates that a fourth order Runge–Kutta (RK) method is embedded in a fifth-order RK method (the word *order* is again used in the same sense as when analyzing the errors of Table 1.1). Essentially, a solution to the ODEs

is computed with the fourth-order RK algorithm, and also with the fifth-order RK algorithm. The two solutions are then compared to estimate the error in the numerical solution. If the estimated error exceeds the user-specified error tolerance, the solution is repeated with a smaller step in t (the *integration step*), to improve the accuracy of the integration in t . Thus, the ODE numerical integration proceeds in steps in t that approximate true integration, and at each step, function FCN is called to evaluate the ODE derivatives in t ; this explains why so many calls to FCN are required (many steps are required to compute a solution that meets the user-specified error tolerance). This is a requirement for acceptable *accuracy*; the calls to FCN may also be increased by the limited *stability* of the integration algorithm (as explained previously). Thus *accuracy and stability* are two central considerations in putting together a NUMOL code (as they are for all numerical methods for integrating PDEs).

Finally, we might consider where the error tolerance for the ODE integration was specified in the solution of equations (1.21) to (1.24). As we might expect, the error tolerance was specified in main program PRO1P1 (Program 1.1a) before the call to RKF45 as the statements

```
RELEERR=1.0E-06
ABSERR=0.0
```

The first statement specifies a relative error tolerance of 10^{-6} , i.e., each dependent variable, $T(I)$, $i = 1, 2, \dots, N$, is computed from the corresponding derivative, $TT(I)$, $I = 1, 2, \dots, N$, with a relative accuracy of 10^{-6} (0.0001% error); the second statement specifies that an absolute error of 0.0 is also applied to each dependent variable during the ODE integration. The larger (relative) error tolerance then sets the integration step size.

Most quality ODE integrators permit the user to specify both a relative error tolerance and an absolute error tolerance for the ODE dependent variables; in the case of RKF45, these tolerances are used in the Fortran statement

```
TOL=RELEERR*ABS(Y(K))+ABSERR
```

where $Y(K)$ is the current value of dependent variable K (or, for the present problem, dependent variable $T(I)$). Without going into the mathematical or programming details (in RKF45), TOL is compared with the error estimated by the integration algorithm. If the estimated error exceeds TOL, the integration step size is reduced and the integration is repeated to achieve greater accuracy. This step size reduction is repeated until the estimated error is less than TOL (or the integration step is reduced to the minimum allowable value set in RKF45, at which time IFLAG is set to 6; when this error condition occurs, the main program PRO1P1 will termi-

nate execution on the return from RKF45 since IFLAG is not equal to 2, the value for a successful integration over one output interval).

Note that in the preceding Fortran statement, only a relative error tolerance can be specified by setting ABSERR=0, or only an absolute error tolerance can be specified by setting RELERR=0. The values of the tolerances used in Program 1.1a are quite stringent (0.0001% relative error), and exceed the usual requirements for the solution of a PDE. These stringent error tolerances were chosen so that the ODE integration would be performed with an accuracy that would surpass the accuracy of the spatial differentiation (calculation of TXX); in this way the error in the NUMOL solution could essentially be attributed to the spatial differentiation performed by subroutine DSS002 (we avoided an interaction between the errors of the ODE integration and the spatial differentiation). However, as the example in the next section demonstrates, this choice of such a stringent error tolerance precluded the calculation of an NUMOL solution with reasonable computer time, and some relaxation of the error tolerance was therefore required.

The details of error estimation and step size adjustment in ODE integrators will not be considered here since extensive coverage is available in several books [e.g., Gear (1.3), Kahaner (1.2)]. As we shall see in the next section, the *selection of the error tolerance for the ODE integrator is an important consideration in the development of a NUMOL code for PDEs*.

1.7 Dirichlet and Neumann Boundary Conditions

In the problem defined by equations (1.21) to (1.24), boundary conditions (1.23) and (1.24) are of the *Dirichlet* type since the dependent variable is specified at the boundaries $x = 0$ and $x = L$. We can also consider an analogous problem with *Neumann* boundary conditions

$$T_t = T_{xx} \quad (1.28)$$

$$T(x, 0) = \cos(\pi x/L) \quad (1.29)$$

$$T_x(0, t) = T_x(L, t) = 0 \quad (1.30)(1.31)$$

Note that *first-order spatial derivatives are now specified at $x = 0$ and $x = L$* . The exact solution to equations (1.28) to (1.31) is

$$T(x, t) = e^{-(\pi^2/L^2)t} \cos(\pi x/L) \quad (1.32)$$

The changes in Programs 1.a and 1.1b for this problem are minor.

Program 1.2a Main Program for NUMOL Integration by RKF45 and DSS002 (for Equations (1.28) to (1.31))

```

PROGRAM PR01P2
C...
C... MAIN PROGRAM FOR THE NUMOL INTEGRATION OF EQUATIONS (1.28) TO
C... (1.31) USING DIFFERENTIATION IN SPACE SUBROUTINE DSS002 AND ODE
C... INTEGRATOR RKF45
C...
C... SET THE NUMBER OF ODES
PARAMETER (N=51)
C...
C... ABSOLUTE DIMENSION THE ARRAYS (WORK AND IWORK ARE REAL AND INTEGER
C... WORK ARRAYS REQUIRED BY SUBROUTINE RKF45)
REAL T(N), TT(N), TE(N), DIFF(N), WORK(1000)
INTEGER IWORK(5)
C...
C... EXTERNAL THE DERIVATIVE SUBROUTINE CALLED BY RKF45
EXTERNAL FCN
C...
C... DEFINE THE INPUT/OUTPUT UNIT NUMBERS
NI=5
NO=6
C...
C... OPEN INPUT AND OUTPUT FILES
OPEN(NI,FILE='DATA')
OPEN(NO,FILE='OUTPUT')
C...
C... SET THE INITIAL, FINAL AND PRINT INTERVAL TIMES
TIME=0.
TF=0.5
TP=0.1
C...
C... SET THE PARAMETERS FOR SUBROUTINE RKF45
RELERR=1.OE-04
ABSERR=1.OE-06
IFLAG=1
TOUT=TP
C...
C... CALL SUBROUTINE FCN TO SET THE INITIAL CONDITION
CALL FCN(TIME,T,TT)
C...
C... CALCULATE THE EXACT SOLUTION, AND THE DIFFERENCE BETWEEN THE
C... NUMOL AND EXACT SOLUTIONS
1 DO 3 I=1,N,10
    TE(I)=EXACT(I,TIME)
    DIFF(I)=T(I)-TE(I)
3 CONTINUE
C...
C... PRINT THE INITIAL CONDITION (IFLAG = 1) OR THE SOLUTION AT THE
C... NEXT OUTPUT POINT (IFLAG = 2)
WRITE(NO,2)TIME,(T(I),I=1,N,10),
1           (TE(I),I=1,N,10),
2           (DIFF(I),I=1,N,10)
2 FORMAT(' TIME = ',F6.2,/,15X,' X=0',5X,'X=0.2',5X,'X=0.4',5X,
1                   'X=0.6',5X,'X=0.8',5X,' X=1',/,,
2                   '      T(X,T)',6F10.6,/,,
3                   '      TE(X,T)',6F10.6,/,,
4                   '      DIFF(X,T)',6F10.6,/)
C...
C... CHECK FOR THE FINAL VALUE OF TIME
IF(TIME.GT.(TF-0.5*TP))THEN

```

continues

```

C...
C...      THE SOLUTION IS COMPLETE SO TERMINATE EXECUTION
C...      STOP
C...      ELSE
C...
C...      SET THE NEXT OUTPUT TIME AND CONTINUE THE INTEGRATION
C...      TOUT=TIME+TP
C...      END IF
C...
C...      CALL SUBROUTINE RKF45 TO START THE SOLUTION FROM THE INITIAL
C...      CONDITION (IFLAG = 1) OR COMPUTE THE SOLUTION TO THE NEXT OUTPUT
C...      POINT (IFLAG = 2) AT TOUT
C...      CALL RKF45(FCN,N,T,TIME,TOUT,RELERR,ABSERR,IFLAG,WORK,IWORK)
C...
C...      TEST FOR AN ERROR CONDITION
C...      IF(IFLAG.NE.2)THEN
C...
C...          PRINT A MESSAGE INDICATING AN ERROR CONDITION AND TERMINATE
C...          EXECUTION
C...          WRITE(NO,1000) IFLAG
C...          STOP
C...      ELSE
C...
C...          CONTINUE THE INTEGRATION
C...          GO TO 1
C...      END IF
C...
C...      ****
C...      FORMATS
C...
1000 FORMAT(1H ,//,' IFLAG = ',I3,/,,
1 ' INDICATING AN INTEGRATION ERROR, SO THE CURRENT RUN' ,/,
2 ' IS TERMINATED. PLEASE REFER TO THE DOCUMENTATION FOR' ,/,
3 ' SUBROUTINE' ,//,25X,'RKF45',//,
4 ' FOR AN EXPLANATION OF THESE ERROR INDICATORS' ) )
END

```

The main program, PRO1P2, is listed in Program 1.2a. The only difference between Programs 1.1a and Program 1.2a is that the latter has the error tolerances

```

RELERR=1.0E-04
ABSERR=1.0E-06

```

The reason for this change in error tolerances will be explained subsequently.

The ODE derivative routine, FCN, and the function for the exact solution [equation (1.32))] are listed in Program 1.2b. Note in particular that the Neumann boundary conditions, equations (1.30) and (1.31), are set in subroutine FCN between the first and second calls to DSS002 [so that the correct first derivatives at the boundaries, TX(1) and TX(N), go into DSS002 when it computes TXX from TX]. This contrasts with the Dirichlet boundary conditions, equations (1.23) and (1.24), which are set in Program 1.1b before the two calls to DSS002 [so that the correct values

Program 1.2b ODE Subroutine FCN and Function EXACT for Equations (1.28) to (1.31)

```

SUBROUTINE FCN(TIME,T,TT)
C...
C... SUBROUTINE FCN IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... INITIAL CONDITION AND ODE DERIVATIVES IN THE NUMOL SOLUTION OF
C... EQUATIONS (1.28) TO (1.31)
C...
C... DIMENSION THE ARRAYS REQUIRED IN FCN
PARAMETER (N=51)
REAL T(N), TT(N), X(N), TX(N), TXX(N)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C... CALCULATE PI FOR USE IN DO LOOP 1
PI=ACOS(-1.0)
C...
C... LENGTH
L=1.0
C...
C... GRID SPACING
DX=L/FLOAT(N-1)
C...
C... INITIAL CONDITION (1.29)
IF(TIME.EQ.0.)THEN
    DO 1 I=1,N
        X(I)=FLOAT(I-1)*DX
        T(I)=COS(PI*X(I)/L)
1      CONTINUE
    END IF
C...
C... DERIVATIVE TX
CALL DSS002(0.,L,N,T,TX)
C...
C... BOUNDARY CONDITION (1.30)
TX(1)=0.
C...
C... BOUNDARY CONDITION (1.31)
TX(N)=0.
C...
C... DERIVATIVE TXX
CALL DSS002(0.,L,N,TX,TXX)
C...
C... EQUATION (1.28)
DO 2 I=1,N
    TT(I)=TXX(I)
2      CONTINUE
    RETURN
END

REAL FUNCTION EXACT(I,T)
C...
C... FUNCTION EXACT COMPUTES THE EXACT SOLUTION TO EQUATIONS (1.28)
C... TO (1.31), I.E., EQUATION (1.32), AT GRID INDEX I AND TIME T
C...
C... NUMBER OF GRID POINTS
PARAMETER (N=51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L

```

continues

```

C...
C... PI FOR USE IN THE EXACT SOLUTION
PI=ACOS(-1.)
C...
C... LENGTH
L=1.
C...
C... GRID SPACING
DX=L/FLOAT(N-1)
C...
C... X FOR WHICH THE EXACT SOLUTION IS TO BE COMPUTED
X=FLOAT(I-1)*DX
C...
C... EXACT SOLUTION AT X AND T
EXACT=EXP((-PI**2/L**2)*T)*COS(PI*X/L)
RETURN
END

```

of the dependent variables at the boundaries, $T(1)$ and $T(N)$, go into DSS002 when it computes TX from T . Also, in function EXACT, COS is used in the exact solution [equation (1.32)].

The output from Programs 1.2a and 1.2b ($N = 51$) is listed in Table 1.3. Again, the initial condition, equation (1.29), is satisfied exactly (at

Table 1.3 Numerical Solution from Programs 1.2a and 1.2b

TIME =	0.00	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	1.000000	.809017	.309017	-.309017	-.809017
		TE(X,T)	1.000000	.809017	.309017	-.309017	-.809017
		DIFF(X,T)	0.000000	0.000000	0.000000	0.000000	0.000000
TIME =	.10	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	.373155	.301897	.115316	-.115316	-.301897
		TE(X,T)	.372708	.301527	.115173	-.115173	-.301527
		DIFF(X,T)	.000447	.000370	.000143	-.000143	-.000370
TIME =	.20	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	.139248	.112657	.043032	-.043032	-.112657
		TE(X,T)	.138911	.112381	.042926	-.042926	-.112381
		DIFF(X,T)	.000337	.000275	.000106	-.000106	-.000275
TIME =	.30	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	.051963	.042040	.016058	-.016058	-.042040
		TE(X,T)	.051773	.041885	.015999	-.015999	-.041885
		DIFF(X,T)	.000189	.000154	.000059	-.000059	-.000154
TIME =	.40	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	.019390	.015688	.005992	-.005992	-.015688
		TE(X,T)	.019296	.015611	.005963	-.005963	-.015611
		DIFF(X,T)	.000094	.000077	.000029	-.000029	-.000077
TIME =	.50	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
		T(X,T)	.007236	.005854	.002236	-.002236	-.005854
		TE(X,T)	.007192	.005818	.002222	-.002222	-.005818
		DIFF(X,T)	.000044	.000036	.000014	-.000014	-.000036

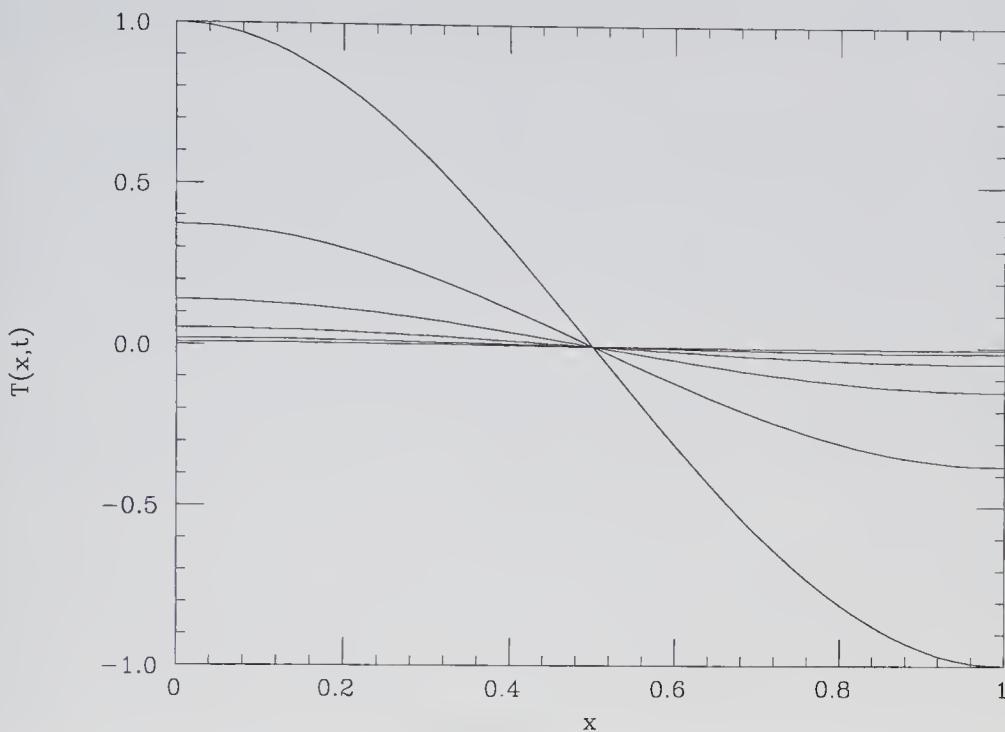


Figure 1.4 Graphical solution from Programs 1.2a and 1.2b.

$t = 0$). As the solution evolves through t , the exact solution, equation (1.32), and the NUMOL solution have the difference DIFF.

The solution in Table 1.3 is plotted in Figure 1.4 (51 points). Note that the boundary conditions, equations (1.30) and (1.31), appear to be satisfied (zero slopes at $x = 0$ and $x = L$). Also Figures 1.2 and 1.4 clearly indicate that the initial and boundary conditions can have a major effect on the solution of a PDE [equations (1.21) and (1.28) are the same; only the initial conditions, equations (1.22) and (1.29), and the boundary conditions, equations (1.23) and (1.24), or (1.30) and (1.31), are different)].

The difference between the exact and NUMOL values of $T(x = 0.4, t = 0.1)$ is shown in Table 1.4 for $N = 11, 21, 31, 41$, and 51. The last

Table 1.4 Exact Error in $T(x = 0.4, t = 0.1)$

Number of Grid Points	Exact Error Computed by Main Program PRO1P2
11	0.002829
21	0.000823
31	0.000383
41	0.000220
51	0.000143

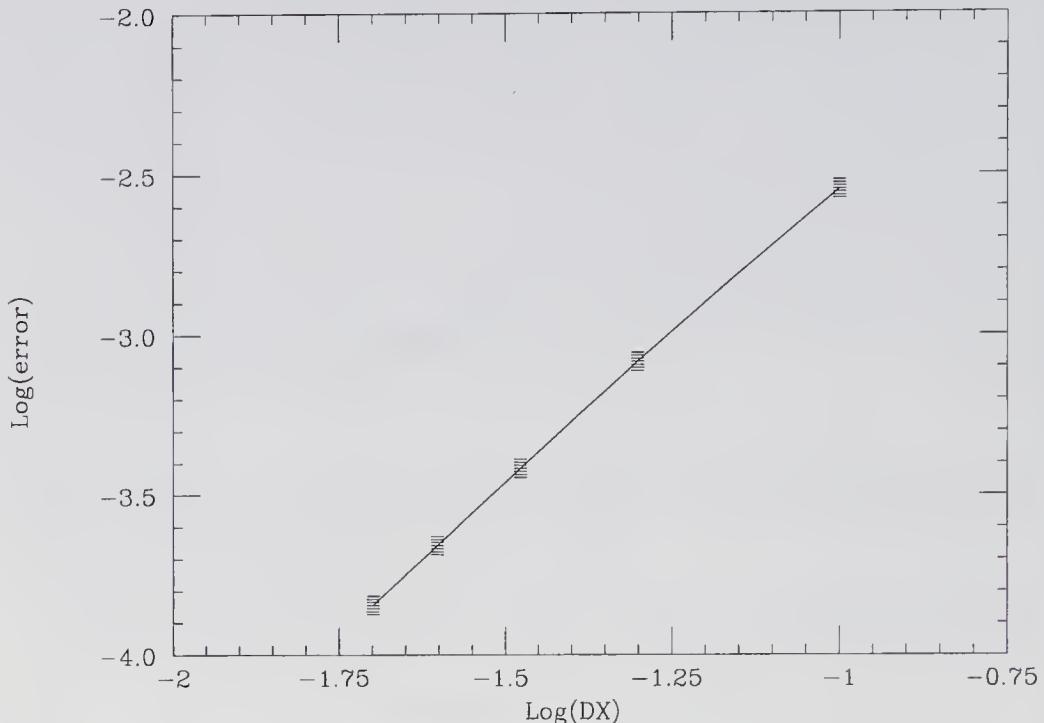


Figure 1.5 Log–log plot of the errors in Table 1.4.

entry in Table 1.4 for $N = 51$ is taken from Table 1.3 (at $x = 0.4$, $t = 0.1$). The other values of the exact error were produced by running Programs 1.2a and 1.2b for $N = 11$, 21, 31, and 41. The errors in Table 1.4 are plotted in Figure 1.5 [again as log of exact error vs. $\log DX (= 1/(N - 1))$]. Again, the slope of the line in Figure 1.5 can be estimated by using just the first and last points

$$\begin{aligned} \text{Slope} &= \frac{\log_{10}(0.002829) - \log_{10}(0.000143)}{\log_{10}(1/11 - 1) - \log_{10}(1/51 - 1)} \\ &= \frac{-2.5484 - (-3.8447)}{-1.000 - (-1.6990)} = 1.855 \end{aligned}$$

Thus, the error is of order 1.855, i.e.,

$$\text{Error} = O(DX^{1.855})$$

Again, as with the previous case in Section 1.5, when Programs 1.2a and 1.2b were executed for $N = 51$ with RELERR=1.0E-06 and ABSERR=0, RKF45 returned the error flag IFLAG=4, indicating that the maximum number of calls to subroutine FCN had been exceeded. However, for the present problem [equations (1.28) to (1.31)], this error condi-

tion was more severe; even with the statement

```
DATA MAXNFE/10000/
```

in subroutine RKF45, *a solution could not be computed beyond the initial conditions at $t = 0$* (RKF45 failed to produce a solution at $t = 0.1$, even after 10000 calls to FCN). This excessive computational effort is due, in part, to the stringent error tolerance used in Program 1.1a, which was the reason for the less stringent error tolerance subsequently used in Program 1.2a (RELERR=1.0E-04, ABSERR=1.0E-06). For the latter, the number of calls to FCN was 2757, substantially less than the 3194 required in Section 1.6 with RELERR=1.0E-06, ABSERR=0, which illustrates the possible sensitivity of the *computational efficiency* of an NUMOL code to the error tolerances specified for the ODE integrator.

In summary, we have observed that the programming of Neumann boundary conditions is similar to that for Dirichlet boundary conditions; the only essential difference is the placement of the coding relative to the calls to the spatial differentiation routine, as illustrated by Programs 1.1b and 1.2b. Next, we consider an alternative arrangement for an NUMOL code, which we shall use in the remainder of this book.

1.8 Arguments versus COMMON

In Section 1.5, we suggested that the coding of an NUMOL solution could be organized so that all of the general numerical calculations (spatial differentiation and ODE integration) could be performed in a set of routines that would not change with the solution of new problems; thus, for a new problem, the user need only write one or more routines to define the specific mathematics of the problem. We shall now consider how this might be done.

We can consider that the ODE integrator in an NUMOL code computes a solution to the problem

$$\begin{aligned} dy_1/dt &= f_1(y_1, y_2, \dots, y_n, t) \\ dy_2/dt &= f_2(y_1, y_2, \dots, y_n, t) \\ &\quad \cdot \\ &\quad \cdot \\ &\quad \cdot \\ dy_n/dt &= f_n(y_1, y_2, \dots, y_n, t) \end{aligned} \tag{1.33}$$

$$y_1(t_0) = y_{1,0}, y_2(t_0) = y_{2,0}, \dots, y_n(t_0) = y_{n,0} \quad (1.34)$$

Equations (1.33) and (1.34) are quite general, and have only the limitations that they are (1) first-order (do not contain higher-order derivatives in t , e.g., d^2y_1/dt^2); (2) initial-value, i.e., the initial conditions are all evaluated at the same value of the independent variable, $t = t_0$; and (3), *explicit ODEs* because each equation contains only one derivative that is explicitly defined mathematically.

Equations (1.33) and (1.34) can be stated in matrix form as

$$d\bar{y}/dt = \bar{f}(\bar{y}, t) \quad (1.35)$$

$$\bar{y}(t_0) = \bar{y}_0 \quad (1.36)$$

where

$$\bar{y} = [y_1 \ y_2 \ \dots \ y_n]^T$$

$$\bar{f} = [f_1 \ f_2 \ \dots \ f_n]^T$$

$$\bar{y}_0 = [y_{1,0} \ y_{2,0} \ \dots \ y_{n,0}]^T$$

(The superscript T denotes a transpose of the three column vectors \bar{y} , \bar{f} , and \bar{y}_0 , to row vectors, and is used just to facilitate writing these vectors.)

The general matrix equations (1.35) and (1.36) serve as the basis for our programming of an NUMOL code. The principal components of these equations are (1) the *dependent variable vector*, \bar{y} [$T(I)$, $I = 1, 2, \dots, N$ in the previous examples], (2) the *derivative vector*, \bar{f} [$TT(I)$, $I = 1, 2, \dots, N$ in the previous examples], and (3) the *initial condition vector*, y_0 , [$\sin(\pi x/L)$ or $\cos(\pi x/L)$ in the previous examples].

We shall now develop an NUMOL code in which the routines are divided into two groups: (1) general-purpose routines for the solution of any PDE system and (2) special-purpose routines that define a particular PDE problem. These two groups of routines will be linked through Fortran COMMON; the names of the COMMON will correspond to the independent variable t (i.e., COMMON/T/), the dependent variable vector \bar{y} (COMMON/Y/), and the derivative vector \bar{f} (COMMON/F/).

For example, for an 51-point grid in the preceding examples [based on equations (1.21) to (1.24) or (1.28) to (1.31)], the COMMON area could be

```
COMMON/T/  TIME
COMMON/Y/  T(51)
COMMON/F/  TT(51)
```

or

```
COMMON/T/ TIME
1      /Y/ T(51)
2      /F/ TT(51)
```

We shall now develop two subroutines: (1) subroutine INITAL to define the initial condition vector \bar{y}_0 and (2) subroutine DERV to define the derivative vector \bar{f} . INITAL would therefore be as shown in Program 1.3a. The coding in Program 1.3a is taken directly from Program 1.1b, with the exceptions that (1) COMMON is used in place of arguments and (2) a test for TIME=0 is not required since INITAL will be called only once at the initial value of $t(= 0)$ to set the initial condition, equation (1.22). Note in particular that as result of executing subroutine INITAL, the dependent variable vector in COMMON/Y/, T(I), is set to the initial condition; in general, this will be the purpose of INITAL (to initialize the dependent variable vector in COMMON/Y/).

The ODEs are programmed in subroutine DERV in Program 1.3b. Again, the coding is taken directly from Program 1.1b, except COMMON

Program 1.3a Subroutine INITAL for Initial Condition (1.22)

```
SUBROUTINE INITAL
C...
C... SUBROUTINE INITAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C... INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (1.21) TO
C... (1.24)
C...
COMMON/T/ TIME
1      /Y/ T(51)
2      /F/ TT(51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L, X(51)
C...
C... CALCULATE PI FOR USE IN DO LOOP 1
PI=ACOS(-1.0)
C...
C... LENGTH
L=1.0
C...
C... GRID SPACING
N=51
DX=L/FLOAT(N-1)
C...
C... INITIAL CONDITION (1.22)
DO 1 I=1,N
    X(I)=FLOAT(I-1)*DX
    T(I)=SIN(PI*X(I)/L)
1 CONTINUE
RETURN
END
```

Program 1.3b Subroutine DERV for the NUMOL Derivatives from Equation (1.21)

```

SUBROUTINE DERV
C...
C... SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24)
C...
      COMMON/T/    TIME
      1        /Y/    T(51)
      2        /F/    TT(51)
C...
C... TYPE SELECTED VARIABLES AS REAL
      REAL L, TX(51), TXX(51)
C...
C... BOUNDARY CONDITION (1.23)
      T(1)=0.
C...
C... BOUNDARY CONDITION (1.24)
      N=51
      T(N)=0.
C...
C... LENGTH
      L=1.
C...
C... DERIVATIVE TX
      CALL DSS002(0.,L,N,T,TX)
C...
C... DERIVATIVE TXX
      CALL DSS002(0.,L,N,TX,TXX)
C...
C... EQUATION (1.21)
      DO 2 I=1,N
      TT(I)=TXX(I)
2     CONTINUE
C...
C... ZERO THE BOUNDARY TIME DERIVATIVES (IN ACCORDANCE WITH BOUNDARY
C... CONDITIONS (1.23) AND (1.24))
      TT(1)=0.
      TT(N)=0.
      RETURN
      END

```

is used in place of arguments. Also, the derivatives in t at the boundaries, $TT(1)$ and $TT(N)$, are set to zero in accordance with the Dirichlet boundary conditions, $T(1) = T(N) = 0$ (the time derivative of a constant, in this case zero, is also zero). Otherwise, DO loop 2 will generally set $TT(1)$ and $TT(N)$ to nonzero values, and RKF45 will use these nonzero derivatives to move $T(1)$ and $T(N)$ away from zero (for at most the interval in t along the solution before the next call to DERV, when $T(1)$ and $T(N)$ are again set to zero; this effect is small, but still, zeroing the time derivatives at the boundaries, when appropriate, is recommended).

Also, we need a routine to print the NUMOL solution as shown in Program 1.3c. A separate subroutine for printing is used so that the coding will be divided into general-purpose routines [main program PRO1P3 (see Program 1.3d) and RKF45] and routines specific to the problem (INITAL, DERV and PRINT). Subroutine PRINT has two arguments, NI and NO,

Program 1.3c Subroutine PRINT and Function EXACT for Equations (1.21) to (1.24)

```

SUBROUTINE PRINT(NI,NO)
C...
C... SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PRO1P3 TO PRINT THE
C... NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24)
C...
COMMON/T/    TIME
1           /Y/   T(51)
2           /F/   TT(51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL TE(51), DIFF(51)
C...
C... CALCULATE THE EXACT SOLUTION, AND THE DIFFERENCE BETWEEN THE
C... NUMOL AND EXACT SOLUTIONS
N=51
DO 3 I=1,N,10
TE(I)=EXACT(I,TIME)
DIFF(I)=T(I)-TE(I)
3 CONTINUE
C...
C... PRINT THE NUMOL SOLUTION TO EQUATIONS (1.21) TO (1.24)
WRITE(NO,2)TIME,(T(I),I=1,N,10),
1           (TE(I),I=1,N,10),
2           (DIFF(I),I=1,N,10)
2 FORMAT(' TIME = ',F6.2,/,15X,', X=0',5X,'X=0.2',5X,'X=0.4',5X,
1           'X=0.6',5X,'X=0.8',5X,' X=1',/,,
2           ', T(X,T)',6F10.6,/,,
3           ', TE(X,T)',6F10.6,/,,
4           ', DIFF(X,T)',6F10.6,/)
RETURN
END

REAL FUNCTION EXACT(I,T)
C...
C... FUNCTION EXACT COMPUTES THE EXACT SOLUTION TO EQUATIONS (1.21)
C... TO (1.24), I.E., EQUATION (1.25), AT GRID INDEX I AND TIME T
C...
C... NUMBER OF GRID POINTS
PARAMETER (N=51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C... PI FOR USE IN THE EXACT SOLUTION
PI=ACOS(-1.)
C...
C... LENGTH
L=1.
C...
C... GRID SPACING
DX=L/FLOAT(N-1)
C...
C... X FOR WHICH THE EXACT SOLUTION IS TO BE COMPUTED
X=FLOAT(I-1)*DX
C...
C... EXACT SOLUTION AT X AND T
EXACT=EXP((-PI**2/L**2)*T)*SIN(PI*X/L)
RETURN
END

```

the input and output unit (device) numbers for Fortran READ/WRITE statements; these unit numbers are set in main program PRO1P3 and passed to subroutine PRINT (note how NO is used in the WRITE statements).

Main program PRO1P3, which is a variant of PRO1P1 (Program 1.1a), with the coding specific to the problem removed, is listed in Program 1.3d. Several features of PRO1P3 should be noted:

- (1) None of the coding is specific to the problem, in this case equations (1.21) to (1.24).
- (2) A small subroutine, FCN, is called by subroutine RKF45 (as before in Program 1.1a); FCN in turn calls the derivative subroutine DERV.
- (3) PRO1P3 reads a data file, DATA, to define: (a) the initial, final, and print interval values of t (T, TF and TP), (b) the number of ODEs (NEQN) and the error tolerance (ERROR). Thus parameters specific to the problem are read as data rather than set in the main program.
- (4) PRO1P3 prints a summary of the data file for user verification.
- (5) PRO1P3 calls subroutine INITAL to define the initial condition, equation (1.22), then calls subroutine PRINT to print the initial condition.
- (6) PRO1P3 calls RKF45 via a loop (as in Program 1.1a) to step through $t = 0, 0.1, \dots, 0.5$. At each value of t , a check is made on the error flag, IFLAG, to determine whether the integration of the ODEs progressed satisfactorily; if not, an error message is printed and the run is terminated. If the integration progressed satisfactorily, PRINT is called to print the solution.
- (7) Before the first call to RKF45, the two error tolerances are set to ERROR, i.e., RELERR=ERROR and ABSERR=ERROR; other approaches for specifying the RKF45 error tolerances are, of course, easily programmed.
- (8) PRO1P3 has a multiple run capability controlled through the variable NORUN in COMMON/T/ (this will be used in subsequent examples). Also, a run termination facility is available through variable NSTOP in COMMON/T/ (this will be used in subsequent examples).

Program 1.3d Main Program PRO1P3 and Interface Subroutine FCN

```

PROGRAM PRO1P3
C...
C...  PROGRAM PRO1P3 CALLS: (1) SUBROUTINE INITAL TO DEFINE THE ODE
C...  INITIAL CONDITIONS, (2) SUBROUTINE RKF45 TO INTEGRATE THE ODES,
C...  AND (3) SUBROUTINE PRINT TO PRINT THE SOLUTION.
C...
C...  THE FOLLOWING CODING IS FOR 250 ODES.  IF MORE ODES ARE TO BE INTE-
continues

```

```

C... GRATED, ALL OF THE 250'S SHOULD BE CHANGED TO THE REQUIRED NUMBER
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/      Y(250)
2      /F/      F(250)
C...
C... THE NUMBER OF DIFFERENTIAL EQUATIONS IS IN COMMON/N/ FOR USE IN
C... SUBROUTINE FCN
COMMON/N/      NEQN
C...
C... COMMON AREA TO PROVIDE THE INPUT/OUTPUT UNIT NUMBERS TO OTHER
C... SUBROUTINES
COMMON/IO/      NI,      NO
C...
C... ABSOLUTE DIMENSIONING OF THE ARRAYS REQUIRED BY RKF45
DIMENSION YV(250), WORK(1000), IWORK(5)
C...
C... EXTERNAL THE DERIVATIVE ROUTINE CALLED BY RKF45
EXTERNAL FCN
C...
C... ARRAY FOR THE TITLE (FIRST LINE OF DATA), CHARACTERS END OF RUNS
CHARACTER TITLE(20)*4, ENDRUN(3)*4
C...
C... DEFINE THE CHARACTERS END OF RUNS
DATA ENDRUN/'END ','OF R','UNS '/
C...
C... DEFINE THE INPUT/OUTPUT UNIT NUMBERS
NI=5
NO=6
C...
C... OPEN INPUT AND OUTPUT FILES
OPEN(NI,FILE='DATA')
OPEN(NO,FILE='OUTPUT')
C...
C... INITIALIZE THE RUN COUNTER
NORUN=0
C...
C... BEGIN A RUN
1 NORUN=NORUN+1
C...
C... INITIALIZE THE RUN TERMINATION VARIABLE
NSTOP=0
C...
C... READ THE FIRST LINE OF DATA
READ(NI,1000,END=999)(TITLE(I),I=1,20)
C...
C... TEST FOR END OF RUNS IN THE DATA
DO 2 I=1,3
IF(TITLE(I).NE.ENDRUN(I))GO TO 3
2 CONTINUE
C...
C... AN END OF RUNS HAS BEEN READ, SO TERMINATE EXECUTION
999 STOP
C...
C... READ THE SECOND LINE OF DATA
3 READ(NI,1001,END=999)TO,TF,TP
C...
C... READ THE THIRD LINE OF DATA
READ(NI,1002,END=999)NEQN,ERROR
C...
C... PRINT A DATA SUMMARY
WRITE(NO,1003)NORUN,(TITLE(I),I=1,20),
1                      TO,TF,TP,
2                      NEQN,ERROR

```

continues

```

C...
C...   INITIALIZE TIME
      T=T0
C...
C...   SET THE INITIAL CONDITIONS
      CALL INITAL
C...
C...   SET THE INITIAL DERIVATIVES (FOR POSSIBLE PRINTING)
      CALL DERV
C...
C...   PRINT THE INITIAL CONDITIONS
      CALL PRINT(NI,NO)
C...
C...   SET THE INITIAL CONDITIONS FOR SUBROUTINE RKF45
      TV=T0
      DO 5 I=1,NEQN
          YV(I)=Y(I)
      5  CONTINUE
C...
C...   SET THE PARAMETERS FOR SUBROUTINE RKF45
      RELERR=ERROR
      ABSERR=ERROR
      IFLAG=1
      TOUT=T0+TP
C...
C...   CALL SUBROUTINE RKF45 TO START THE SOLUTION FROM THE INITIAL
C...   CONDITION (IFLAG = 1) OR COMPUTE THE SOLUTION TO THE NEXT PRINT
C...   POINT (IFLAG = 2)
      4  CALL RKF45(FCN,NEQN,YV,TV,TOUT,RELERR,ABSERR,IFLAG,WORK,IWORK)
C...
C...   PRINT THE SOLUTION AT THE NEXT PRINT POINT
      T=TV
      DO 6 I=1,NEQN
          Y(I)=YV(I)
      6  CONTINUE
      CALL DERV
      CALL PRINT(NI,NO)
C...
C...   TEST FOR AN ERROR CONDITION
      IF(IFLAG.NE.2)THEN
C...
C...       PRINT A MESSAGE INDICATING AN ERROR CONDITION
          WRITE(NO,1004)IFLAG
C...
C...       GO ON TO THE NEXT RUN
          GO TO 1
      END IF
C...
C...   CHECK FOR A RUN TERMINATION
      IF(NSTOP.NE.0)GO TO 1
C...
C...   CHECK FOR THE END OF THE RUN
      TOUT=TV+TP
      IF(TV.LT.(TF-0.5*TP))GO TO 4
C...
C...   THE CURRENT RUN IS COMPLETE, SO GO ON TO THE NEXT RUN
      GO TO 1
C...
C... ****
C...

```

continues

```

C... FORMATS
C...
1000 FORMAT(20A4)
1001 FORMAT(3E10.0)
1002 FORMAT(I5,20X,E10.0)
1003 FORMAT(1H1,
 1 , 'RUN NO. - ',I3,2X,20A4,//,
 2 , 'INITIAL T - ',E10.3,//,
 3 , 'FINAL T - ',E10.3,//,
 4 , 'PRINT T - ',E10.3,//,
 5 , 'NUMBER OF DIFFERENTIAL EQUATIONS - ',I3,//,
 6 , 'MAXIMUM INTEGRATION ERROR - ',E10.3,//,
 7 1H1)
1004 FORMAT(1H ,//,'IFLAG = ',I3,//,
 1 ' INDICATING AN INTEGRATION ERROR, SO THE CURRENT RUN' ,//,
 2 ' IS TERMINATED. PLEASE REFER TO THE DOCUMENTATION FOR' ,//,
 3 ' SUBROUTINE',//,25X,'RKF45',//,
 4 ' FOR AN EXPLANATION OF THESE ERROR INDICATORS' ) )
END

SUBROUTINE FCN(TV,YV,YDOT)
C...
C... SUBROUTINE FCN IS AN INTERFACE ROUTINE BETWEEN SUBROUTINES RKF45
C... AND DERV
C...
C... NOTE THAT THE SIZE OF ARRAYS Y AND F IN THE FOLLOWING COMMON AREA
C... IS ACTUALLY SET BY THE CORRESPONDING COMMON STATEMENT IN MAIN
C... PROGRAM PRO1P3
COMMON/T/           T,      NSTOP,      NORUN
 1     /Y/           Y(1)
 2     /F/           F(1)
C...
C... THE NUMBER OF DIFFERENTIAL EQUATIONS IS AVAILABLE THROUGH COMMON
C... /N/
COMMON/N/          NEQN
C...
C... ABSOLUTE DIMENSION THE DEPENDENT VARIABLE, DERIVATIVE VECTORS
REAL YV(250), YDOT(250)
C...
C... TRANSFER THE INDEPENDENT VARIABLE, DEPENDENT VARIABLE VECTOR
C... FOR USE IN SUBROUTINE DERV
T=TV
DO 1 I=1,NEQN
Y(I)=YV(I)
1 CONTINUE
C...
C... EVALUATE THE DERIVATIVE VECTOR
CALL DERV
C...
C... TRANSFER THE DERIVATIVE VECTOR FOR USE BY SUBROUTINE RKF45
DO 2 I=1,NEQN
YDOT(I)=F(I)
2 CONTINUE
RETURN
END

```

In summary, PRO1P3 is a general-purpose main program to integrate ODEs via subroutine RKF45 that, in principle, can be applied to any NUMOL problem for which the user provides subroutines INITIAL, DERV, PRINT, and a data file, DATA, to define a specific PDE problem.

Program 1.3e File DATA Read My Main Program PRO1P3

```

NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24)
0.          0.5          0.1
51           0.000001
END OF RUNS

```

Finally, file DATA must be included with the execution of Programs 1.3a to 1.3d; see Program 1.3e. The four lines of data include

- (1) Line 1—a documentation title that is merely read and printed.
- (2) Line 2—the initial, final, and print interval values of t (variables T0, TF, TP), read with an 3E10.0 format (see the formats at the end of PRO1P3).
- (3) Line 3—the number of ODEs (NEQN), and the error tolerance (ERROR), read with an I5,20X,E10.0 format.
- (4) Line 4—the characters END OF RUNS, which are detected by main program PRO1P3 to terminate execution.

Of course, other formats could be used for reading the data, e.g., READ(NI,*) in place of READ(NI,1000) for unformatted input.

This completes the program. Programs 1.3a to 1.3e might seem more complicated than Programs 1.1a and 1.1b, but, again, they have the major advantage of separating the general-purpose and problem-specific coding. We will therefore use this approach in all of the subsequent examples.

The output from Programs 1.3a to 1.3e is given below in Table 1.5a and Table 1.5b. First, a data summary is printed by PRO1P3 to ensure that the data file has been read correctly.

The output printed by subroutine PRINT is shown in Table 1.5b, which, as expected, is identical to Table 1.1.

We have in this final section developed, by an example of the NUMOL code for equations (1.21) to (1.24), the general approach we shall follow in all of the remaining examples in this book; only minor variations on this basic approach will be used; e.g., all constants could be initialized in INITIAL, which is called only once, and these initialized constants could

Table 1.5a Data Summary from PRO1P3

```

RUN NO. - 1 NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24)

INITIAL T - 0.000E+00

FINAL T - .500E+00

PRINT T - .100E+00

NUMBER OF DIFFERENTIAL EQUATIONS - 51

MAXIMUM INTEGRATION ERROR - .100E-05

```

Table 1.5b Numerical Solution from Subroutine PRINT

TIME =	0.00	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.587785	.951057	.951057	.587785	.000000	
TE(X,T)	0.000000	.587785	.951057	.951057	.587785	.000000	
DIFF(X,T)	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
TIME =	.10	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.219357	.354927	.354927	.219357	.000000	
TE(X,T)	0.000000	.219072	.354466	.354466	.219072	.000000	
DIFF(X,T)	0.000000	.000285	.000460	.000460	.000285	.000000	
TIME =	.20	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.081862	.132456	.132456	.081862	.000000	
TE(X,T)	0.000000	.081650	.132112	.132112	.081650	.000000	
DIFF(X,T)	0.000000	.000212	.000343	.000343	.000212	.000000	
TIME =	.30	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.030550	.049431	.049431	.030550	.000000	
TE(X,T)	0.000000	.030432	.049239	.049239	.030432	.000000	
DIFF(X,T)	0.000000	.000119	.000192	.000192	.000119	.000000	
TIME =	.40	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.011401	.018447	.018447	.011401	.000000	
TE(X,T)	0.000000	.011342	.018352	.018352	.011342	.000000	
DIFF(X,T)	0.000000	.000059	.000096	.000096	.000059	.000000	
TIME =	.50	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.004255	.006884	.006884	.004255	.000000	
TE(X,T)	0.000000	.004227	.006840	.006840	.004227	.000000	
DIFF(X,T)	0.000000	.000028	.000045	.000045	.000028	.000000	

then be passed to DERV, PRINT, and any subordinate routines (e.g., EXACT) through COMMON.

Of course, many alternative approaches to the organization of the code are possible, but we have found that this approach is sufficiently flexible to accommodate any PDE problem that can be approached via the NUMOL. For example, the ODE integrator can be changed merely by replacing the call to RKF45 in PRO1P3 with a call to any other ODE integrator, and in fact, we shall consider the use of a series of quality ODE integrators in subsequent chapters. We recommend that the reader run Programs 1.3a to 1.3e to ensure that the output in Tables 1.5a and 1.5b can be reproduced.

Also, we should again point out that the NUMOL is a methodology by which all of the knowledge, algorithms, and software for initial-value ODEs can, in principle, be applied to PDEs. Thus, we can make use of an extensive body of knowledge and procedures for ODEs in the solution of a spectrum of PDEs. The NUMOL is flexible and open-ended; it has essentially become a philosophy in approaching PDE problems, and its utility is

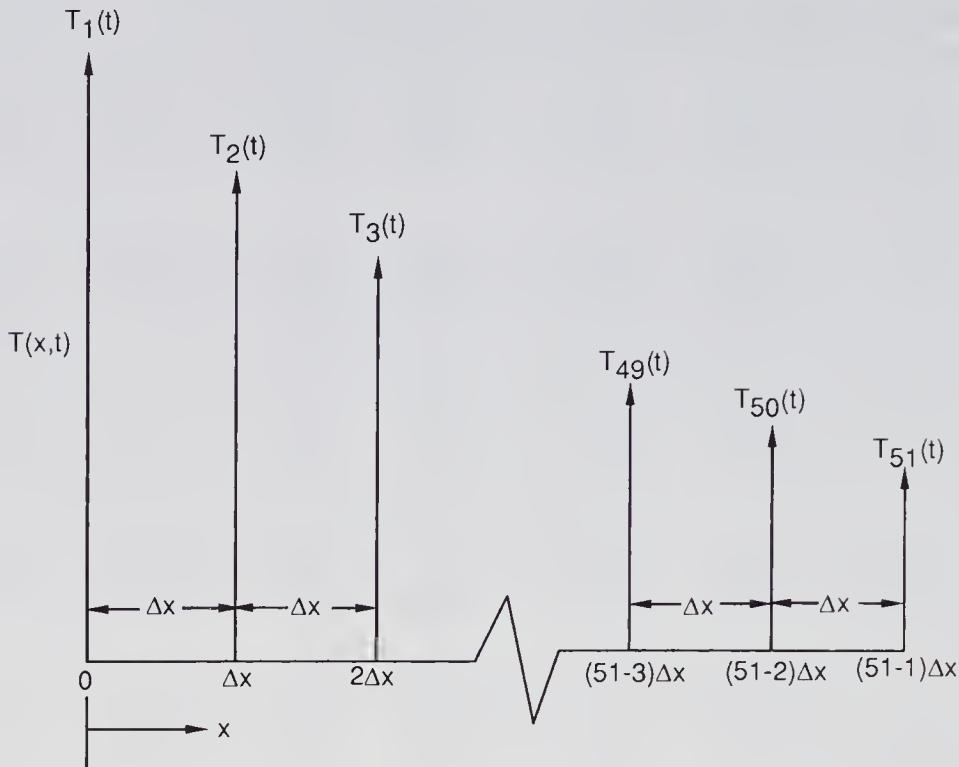


Figure 1.6 The origin of the name the *numerical method of lines*.

limited only by the imagination of the user in putting together combinations of techniques through available quality software. Speaking of software, since we have used exclusively transportable Fortran 77 in the approach discussed previously, the NUMOL code that is produced can be run on any computer with a Fortran 77 compiler; also, any Fortran-callable library can be used, e.g., calls to routines for special functions in DERV, calls to graphics in PRINT or writing an ASCII file of the NUMOL solution in PRINT that can then be sent to any graphics software that accepts an ASCII file. Going one step further, the basic ideas discussed in this chapter could, at least in principle, be implemented with any other standard computer language with some computational capabilities (e.g., via a Pascal or C compiler).

Finally, we should indicate the apparent origin of the name *the numerical method of lines*. If, for example, we consider the solution to equations (1.21) to (1.24) in Programs 1.3a to 1.3e, we can envision the evolution of the *numerical solution*, $T(x, t)$, from $t = 0$ along *lines of constant x* (at each of the 51 grid points) as illustrated in Figure 1.6. The solution at a particular value of t is given by the corresponding values of $T(x, t)$ along each of the lines.

References

- (1.1) Forsythe, G. E., M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations*, Prentice Hall, Englewood Cliffs, NJ, 1977.
- (1.2) Kahaner, D., C. Moler, and S. Nash, *Numerical Methods and Software*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
- (1.3) Gear, C. W., *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

Problems

- (1.1)** Modify Programs 1.1a to 1.1c to compute a NUMOL solution to the problem

$$T_t = T_{xx} \quad (1.1a)$$

$$T(x, 0) = x \quad (1.1b)$$

$$T(0, t) = 0, T(L, t) = 1 \quad (1.1c)(1.1d)$$

Compare the NUMOL and exact solutions.

- (1.2)** Modify Programs 1.3a to 1.3c to compute a NUMOL solution to the problem

$$T_t = T_{xx} \quad (1.2a)$$

$$T(x, 0) = x \quad (1.2b)$$

$$T_x(0, t) = 1, T_x(L, t) = 1 \quad (1.2c)(1.2d)$$

Compare the NUMOL and exact solutions.

- (1.3)** Initial-value ODEs can be integrated numerically by the *Euler method*. For equations (1.21) to (1.24), some Fortran code to do this would look like

```

      CALL FCN(TIME, T, TT)
      DO1 I=1, N
      T(I)=T(I)+TT(I)*H
      1 CONTINUE
    
```

where H is the integration step size. Write a subroutine EULER(FCN,H,N,TIME,T,TT) that could be used in place of RKF45 to compute an NUMOL solution to equations (1.21) to (1.24). EULER can contain the preceding code, and should be called from main program PRO1P1 (Program 1.1a). Subroutine

FCN and function EXACT in Program 1.1b can remain unchanged (FCN is called from Euler as an external). It will be necessary to select H before calling EULER.

- (1.4) Execute the program of problem (1.3) for a series of integration steps sizes (H). Plot the error in the NUMOL solution vs. H [log(error) vs. log(H) at a selected t and x]. From the slope of this plot, what can you infer about the order of the Euler method?
- (1.5) The *modified Euler method* is an improvement in the Euler method (it gives better accuracy). It could be implemented for equations (1.21) to (1.24) with code something like

```

CALL FCN(TIME,T,TT)
DO1 I=1,N
T(I)=T(I)+TT(I)*H
1      CONTINUE
CALL FCN(TIME,T,TTP)
DO2 I=1,N
T(I)=T(I)+(TT(I)+TTP(I))*H/2.0
2      CONTINUE

```

Extend subroutine EULER in problem (1.3) to implement the modified Euler method. TTP is an additional array to store the derivatives in t computed from the second call to FCN.

- (1.6) Repeat problem (1.4) for the modified Euler method of problem (1.5).
- (1.7) What is the geometric interpretation of the Euler method and modified Euler method for a single ODE $dy/dt = f(y, t)$?
- (1.8) Verify that equation (1.32) is a solution to equations (1.28) to (1.31).

2

Some Applications of the Numerical Method of Lines

In Chapter 1, we gave an introduction to the NUMOL covering most of the basic concepts. We now proceed to demonstrate the flexibility and versatility of the NUMOL by applying it to a series of applications of increasing complexity. Our intention, also, is to illustrate more advanced programming techniques by these examples.

2.1 Additional COMMON

In Section 1.8 we developed an approach to the programming of a NUMOL code in which we used general-purpose and problem-specific routines. Communication between these two sets of routines was through COMMON/T/, /Y/, and /F/ corresponding to the elements of the general ODE system

$$\begin{aligned} d\bar{y}/dt &= \bar{f}(\bar{y}, t) \\ \bar{y}(t_0) &= \bar{y}_0 \end{aligned}$$

In particular, \bar{y}_0 was defined in subroutine INITIAL, $\bar{f}(\bar{y}, t)$ was defined in subroutine DERV, and the solution was printed in subroutine PRINT (and again, t was passed through COMMON/T/, \bar{y} was passed through COMMON/Y/, and \bar{f} was passed through COMMON/F/).

The example of equations (1.21) to (1.24) programmed in Programs 1.3a to 1.3e also illustrated that several constants were used in all three subroutines (INITAL, DERV, and PRINT). This suggests that these constants could be set in subroutine INITAL, which is called only once, then passed through COMMON for use in the other subroutines. This has the advantages of simplicity and efficiency (the amount of repetitive coding is reduced and the constants are defined only once rather than each time DERV and PRINT are called). The only thing we must observe in using additional COMMON to share constants between the three subroutines is that the reserved names /T/, /Y/, and /F/ should not be used for this purpose, since by putting additional entries into these COMMON areas, we will interfere with the operation of the ODE integrator. Also, any variables (which change with t) can be put in the additional COMMON, so that, for example, spatial derivatives computed in DERV (TX and TXX) can be passed to PRINT for printing. This is a very effective technique for debugging a NUMOL code, and for getting a comprehensive picture of the NUMOL solution; we shall illustrate these ideas in subsequent examples.

First, we list in Program 2.1 subroutines INITAL, DERV, and

Program 2.1 Subroutines INITAL, DERV, and PRINT with Additional COMMON/R and /I/

```

SUBROUTINE INITAL
C...
C...  SUBROUTINE INITAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C...  INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (1.21) TO
C...  (1.24)
C...
C...  COMMON/T/    TIME
1      /Y/    T(51)
2      /F/    TT(51)
3      /R/    X(51),    TX(51),    TXX(51),    TE(51),    DIFF(51),
4                  L,        PI,        DX
5      /I/    N
C...
C...  TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C...  CALCULATE PI FOR USE IN DO LOOP 1
PI=ACOS(-1.0)
C...
C...  LENGTH
L=1.0
C...
C...  GRID SPACING
N=51
DX=L/FLOAT(N-1)
C...
C...  INITIAL CONDITION (1.22)
DO 1 I=1,N
     X(I)=FLOAT(I-1)*DX
     T(I)=SIN(PI*X(I)/L)
1

```

continues

```

1      CONTINUE
      RETURN
      END

      SUBROUTINE DERV
C...
C...  SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C...  ODES IN THE NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24)
C...
C...  COMMON/T/    TIME
1      /Y/    T(51)
2      /F/    TT(51)
3      /R/    X(51),    TX(51),    TXX(51),    TE(51),    DIFF(51),
4          L,        PI,        DX
5      /I/    N

C...
C...  TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C...  BOUNDARY CONDITION (1.23)
T(1)=0.
TT(1)=0.

C...
C...  BOUNDARY CONDITION (1.24)
T(N)=0.
TT(N)=0.

C...
C...  DERIVATIVE TX
CALL DSS002(0.,L,N,T,TX)
C...
C...  DERIVATIVE TXX
CALL DSS002(0.,L,N,TX,TXX)
C...
C...  EQUATION (1.21)
DO 2 I=2,N-1
TT(I)=TXX(I)
2      CONTINUE
      RETURN
      END

      SUBROUTINE PRINT(NI,NO)
C...
C...  SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PRO1P3 TO PRINT THE
C...  NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24)
C...
C...  COMMON/T/    TIME
1      /Y/    T(51)
2      /F/    TT(51)
3      /R/    X(51),    TX(51),    TXX(51),    TE(51),    DIFF(51),
4          L,        PI,        DX
5      /I/    N

C...
C...  TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C...  CALCULATE THE EXACT SOLUTION, AND THE DIFFERENCE BETWEEN THE
C...  NUMOL AND EXACT SOLUTIONS
DO 3 I=1,N,10
TE(I)=EXACT(I)
DIFF(I)=T(I)-TE(I)
3      CONTINUE
C...
C...  PRINT THE NUMOL SOLUTION TO EQUATIONS (1.21) TO (1.24)
WRITE(NO,2)TIME,(T(I),I=1,N,10),

```

continues

```

1           (TE(I), I=1, N, 10),
2           (DIFF(I), I=1, N, 10)
2 FORMAT( ' TIME = ', F6.2,/, 15X, ' X=0', 5X, ' X=0.2', 5X, ' X=0.4', 5X,
1                           ' X=0.6', 5X, ' X=0.8', 5X, ' X=1', /,
2                           ' T(X,T)', 6F10.6, /,
3                           ' TE(X,T)', 6F10.6, /,
4                           ' DIFF(X,T)', 6F10.6, /)
      RETURN
      END

      REAL FUNCTION EXACT(I)
C...
C...  FUNCTION EXACT COMPUTES THE EXACT SOLUTION TO EQUATIONS (1.21)
C...  TO (1.24), I.E., EQUATION (1.25), AT GRID INDEX I AND TIME T
C...
      COMMON/T/,    TIME
      /Y/   T(51)
      /F/   TT(51)
      /R/   X(51),    TX(51),    TXX(51),    TE(51),  DIFF(51),
      /L/   PI,        DX
      /I/   N
C...
C...  TYPE SELECTED VARIABLES AS REAL
      REAL L
C...
C...  EXACT SOLUTION AT X AND T
      EXACT=EXP((-PI**2/L**2)*TIME)*SIN(PI*X(I)/L)
      RETURN
      END

```

PRINT from Programs 1.3a to 1.3c, but modified slightly to include two additional COMMON areas, /R/ for real constants and /I/ for integer constants (the names of the COMMON blocks are arbitrary with the exception of /T/, /Y/, and /F/). The associated data are again as shown in Table 2.1. The output from Program 2.1 (with the main program Program 1.3d) is the same as in Table 1.5b.

Note that COMMON/R/ is used to both dimension arrays X(51) to DIFF(51) and pass these arrays between the subroutines, notably to PRINT where they can be printed; also plotting from PRINT in subsequent examples will provide a means for graphically displaying the NUMOL solution.

Table 2.1 Data for Program 2.1

NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24)
0. 0.5 0.1
51 0.000001
END OF RUNS

2.2 Inconsistent Initial and Boundary Conditions

We now consider a variant of the problem defined by equations (1.21) to (1.24)

$$T_t = T_{xx} \quad (2.1)$$

$$T(x, 0) = 0, \quad 0 \leq x \leq 1 \quad (2.2)$$

$$T(0, t) = 0, \quad T(1, t) = 1, \quad t > 0 \quad (2.3) \quad (2.4)$$

Note that initial condition (2.2) specifies $T(1, 0) = 0$ (at $x = 1, t = 0$), while boundary condition (2.4) specifies $T(1, t) = 1$, (at $x = 1, t > 0$) so that $T(x, t)$ at $x = 1$ undergoes a finite jump from 0 to 1 when $t = 0$. In other words, a discontinuity is specified at $x = 1$, or the *initial and boundary conditions are inconsistent* in the sense of not specifying a smooth transition from the initial condition to the boundary condition. As might be expected, some care must be given to computing a NUMOL solution when such a discontinuity occurs. Actually, for some types of PDE, such as equation (2.1), this type of discontinuity does not present a problem (its effect soon damps out with increasing values of t), while for other types of PDE, it becomes a major part of the solution; we shall defer the discussion of the latter case to Chapter 3.

Equations (2.1) to (2.4) are programmed in the subroutines INITAL, DERV, and PRINT in Program 2.2 (plus data), which again were executed with the main program, Program 1.3d. Note that two sets of data are provided for two runs of Program 2.2; for the first set of data, t runs to 0.25, and the solution is printed at intervals of 0.05 while for the second set of data, t runs to 1.0 and is printed at intervals of 0.2. These data illustrate the multiple run capability of main program Program 1.3d.

Program 2.2 Subroutines INITAL, DERV, PRINT, and Data for Equations (2.1) to (2.4)

```

SUBROUTINE INITAL
C...
C...: SUBROUTINE INITAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C...: INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (2.1) TO
C...: (2.4)
C...
COMMON/T/    TIME
1      /Y/    T(51)
2      /F/    TT(51)
3      /R/    X(51),    TX(51),    TXX(51),
4                  L,        DX
5      /I/    N

```

continues

```

C...
C...  TYPE SELECTED VARIABLES AS REAL
      REAL L
C...
C...  LENGTH
      L=1.0
C...
C...  GRID SPACING
      N=51
      DX=L/FLOAT(N-1)
C...
C...  INITIAL CONDITION (2.2)
      DO 1 I=1,N
          X(I)=FLOAT(I-1)*DX
          T(I)=0.
1     CONTINUE
      RETURN
      END

      SUBROUTINE DERV
C...
C...  SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C...  ODES IN THE NUMOL SOLUTION OF EQUATIONS (2.1) TO (2.4)
C...
      COMMON/T/    TIME
      1        /Y/    T(51)
      2        /F/    TT(51)
      3        /R/    X(51),    TX(51),    TXX(51),
      4                      L,           DX
      5        /I/     N
C...
C...  TYPE SELECTED VARIABLES AS REAL
      REAL L
C...
C...  BOUNDARY CONDITION (2.3)
      T(1)=0.
      TT(1)=0.
C...
C...  BOUNDARY CONDITION (2.4)
      T(N)=1.
      TT(N)=0.
C...
C...  DERIVATIVE TX
      CALL DSS002(0.,L,N,T,TX)
C...
C...  DERIVATIVE TXX
      CALL DSS002(0.,L,N,TX,TXX)
C...
C...  EQUATION (2.1)
      DO 2 I=2,N-1
          TT(I)=TXX(I)
2     CONTINUE
      RETURN
      END

      SUBROUTINE PRINT(NI,NO)
C...
C...  SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PR01P3 TO PRINT THE
C...  NUMOL SOLUTION OF EQUATIONS (2.1) TO (2.4)
C...
      COMMON/T/    TIME
      1        /Y/    T(51)
      2        /F/    TT(51)
      3        /R/    X(51),    TX(51),    TXX(51),
      4                      L,           DX
      5        /I/     N

```

continues

```

C...
C...  TYPE SELECTED VARIABLES AS REAL
      REAL L
C...
C...  PRINT THE NUMOL SOLUTION TO EQUATIONS (2.1) TO (2.4)
      WRITE(NO,2)TIME,(T(I),I=1,N,10)
2      FORMAT(' TIME = ',F6.2,/,15X,' X=0',5X,'X=0.2',5X,'X=0.4',5X,
1                           'X=0.6',5X,'X=0.8',5X,' X=1',/,,
2                           ' T(X,T)',6F10.6,/)
      RETURN
END

NUMOL SOLUTION OF EQUATIONS (2.1) TO (2.4), T = 0, 0.05,..., 0.25
0.          0.25      0.05
51          0.000001
NUMOL SOLUTION OF EQUATIONS (2.1) TO (2.4), T = 0, 0.2,..., 1.0
0.          1.0       0.2
51          0.000001
END OF RUNS

```

The output from Program 2.2 is shown in Table 2.2. The main program, Program 1.3d, will continue to compute solutions until it reads the characters END OF RUNS from the data file, DATA (the reader should study the coding in Program 1.3d to see how this is done).

Table 2.2 Numerical Solution from Program 2.2

RUN NO.	-	1	NUMOL SOLUTION OF EQUATIONS (2.1) TO (2.4), T = 0, 0.05,				
							..., 0.25
INITIAL T	-	0.000E+00					
FINAL T	-	.250E+00					
PRINT T	-	.500E-01					
NUMBER OF DIFFERENTIAL EQUATIONS	-	51					
MAXIMUM INTEGRATION ERROR	-	.100E-05					
TIME =	0.00	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
TIME =	.05	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.011539	.058201	.206086	.526924	1.000000	
TIME =	.10	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.066439	.178030	.370697	.654574	1.000000	
TIME =	.15	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.115635	.262663	.461634	.713970	1.000000	
TIME =	.20	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.148070	.315859	.515715	.747837	1.000000	

continues

```

TIME = .25
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 0.000000  .168222  .348565  .548545  .768189  1.000000

RUN NO. - 2 NUMOL SOLUTION OF EQUATIONS (2.1) TO (2.4), T = 0, 0.2,
          . . . , 1.0

INITIAL T - 0.000E+00
FINAL T - .100E+01
PRINT T - .200E+00
NUMBER OF DIFFERENTIAL EQUATIONS - 51
MAXIMUM INTEGRATION ERROR - .100E-05

TIME = 0.00
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 0.000000  0.000000  0.000000  0.000000  0.000000  1.000000

TIME = .20
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 0.000000  .148070  .315859  .515715  .747837  1.000000

TIME = .40
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 0.000000  .192751  .388271  .588271  .792751  1.000000

TIME = .60
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 0.000000  .198990  .398367  .598367  .798990  1.000000

TIME = .80
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 0.000000  .199859  .399773  .599773  .799859  1.000000

TIME = 1.00
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 0.000000  .199980  .399968  .599968  .799980  1.000000

```

In the present case, the first run was executed so that the solution would be clearly defined at different values of t as indicated in Figure 2.1 (a plot of the solution from Run 1); the second run was executed further in t so that the solution closely approached the infinite-time solution

$$T(x, \infty) = x \quad (2.5)$$

The output of Run 2 indicates a close correspondence to equation (2.5). Also, the solution at $t = 0$ indicates that boundary condition (2.4) had already been imposed, which is due to the call to DERV in main program Program 1.3d before the initial call to PRINT [so that the line $T(N) = 1.$ is executed in DERV before the initial conditions at $t = 0$ are printed in PRINT]. Of course, a test for $t = 0$ could be put in PRINT to execute $T(N) = 0.$, but this would not affect the numerical solution for $t > 0$.

The exact solution was not computed and compared with the NUMOL solution in Program 2.2. We can mention two reasons why this

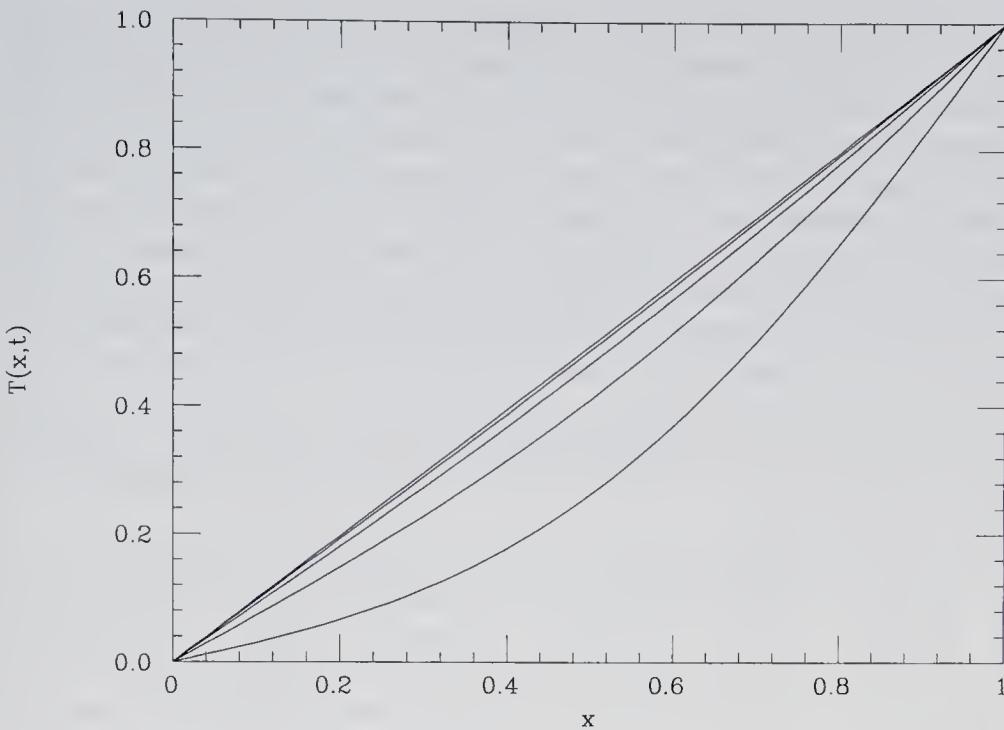


Figure 2.1 Plotted numerical solution from Program 2.2, Run 1.

was not done: (1) although the exact solution is not difficult to derive, it does take the form of an infinite series, obtained, for example, from the separation of variables and a Fourier series; thus, it is not as straightforward as equation (1.25) to derive; and (2) the infinite-series solution will probably not be the standard we seek for evaluating the NUMOL solution, since, for a problem with a discontinuity, an infinite-series solution will exhibit an oscillation in the neighborhood of the discontinuity (near $x = 1$), the Gibbs' phenomenon, which will occur no matter how many terms are used in the infinite series; a possible alternative is to derive a short-time solution that does not lead to an infinite series, but this would not be valid over the full range of t to $t = \infty$.

In any case, this example does illustrate how the effect of the discontinuity at $x = 1$ is damped, and has little effect beyond $t = 0$; this is rather surprising since all of the mathematics on which the spatial differentiation in DSS002 and the ODE integration in RKF45 is based assumes well-behaved functions (functions that are smooth, or free of discontinuities). This rather fortuitous result, however, will in general not occur as we shall see in Chapter 3, and we should be aware of the potential numerical problems caused by discontinuities.

Also, we should appreciate the importance of discontinuities in applications, i.e., equations (2.1) to (2.4) illustrate a common situation in the

analysis of physical systems. For example, a steel ingot may leave a furnace at 1000°C and be immersed in an oil quench at room temperature (25°C). The surface of the ingot therefore has an initial condition of 1000 at $t = 0$ and a boundary condition of 25 for $t > 0$. We might then want to calculate the temperature within the ingot as a function of time and position to determine whether the temperature changes rapidly enough to give the desired physical properties throughout the steel (such as hardness). This physical situation is exactly what the preceding problem, equations (2.1) to (2.4), describes. Another common situation is the motion of mechanical systems, for example, when an automobile hits a wall in a safety test, or the booster is discharged from the space shuttle; the equations of motion describing these systems undergo an instantaneous change that must be accommodated if we are going to simulate these systems realistically with a mathematical model.

Finally, we should note the ease with which initial condition (1.22) was changed to initial condition (2.2) in subroutine INITAL, and boundary condition (1.24) was changed to boundary condition (2.4) in subroutine DERV. This illustrates the power of the numerical (NUMOL) approach, since the exact solution changes substantially. This point is illustrated even more dramatically in the next example, where we go from a problem with a known exact solution to a problem with an unknown exact solution by a trivial change in subroutine DERV.

2.3 A Nonlinear Problem

We now consider a nonlinear problem, which to the best of our knowledge does not have an exact solution. We should, of course, first define what we mean by a nonlinear problem. This will be done in Chapter 6 when we classify PDEs. However, for our immediate purpose, we shall say simply that a nonlinear PDE problem does not have the *dependent variable and its various derivatives entirely to the first power*. Thus, we have the important distinction between the *order* of a PDE problem [as defined by the highest-order derivatives in the PDE(s)] and the *degree* of a PDE problem [as determined by the power of the terms containing the dependent variable(s)]. It follows that *first-degree equations are linear*.

To illustrate the distinction between a linear and a nonlinear problem, consider a variant of equations (2.1) to (2.4):

$$T_t = T_{xx} \quad (2.6)$$

$$T(x, 0) = 0 \quad (2.7)$$

$$T(0, t) = 1, \quad T_x(1, t) = (1 - T(1, t)^m) \quad (2.8) \quad (2.9)$$

where m is a given constant. Two things should be noted about boundary condition (2.9): (1) it is a boundary condition of the third type, i.e., it is a combination of Dirichlet and Neumann boundary conditions since it includes $T(1, t)$ and its derivative $T_x(1, t)$, and (2) it is nonlinear for $m \neq 1$ since $T(1, t)$ is not to the first power. Thus, for $m = 1$, an exact solution to equations (2.6) to (2.9) can easily be derived, while for $m \neq 1$, an exact solution is probably not available (what a difference m makes!); as we shall see, the change in the programming for these two cases is trivial.

Subroutines INITIAL, DERV, and PRINT (plus data) for the linear ($m = 1$) and nonlinear ($m \neq 1$) cases are listed in Program 2.3. Note that

Program 2.3 Subroutines INITIAL, DERV, PRINT, and Data for Equations (2.6) to (2.9)

```

SUBROUTINE INITIAL
C...
C... SUBROUTINE INITIAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C... INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (2.6) TO
C... (2.9)
C...
      COMMON/T/    TIME,      NSTOP,      NORUN
      1          /Y/    T(51)
      2          /F/    TT(51)
      3          /R/    X(51),    TX(51),    TX(51),
      4                  L,        DX
      5          /I/    N
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C... LENGTH
L=1.0
C...
C... GRID SPACING
N=51
DX=L/FLOAT(N-1)
C...
C... INITIAL CONDITION (2.7)
DO 1 I=1,N
    X(I)=FLOAT(I-1)*DX
    T(I)=0.
1 CONTINUE
RETURN
END

SUBROUTINE DERV
C...
C... SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (2.6) TO (2.9)
C...
      COMMON/T/    TIME,      NSTOP,      NORUN
      1          /Y/    T(51)
      2          /F/    TT(51)
      3          /R/    X(51),    TX(51),    TX(51),
      4                  L,        DX
      5          /I/    N
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L

```

continues

```

C...
C... BOUNDARY CONDITION (2.8)
      T(1)=1.
      TT(1)=0.
C...
C... DERIVATIVE TX
      CALL DSS002(0.,L,N,T,TX)
C...
C... BOUNDARY CONDITION (2.9)
C...
C... LINEAR CASE
      IF(NORUN.EQ.1)M=1
C...
C... NONLINEAR CASE
      IF(NORUN.EQ.2)M=4
C...
      TX(N)=(1.0-T(N)**M)
C...
C... DERIVATIVE TXX
      CALL DSS002(0.,L,N,TX,TXX)
C...
C... EQUATION (2.6)
      DO 2 I=2,N
      TT(I)=TXX(I)
2     CONTINUE
      RETURN
      END

      SUBROUTINE PRINT(NI,NO)
C...
C... SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PRO1P3 TO PRINT THE
C... NUMOL SOLUTION OF EQUATIONS (2.6) TO (2.9)
C...
      COMMON/T/    TIME,      NSTOP,      NORUN
      1        /Y/    T(51)
      2        /F/    TT(51)
      3        /R/    X(51),    TX(51),    TXX(51),
      4              L,          DX
      5        /I/    N
C...
C... TYPE SELECTED VARIABLES AS REAL
      REAL L
C...
C... PRINT THE NUMOL SOLUTION TO EQUATIONS (2.6) TO (2.9)
      WRITE(NO,2)TIME,(T(I),I=1,N,10)
2     FORMAT(' TIME = ',F6.2,/,15X,' X=0',5X,'X=0.2',5X,'X=0.4',5X,
      1                           'X=0.6',5X,'X=0.8',5X,' X=1',/,
      2                           ' T(X,T)',6F10.6,/)
      RETURN
      END

NUMOL SOLUTION OF EQUATIONS (2.6) TO (2.9), M = 1
O.      1.0          0.2
      51          0.000001
NUMOL SOLUTION OF EQUATIONS (2.6) TO (2.9), M = 4
O.      1.0          0.2
      51          0.000001
END OF RUNS

```

the two cases, $m = 1$ (linear) and $m = 4$ (nonlinear) are programmed in DERV through the use of the run counter, NORUN, which is the third element in COMMON/T/. NORUN is set by the main program, Program 1.3d, to the value of 1 when the first set of data is read, to 2 when the

second set is read, etc. Thus, NORUN can be used to change the coding from run to run. Also, the ease of programming the nonlinear boundary condition, equation (2.9) with $m = 4$, should be noted.

The numerical output from subroutine PRINT for the two runs is listed in Table 2.3. Three features of the solutions in Table 2.3 can be noted: (1) the solutions for the two cases ($m = 1$ and 4) are significantly different (compare the two solutions at $t = 0.2$); (2) the solutions appear to be approaching the infinite-time solution $T(x, \infty) = 1$; and (3) the solutions in both cases go through a minimum in the neighborhood of $x = 1$ (note the solutions at $x = 0.6, 0.8$, and 1.0 for $t = 0.2$).

Table 2.3 Numerical Solution from Program 2.3

```
RUN NO. - 1 NUMOL SOLUTION OF EQUATIONS (2.6) TO (2.9), M = 1
INITIAL T - 0.000E+00
FINAL T - .100E+01
PRINT T - .200E+00
NUMBER OF DIFFERENTIAL EQUATIONS - 51
MAXIMUM INTEGRATION ERROR - .100E-05
TIME = 0.00
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000
TIME = .20
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 1.000000 .791814 .618976 .509723 .480446 .534235
TIME = .40
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 1.000000 .909475 .833664 .784880 .771036 .794366
TIME = .60
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 1.000000 .960249 .926954 .905521 .899433 .909677
TIME = .80
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 1.000000 .982541 .967918 .958505 .955831 .960330
TIME = 1.00
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T) 1.000000 .992332 .985910 .981776 .980601 .982577

RUN NO. - 2 NUMOL SOLUTION OF EQUATIONS (2.6) TO (2.9), M = 4
INITIAL T - 0.000E+00
FINAL T - .100E+01
PRINT T - .200E+00
```

continues

NUMBER OF DIFFERENTIAL EQUATIONS - 51

MAXIMUM INTEGRATION ERROR - .100E-05

TIME =	0.00	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
TIME =	.20	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	1.000000	.802771	.645579	.561538	.571614	.681308	
TIME =	.40	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	1.000000	.936506	.888307	.867308	.879232	.922002	
TIME =	.60	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	1.000000	.982163	.968859	.963489	.967464	.979825	
TIME =	.80	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	1.000000	.995175	.991593	.990175	.991292	.994657	
TIME =	1.00	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	1.000000	.998707	.997749	.997371	.997673	.998577	

Now, we come to an interesting question: Can we reasonably expect the solution to go through a minimum? In other words, is this correct, or is there an error in the solution? We can explain this feature of the solution in the following way using physical arguments. Remember that equation (2.6) describes the conduction of heat in a solid (as explained during the derivation in Section 1.2). Since the system starts out at $T(x, 0) = 0$, and the left face is then subject to the boundary condition $T(0, t) = 1$, heat flows into the system through the left face ($x = 0$) since heat flows from regions of higher temperature to regions of lower temperature. However, heat also flows into the system through the right face at a rate proportional to $T_x(1, t)$ [this follows from Fourier's first law, equation (1.3); note that the heat flow is in the negative x direction since $T_x(1, t) > 0$ from equation (2.9)]. Thus, *heat is flowing into the system through both faces, at $x = 0$ and $x = 1$.* Consequently, it seems reasonable that the temperature, $T(x, t)$, would go through a minimum at some interior value of x between $x = 0$ and $x = 1$. (Do you agree?)

This discussion raises a more general question: *How do we know when the solution is correct or when do we believe what comes out of the computer, and when do we not?* This is perhaps the most difficult question of all to answer in this kind of work, since *we really have no independent check* (if we had an exact solution, there would be no need to calculate a

numerical solution; in most realistic physical problems, we will not have an exact solution to serve as a check). Thus, about the best we can do is to check the numerical (NUMOL) solution as we did in the last example, i.e.: (1) whether the solution has properties that we can explain physically, such as the minimum and (2) whether the solution satisfies some special case that we know is correct, such as $T(x, \infty) = 1$.

Thus, there is no comprehensive and exhaustive way to check the numerical solutions to PDEs. Each problem is a new problem in this regard, and we must use all of our experience, intuition, and knowledge of the problem to check the numerical solution in whatever ways occur to us, and this process is highly dependent on the particular problem. We wish we had something more rigorous to offer, but we don't. In other words, computing numerical solutions to PDEs will never ultimately be reduced to “button-pushing” on a computer keyboard, at least for new problems that have not been studied previously (which, after all, are often the most important and intriguing problems). The evaluation of numerical PDE solutions is a highly challenging exercise, with success strongly dependent on the experience of the analyst. To state this another way, we tend to be wary of “automatic” procedures that will compute solutions to PDEs without requiring any knowledge of the algorithms and the associated coding; errors can easily be camouflaged by “sophisticated” graphical output, which is in fact displaying incorrect solutions.

On the other hand, we do not wish to convey the impression that the evaluation of numerical PDE solutions is essentially an impossible task. We only suggest that every area of existing knowledge about the solution should be used in the evaluation. This is really a challenge that should be accepted in anticipation of finding interesting and possibly unexpected results.

Incidentally, boundary condition (2.9) is not a just mathematical curiosity. In fact, with $m = 1$, it states that the rate of heat transfer per unit area (the heat *flux*) is proportional to the temperature difference $1 - T(1, t)$. This relationship between heat flux and temperature difference is generally termed *Newton heating (or cooling)* and is broadly applicable to heat transfer in convective systems. On the other hand, with $m = 4$, the rate of heat transfer is proportional to the difference in the fourth powers of the temperatures, $1^4 - T(1, t)^4$, which describes radiant heat transfer according to the *Stefan–Boltzmann law*.

Next, we consider the solution of the heat-conduction equation (Fourier's second law) in cylindrical coordinates. We might wish to compute, for example, the temperature distribution in a long cylindrical rod rather than a slab.

2.4 A Linear PDE in Cylindrical Coordinates: _____ The Parabolic Fourier Second Law _____

If we again use an energy balance, as in Section 1.2, on an incremental cylindrical element of radius r and thickness Δr , then take the limit as $\Delta r \rightarrow 0$, we arrive at the following PDE in cylindrical coordinates, which is analogous to equation (1.4)

$$\rho C_p \frac{\partial T}{\partial t} = k \left(\frac{\partial^2 T}{\partial r^2} + \frac{1}{r} \frac{\partial T}{\partial r} \right) \quad (2.10)$$

In terms of the subscript notation.

$$T_t = \alpha(T_{rr} + (1/r)T_r) \quad (2.11)$$

which is analogous to equation (1.10) with $\alpha = k/(\rho C_p)$ [equations (2.10) and (2.11) are *Fourier's second law in cylindrical coordinates*, an example of a *parabolic PDE* with a *variable coefficient* ($1/r$); the *geometric classification of PDEs*, i.e., *parabolic, elliptic, hyperbolic*, will be discussed in Chapter 6].

The initial and boundary conditions for equation (2.11) will be taken as

$$T(r, 0) = 0 \quad (2.12)$$

$$T_r(0, t) = 0, \quad T(r_0, t) = T_0 \quad (2.13) \quad (2.14)$$

We have to attend to one mathematical detail before proceeding to a NUMOL solution. At $r = 0$, the term $(1/r)T_r$ in equation (2.11) is *indeterminate* ($0/0$) since from boundary condition (2.13) $T_r = 0$. To evaluate this indeterminate form, we apply l'Hospital's rule (differentiating the numerator T_r and the denominator r with respect to r):

$$\lim_{r \rightarrow 0} \frac{1}{r} T_r = \lim_{r \rightarrow 0} \frac{1}{1} T_{rr} = T_{rr} \quad (2.15)$$

Thus, when we step along the grid in r , we will use equation (2.11) except at $r = 0$, when we will switch to (2.15) for the term $(1/r)T_r$.

Otherwise, the coding in subroutine DERV is a straightforward implementation of equations (2.11), (2.13), and (2.14). Subroutines INITAL, DERV, and PRINT, plus data, are listed in Program 2.4 (for $\alpha = 1$, $r_0 = 1$, $T_0 = 25$). Note in particular the coding in subroutine DERV for the variable coefficient, $1/r$, in equation (2.11). Otherwise, the coding is essentially the same as for *Cartesian coordinates* (Programs 1.3a to 1.3c), which demonstrates the versatility of the NUMOL in accommodating various

orthogonal coordinate systems (coordinate systems for which the boundary conditions can be stated relatively easily, or in other words, coordinate systems that avoid irregular boundaries or geometries in defining a PDE problem).

Program 2.4 Subroutines INITIAL, DERV, PRINT, and Data for Equations (2.11) to (2.14)

```

SUBROUTINE INITIAL
C...
C... SUBROUTINE INITIAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C... INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (2.11) TO
C... (2.14)
C...
      COMMON/T/    TIME,      NSTOP,      NORUN
      1          /Y/    T(51)
      2          /F/    TT(51)
      3          /R/    R(51),    TR(51),    TRR(51),
      4                  R0,        T0,        DR
      5          /I/    N
C...
C... SURFACE TEMPERATURE
T0=25.
C...
C... RADIUS
R0=1.0
C...
C... GRID SPACING
N=51
DR=R0/FLOAT(N-1)
C...
C... INITIAL CONDITION (2.12)
DO 1 I=1,N
      R(I)=FLOAT(I-1)*DR
      T(I)=0.
1     CONTINUE
      RETURN
      END

SUBROUTINE DERV
C...
C... SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (2.11) TO (2.14)
C...
      COMMON/T/    TIME,      NSTOP,      NORUN
      1          /Y/    T(51)
      2          /F/    TT(51)
      3          /R/    R(51),    TR(51),    TRR(51),
      4                  R0,        T0,        DR
      5          /I/    N
C...
C... BOUNDARY CONDITION (2.14)
      T(N)=T0
      TT(N)=0.
C...
C... DERIVATIVE TR
      CALL DSS002(0.,R0,N,T,TR)
C...
C... BOUNDARY CONDITION (2.13)
      TR(1)=0.

```

continues

```

C...
C...  DERIVATIVE TRR
      CALL DSS002(0.,RO,N,TR,TRR)
C...
C...  EQUATION (2.11)
      DO 2 I=1,N-1
C...
C...  R = 0 (EQUATIONS (2.11) AND (2.15))
      IF(I.EQ.1)THEN
          TT(I)=2.0*TRR(I)
C...
C...  R NE 0 (EQUATION (2.11))
      ELSE IF(I.NE.1)THEN
          TT(I)=TRR(I)+1.0/R(I)*TR(I)
      END IF
2    CONTINUE
      RETURN
      END

      SUBROUTINE PRINT(NI,NO)
C...
C...  SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PR01P3 TO PRINT THE
C...  NUMOL SOLUTION OF EQUATIONS (2.11) TO (2.14)
C...
      COMMON/T/      TIME,      NSTOP,      NORUN
      1        /Y/      T(51)
      2        /F/      TT(51)
      3        /R/      R(51),     TR(51),     TRR(51),
      4                  RO,       TO,        DR
      5        /I/      N
C...
C...  PRINT THE NUMOL SOLUTION TO EQUATIONS (2.11) TO (2.14)
      WRITE(NO,2)TIME,(T(I),I=1,N,10)
2    FORMAT(' TIME = ',F6.2,/,15X,' R=0',5X,'R=0.2',5X,'R=0.4',5X,
      1                           'R=0.6',5X,'R=0.8',5X,' R=1',/,
      2                           ' T(R,T)',6F10.3,/)
      RETURN
      END

NUMOL SOLUTION OF EQUATIONS (2.11) TO (2.14)
0.           1.0           0.2
      51           0.000001
END OF RUNS

```

The numerical output from Program 2.4 is listed in Table 2.4. Note that the solution approaches the infinite-time solution $T(r, \infty) = T_0 = 25$, as expected. The exact solution for finite time, again as in the case of Cartesian coordinates, involves an infinite series, with the radial dependency (function of r) expressed as Bessel functions: therefore, the exact solution is relatively difficult to evaluate [and may also exhibit oscillation or a Gibbs phenomenon because of the discontinuity between the initial and boundary conditions, equations (2.12) and (2.14), at $r = r_0$]. This problem therefore demonstrates that even though an exact solution is available (generally for a single linear PDE), the numerical (NUMOL) solution is often easier to obtain.

In summary, with this example we have defined the essential details for programming a NUMOL solution in various orthogonal coordinate

Table 2.4 Numerical Solution from Program 2.4

RUN NO.	-	1	NUMOL SOLUTION OF EQUATIONS (2.11) TO (2.14)				
INITIAL T	-	0.000E+00					
FINAL T	-	.100E+01					
PRINT T	-	.200E+00					
NUMBER OF DIFFERENTIAL EQUATIONS	-	51					
MAXIMUM INTEGRATION ERROR	-	.100E-05					
TIME	=	0.00					
		R=0	R=0 . 2	R=0 . 4	R=0 . 6	R=0 . 8	R=1
T(R,T)		0.000	0.000	0.000	0.000	0.000	25.000
TIME	=	.20					
		R=0	R=0 . 2	R=0 . 4	R=0 . 6	R=0 . 8	R=1
T(R,T)		12.461	13.162	15.157	18.132	21.603	25.000
TIME	=	.40					
		R=0	R=0 . 2	R=0 . 4	R=0 . 6	R=0 . 8	R=1
T(R,T)		21.035	21.261	21.901	22.845	23.938	25.000
TIME	=	.60					
		R=0	R=0 . 2	R=0 . 4	R=0 . 6	R=0 . 8	R=1
T(R,T)		23.752	23.824	24.025	24.322	24.666	25.000
TIME	=	.80					
		R=0	R=0 . 2	R=0 . 4	R=0 . 6	R=0 . 8	R=1
T(R,T)		24.607	24.630	24.693	24.787	24.895	25.000
TIME	=	1.00					
		R=0	R=0 . 2	R=0 . 4	R=0 . 6	R=0 . 8	R=1
T(R,T)		24.877	24.884	24.903	24.933	24.967	25.000

systems. In particular, some special coding may be required at specific points in the spatial grid [equation (2.15) at $r = 0$]. Next, we consider a nonlinear problem in spherical coordinates.

2.5 A Nonlinear PDE in Spherical Coordinates

Consider a problem of heat conduction in a sphere in which an exothermic chemical reaction takes place. An energy balance on a shell of radius r and thickness Δr gives

$$(4\pi r^2 \Delta r) \rho C_p \partial T / \partial t = (4\pi r^2 q|_r - 4\pi(r + \Delta r)^2 q|_{r+\Delta r}) + (4\pi r^2 \Delta r) \Delta h r_a \quad (2.16)$$

where most of the constants and variables are the same as in the derivation

of equation (1.2). The required additional definitions are

- Δh heat of reaction (joules per mole of reactant)
- r_a volumetric rate of reaction (moles of reactant per cubic centimeter per second)

If again we use Fourier's first law to eliminate the conductive heat flux q from equation (2.16):

$$q = -k \frac{\partial T}{\partial r}$$

Equation (2.16) can be easily rearranged to

$$(4\pi r^2 \Delta r) \rho C_p \frac{\partial T}{\partial t} = k(4\pi(r + \Delta r)^2 \frac{\partial T}{\partial r}|_{r+\Delta r} - 4\pi r^2 \frac{\partial T}{\partial r}|_r) + (4\pi r^2 \Delta r) \Delta h r_a$$

Division by $4\pi r^2 \Delta r \rho C_p$, followed by $\Delta r \rightarrow 0$, gives

$$\frac{\partial T}{\partial t} = k/(\rho C_p) \frac{1}{r^2 \partial r} \left(r^2 \frac{\partial T}{\partial r} \right) + (\Delta h / (\rho C_p)) r_a$$

Finally, the rate of reaction is given by the Arrhenius temperature dependence

$$r_a = k_0 e^{-E/(RT)}$$

which, with expansion of the radial derivative group (using the differentiation of a product rule), gives

$$\frac{\partial T}{\partial t} = k/(\rho C_p) (\frac{\partial^2 T}{\partial r^2} + (2/r) \frac{\partial T}{\partial r}) + (\Delta h k_0 / (\rho C_p)) e^{-E/(RT)}$$

where

E = activation energy (J/mol)

R = gas constant (J/mol · K)

In subscript notation, we have

$$T_t = \alpha(T_r + (2/r)T_r) + \beta e^{-E/(RT)} \quad (2.17)$$

where $\alpha = k/(\rho C_p)$, $\beta = \Delta h k_0 / (\rho C_p)$.

We shall now compute a NUMOL solution to equation (2.17) subject to the auxiliary conditions

$$T(r, 0) = T_0 \quad (2.18)$$

$$T_r(0, t) = 0, \quad T(r_0, t) = T_a \quad (2.19) \quad (2.20)$$

Note that equation (2.17) is highly nonlinear due to the exponential in T .

We shall observe that this nonlinearity has a strong, unexpected effect on the solution.

Subroutines INITIAL, DERV, and PRINT, plus data, are listed in Program 2.5. Note the ease of programming the nonlinear exponential in subroutine DERV, and the use of the special form of equation (2.17) at $r = 0$. The numerical output from Program 2.5 is listed in Table 2.5.

Program 2.5 Subroutines INITIAL, DERV, PRINT, and Data for Equations (2.17) to (2.20)

```

SUBROUTINE INITIAL
C...
C...  SUBROUTINE INITIAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C...  INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (2.17) TO
C...  (2.20)
C...
      COMMON/T/,      TIME,      NSTOP,      NORUN
      1        /Y/   T(51)
      2        /F/   TT(51)
      3        /R/   R(51),    TR(51),    TRR(51),
      4                  R0,       TO,       TA,       A,       B,       RG,
      5                  E,        DR
      6        /I/   N
      REAL           K,        KO
C...
C...  SET THE PROBLEM CONSTANTS (ENERGY IS IN CALORIES RATHER THAN
C...  JOULES)
      T0=25.
      R0=1.0
      E=30800.0
      RG=1.987
      RHO=1.71
      CP=0.300
      K=2.4E-04
      DH=1276.
      KO=1.26E+11
C...
C...  SET THE SURFACE TEMPERATURE FOR BOUNDARY CONDITION (2.20)
      IF(NORUN.EQ.1)TA=125.
      IF(NORUN.EQ.2)TA=158.
C...
C...  COMPUTE THE THERMAL DIFFUSIVITY, COEFFICIENT OF REACTION RATE TERM
      A=K/(RHO*CP)
      B=(DH*KO/(RHO*CP))
C...
C...  GRID SPACING
      N=51
      DR=R0/FLOAT(N-1)
C...
C...  INITIAL CONDITION (2.18)
      DO 1 I=1,N
          R(I)=FLOAT(I-1)*DR
          T(I)=T0
1     CONTINUE
      RETURN
      END

      SUBROUTINE DERV
C...
C...  SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE

```

continues

66 2. Some Applications of the Numerical Method of Lines

```

C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (2.17) TO (2.20)
C...
      COMMON/T/    TIME,      NSTOP,      NORUN
      1          /Y/    T(51)
      2          /F/    TT(51)
      3          /R/    R(51),    TR(51),    TRR(51),
      4                  RO,        TO,        TA,        A,        B,        RG,
      5                  E,        DR
      6          /I/    N
      REAL        K,        KO

C...
C... BOUNDARY CONDITION (2.20)
      T(N)=TA
      TT(N)=0.

C...
C... DERIVATIVE TR
      CALL DSS002(0.,RO,N,T,TR)
C...
C... BOUNDARY CONDITION (2.19)
      TR(1)=0.

C...
C... DERIVATIVE TRR
      CALL DSS002(0.,RO,N,TR,TRR)
C...
C... EQUATION (2.17)
      DO 1 I=1,N-1
C...
C...     R NE 0
      IF(I.NE.1)THEN
      1          TT(I)=A*(TRR(I)+(2./R(I))*TR(I))
                  +B*EXP(-E/(RG*(T(I)+273.16)))
C...
C...     R = 0
      ELSE IF(I.EQ.1)THEN
      1          TT(I)=3.*A*TRR(I)
                  +B*EXP(-E/(RG*(T(I)+273.16)))
      END IF
      1 CONTINUE
      RETURN
      END

      SUBROUTINE PRINT(NI,NO)
C...
C... SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PRO1P3 TO PRINT THE
C... NUMOL SOLUTION OF EQUATIONS (2.17) TO (2.20)
C...
      COMMON/T/    TIME,      NSTOP,      NORUN
      1          /Y/    T(51)
      2          /F/    TT(51)
      3          /R/    R(51),    TR(51),    TRR(51),
      4                  RO,        TO,        TA,        A,        B,        RG,
      5                  E,        DR
      6          /I/    N
      REAL        K,        KO

C...
C... PRINT THE NUMOL SOLUTION TO EQUATIONS (2.17) TO (2.20)
      WRITE(NO,2)TIME,(T(I),I=1,N,10)
      2 FORMAT(' TIME = ',F6.1,/,15X,' R=0',5X,' R=0.2',5X,' R=0.4',5X,
      1                               ' R=0.6',5X,' R=0.8',5X,' R=1',/,
      2                               ' T(R,T)',6F10.3,/)

      RETURN
      END

```

continues

```

NUMOL SOLUTION OF EQUATIONS (2.17) TO (2.20), TS = 125
O.      1000.      200.
      51      0.000001
NUMOL SOLUTION OF EQUATIONS (2.17) TO (2.20), TS = 158
O.      1000.      200.
      51      0.000001
END OF RUNS

```

Table 2.5 Numerical Solution from Program 2.5

```

RUN NO. - 1 NUMOL SOLUTION OF EQUATIONS (2.17) TO (2.20), TS = 125
INITIAL T - 0.000E+00
FINAL T - .100E+04
PRINT T - .200E+03
NUMBER OF DIFFERENTIAL EQUATIONS - 51
MAXIMUM INTEGRATION ERROR - .100E-05
TIME = 0.0
      R=0      R=0.2      R=0.4      R=0.6      R=0.8      R=1
T(R,T) 25.000  25.000  25.000  25.000  25.000  125.000
TIME = 200.0
      R=0      R=0.2      R=0.4      R=0.6      R=0.8      R=1
T(R,T) 50.519  54.450  66.060  84.156  105.497  125.000
TIME = 400.0
      R=0      R=0.2      R=0.4      R=0.6      R=0.8      R=1
T(R,T) 93.616  95.628  101.220  109.160  117.701  125.000
TIME = 600.0
      R=0      R=0.2      R=0.4      R=0.6      R=0.8      R=1
T(R,T) 112.745  113.560  115.810  118.964  122.281  125.000
TIME = 800.0
      R=0      R=0.2      R=0.4      R=0.6      R=0.8      R=1
T(R,T) 120.671  120.983  121.835  122.998  124.159  125.000
TIME = 1000.0
      R=0      R=0.2      R=0.4      R=0.6      R=0.8      R=1
T(R,T) 124.018  124.116  124.374  124.695  124.947  125.000
RUN NO. - 2 NUMOL SOLUTION OF EQUATIONS (2.17) TO (2.20), TS = 158
INITIAL T - 0.000E+00
FINAL T - .100E+04
PRINT T - .200E+03
NUMBER OF DIFFERENTIAL EQUATIONS - 51
MAXIMUM INTEGRATION ERROR - .100E-05
TIME = 0.0
      R=0      R=0.2      R=0.4      R=0.6      R=0.8      R=1
T(R,T) 25.000  25.000  25.000  25.000  25.000  158.000

```

continues

TIME =	200.0	R=0	R=0.2	R=0.4	R=0.6	R=0.8	R=1
T(R,T)	58.977	64.219	79.711	103.905	132.511	158.000	

TIME =	400.0	R=0	R=0.2	R=0.4	R=0.6	R=0.8	R=1
T(R,T)	117.064	119.853	127.638	138.717	150.231	158.000	

TIME =	600.0	R=0	R=0.2	R=0.4	R=0.6	R=0.8	R=1
T(R,T)	147.481	148.824	152.398	156.754	159.443	158.000	

TIME =	800.0	R=0	R=0.2	R=0.4	R=0.6	R=0.8	R=1
T(R,T)	212.505	210.201	200.826	186.611	172.061	158.000	

TIME =	803.9	R=0	R=0.2	R=0.4	R=0.6	R=0.8	R=1
T(R,T)	3078.931	243.068	210.408	189.264	172.730	158.000	

IFLAG = 6

INDICATING AN INTEGRATION ERROR, SO THE CURRENT RUN IS TERMINATED. PLEASE REFER TO THE DOCUMENTATION FOR SUBROUTINE

RKF45

FOR AN EXPLANATION OF THESE ERROR INDICATORS

The solution for the two cases $T_a = 125^\circ\text{C}$ and 158°C are clearly different, which is due to the nonlinearity of this problem. For $T_a = 125$, the solution is smooth in the sense that it approaches the surface temperature of 125 [$T(r, \infty) = 125$]; thus, the heat is removed from the sphere by conduction as rapidly as it is generated by the reaction. However, at least two alternative situations could occur:

(1) The reaction could generate heat at a rate sufficient to cause the temperature inside the sphere to exceed 125, but the solution would eventually approach a steady state at the higher temperature in which the rate of heat generation is balanced by the rate of radial heat conduction [so $T_t \rightarrow 0$; thus the radial heat-conduction term and the reaction heat-generation term in equation (2.17) would be equal, but opposite in sign].

(2) The reaction could generate heat at a rate sufficient to cause the temperature inside the sphere to exceed 125. Also, the heat of reaction may not be removed by radial conduction at the same rate it is generated by the reaction, so the temperature goes even higher. Furthermore, since the rate of heat generation, as given by the exponential, increases with increasing temperature, the reaction generates heat at a still higher rate, thereby increasing the temperature further, etc., etc., until the solid eventually explodes! This is the situation for the second case, $T_a = 158$, when heat cannot be removed by radial conduction rapidly enough to balance the rate of heat generation by the reaction. Eventually, at

$t = 803.9$, the sphere begins to explode at the center [$T(0, 803.9) = 3078.931$]. At this instance in time, RKF45 is no longer able to compute a numerical solution with the specified accuracy (0.000001 in the third line of data), and terminates execution with an error indicator $\text{IFLAG} = 6$. We did make a run in which the integrator was forced to continue the calculation even though the error criterion could not be met, and the explosion (elevated temperature) quickly spread throughout the sphere.

Two additional points could be made concerning the second case: (1) the problem constants set in INITAL are for an industrial explosive and (2) to complete the analysis of what took place in the second solution, the reader should verify that the exponential term, $e^{-E/(RT)}$, does in fact increase with T (the rates of all chemical reactions increase with increasing temperature).

Finally, we can consider briefly some essential differences between linear and nonlinear models, as illustrated by equations (2.17) to (2.20).

(1) For linear systems, the absolute values of the dependent variables are inconsequential; for example, specifying a temperature of zero at a boundary does not imply a temperature of absolute zero, but rather, zero relative to some datum (and the datum is not part of the statement of a linear system model). On the other hand, the absolute values of the dependent variables are an essential part of a nonlinear system specification; for example, the absolute value of the temperature is required in the exponential for the rate of the reaction in equation (2.17), and thus, if the temperature $T(r, t)$ is in degrees centigrade (Celsius) this must be converted to degrees kelvin when used in the exponential, which in subroutine DERV of Program 2.5 required the addition of 273.16.

(2) For linear systems, the solution is generally proportional to the inputs such as a temperature at the boundary (if the boundary temperature is changed, the solution will change proportionately). For a nonlinear system, no such simple mathematical relationship exists for changes in the solution with model parameters; in fact, the solution may completely change its character. This was illustrated in the preceding example when a change in the surface temperature from 125 to 158 produced one solution that was stable (125) and a second solution that was unstable (158).

(3) We can infer from (2) another important distinction between linear and nonlinear systems. For the former, the solution is unique (there is only one solution), while for the latter, multiple solutions are possible, and in fact, often exist. The problem of multiple solutions of nonlinear systems, some of which can be unstable, has been studied extensively, but we shall not discuss this matter further.

2.6 A PDE Second-Order in Time: _____ The Hyperbolic Wave Equation _____

In all of the examples considered so far, the initial-value variable, t , has appeared in only first-order derivatives in the PDEs. There are, however, important PDEs in which the initial-value independent variable appears in second-order derivatives. A classic PDE of this type is the *wave equation*, which in one-dimensional Cartesian coordinates is

$$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2} \quad (2.21)$$

or in subscript notation

$$u_{tt} = a^2 u_{xx} \quad (2.22)$$

where a is a characteristic velocity. Note that the dependent variable is $u(=u(x, t))$, which is a rather standard choice in the PDE literature, particularly if the dependent variable is not given a physical meaning that might suggest another name [e.g., $T(x, t)$ to represent temperature]. The use of u , in addition to being a generally used notation for the dependent variable, also has the minor advantage of allowing us to use T for time in our Fortran coding (previously we had to use TIME because T was the temperature).

We shall develop a NUMOL solution to equation (2.22) for the auxiliary conditions

$$u(x, 0) = \sin(\pi x/L), \quad u_t(x, 0) = 0 \quad (2.23) \quad (2.24)$$

and

$$u(0, t) = u(L, t) = 0 \quad (2.25) \quad (2.26)$$

Because equation (2.22) is second-order in t and x , it requires *two auxiliary conditions for each of these independent variables*. Note that t is an initial-value independent variable [equations (2.23) and (2.24) are both specified at $t = 0$] and x is a boundary-value variable [equations (2.25) and (2.26) are specified at different values of x , $x = 0$ and $x = L$].

The exact solution to equations (2.22) to (2.26) is

$$u(x, t) = \cos(a\pi t/L) \sin(\pi x/L) \quad (2.27)$$

which, again, will be used to evaluate the NUMOL solution.

Before we proceed with the programming, we have to reformulate the problem because RKF45 (and most other ODE integrators) will accommodate only ODEs first-order in t , while equation (2.22) is second-order in t .

This incompatibility is easily resolved. If we define two variables, $u_1(x, t)$ and $u_2(x, t)$, as

$$u_1(x, t) = u(x, t), \quad u_2(x, t) = u_t(x, t) \quad (2.28) \quad (2.29)$$

then equations (2.22) to (2.27) can be written in terms of u_1 and u_2 as

$$u_{2t} = a^2 u_{1xx}, \quad u_{1t} = u_2 \quad (2.30) \quad (2.31)$$

$$u_1(x, 0) = \sin(\pi x/L), \quad u_2(x, 0) = 0 \quad (2.32) \quad (2.33)$$

$$u_1(0, t) = u_1(L, t) = 0 \quad (2.34) \quad (2.35)$$

$$u_1(x, t) = \cos(a\pi t/L) \sin(\pi x/L) \quad (2.36)$$

Note that equations (2.30) and (2.31) are now first-order in t ; in other words, we have replaced one second-order equation in t , (2.22), with two first-order equations in t , (2.30) and (2.31). In general, we can write an *nth-order equation as n first-order equations by defining n variables* in analogy with equations (2.28) and (2.29).

Subroutines INITAL, DERV, and PRINT for equations (2.30) to (2.35), plus data, are listed in Program 2.6 (for $a = L = 1$). The second line

Program 2.6 Subroutines INITAL, DERV, PRINT, and Data for Equations (2.30) to (2.35)

```

SUBROUTINE INITAL
C...
C...  SUBROUTINE INITAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C...  INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (2.30) TO
C...  (2.35)
C...
COMMON/T/
      /Y/      U1(51),      U2(51)
      /F/      U1T(51),     U2T(51)
      /R/      X(51),       U1X(51),    U1XX(51),    UE(51),    DIFF(51),
      /L/           L,          PI,          DX
      /I/           N
C...
C...  TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C...  CALCULATE PI FOR USE IN DO LOOP 1
PI=ACOS(-1.0)
C...
C...  LENGTH
L=1.0
C...
C...  GRID SPACING
N=51
DX=L/FLOAT(N-1)
C...
C...  INITIAL CONDITIONS (2.32), (2.33)
DO 1 I=1,N
      X(I)=FLOAT(I-1)*DX
      U1(I)=SIN(PI*X(I)/L)
      U2(I)=0.

```

continues

```

1      CONTINUE
RETURN
END

SUBROUTINE DERV
C...
C... SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (2.30) TO (2.35)
C...
COMMON/T/           T
1      /Y/    U1(51),   U2(51)
2      /F/    U1T(51),  U2T(51)
3      /R/    X(51),    U1X(51),   U1XX(51),    UE(51),   DIFF(51),
4                  L,        PI,        DX
5      /I/    N

C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L

C...
C... BOUNDARY CONDITION (2.34)
U1(1)=0.
U2(1)=0.
U1T(1)=0.
U2T(1)=0.

C...
C... BOUNDARY CONDITION (2.35)
U1(N)=0.
U2(N)=0.
U1T(N)=0.
U2T(N)=0.

C...
C... DERIVATIVE UX
CALL DSS002(0.,L,N,U1,U1X)

C...
C... DERIVATIVE UXX
CALL DSS002(0.,L,N,U1X,U1XX)

C...
C... PDES
DO 2 I=2,N-1

C...
C... EQUATION (2.31)
U1T(I)=U2(I)

C...
C... EQUATION (2.30)
U2T(I)=U1XX(I)

2      CONTINUE
RETURN
END

SUBROUTINE PRINT(NI,NO)
C...
C... SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PRO1P3 TO PRINT THE
C... NUMOL SOLUTION OF EQUATIONS (2.30) TO (2.35)
C...
COMMON/T/           T
1      /Y/    U1(51),   U2(51)
2      /F/    U1T(51),  U2T(51)
3      /R/    X(51),    U1X(51),   U1XX(51),    UE(51),   DIFF(51),
4                  L,        PI,        DX
5      /I/    N

C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L

```

continues

```

C...
C... CALCULATE THE EXACT SOLUTION, AND THE DIFFERENCE BETWEEN THE
C... NUMOL AND EXACT SOLUTIONS
DO 3 I=1,N,10
UE(I)=EXACT(I)
DIFF(I)=U1(I)-UE(I)
3 CONTINUE
C...
C... PRINT THE NUMOL SOLUTION TO EQUATIONS (2.30) TO (2.35)
WRITE(NO,2)    T, (U1(I), I=1,N,10),
1             (UE(I), I=1,N,10),
2             (DIFF(I), I=1,N,10)
2 FORMAT(' TIME = ',F6.2,/,15X,' X=0',5X,'X=0.2',5X,'X=0.4',5X,
1           'X=0.6',5X,'X=0.8',5X,' X=1',/,,
2           , U(X,T)',6F10.6,/,
3           , TE(X,T)',6F10.6,/,
4           , DIFF(X,T)',6F10.6,/)
RETURN
END

REAL FUNCTION EXACT(I)
C...
C... FUNCTION EXACT COMPUTES THE EXACT SOLUTION TO EQUATIONS (2.30)
C... TO (2.35), I.E., EQUATION (2.36), AT GRID INDEX I AND TIME T
C...
COMMON/T/
1     /Y/      U1(51),      U2(51)
2     /F/      U1T(51),     U2T(51)
3     /R/      X(51),       U1X(51),     U1XX(51),      UE(51),    DIFF(51),
4           L,          PI,          DX
5     /I/      N

C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C... EXACT SOLUTION AT X AND T
EXACT=COS(PI*T/L)*SIN(PI*X(I)/L)
RETURN
END

NUMOL SOLUTION OF EQUATIONS (2.30) TO (2.35)
0.        2.0        0.4
102      0.000001
END OF RUNS

```

of data specifies the interval $0 \leq t \leq 2$, which, according to equation (2.36), corresponds to one complete cycle of the solution—note the t dependence of $\cos(a\pi t/L)$ with $a = L = 1$. This cycle of the solution is evident in the numerical output produced by Program 2.6, listed in Table 2.6.

Note the solutions at $t = 0$ and $t = 2$ are essentially the same. Also, a comparison of equations (1.25) and (2.27) indicates a major difference between parabolic and hyperbolic PDEs. For the parabolic PDE, equation (1.21), the solution decays with t , according to the term $e^{-(\pi^2/L^2)t}$ in equation (1.25), so that the solution eventually becomes essentially independent of the initial condition; a practical consequence of this property, as we observed in Section 2.2, is that an inconsistency between the initial and boundary conditions ultimately has little effect (for sufficiently large t), since this type of discontinuity is damped.

Table 2.6 Numerical Solution from Program 2.6

```

RUN NO. -      1    NUMOL SOLUTION OF EQUATIONS (2.30) TO (2.35)

INITIAL T -   0.000E+00

FINAL T -   .200E+01

PRINT T -   .400E+00

NUMBER OF DIFFERENTIAL EQUATIONS - 102

MAXIMUM INTEGRATION ERROR - .100E-05

TIME = 0.00
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
U(X,T) 0.000000  .587785  .951057  .951057  .587785  0.000000
TE(X,T) 0.000000  .587785  .951057  .951057  .587785  .000000
DIFF(X,T) 0.000000  0.000000  0.000000  0.000000  0.000000  .000000

TIME = .40
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
U(X,T) 0.000000  .182098  .294640  .294640  .182098  0.000000
TE(X,T) 0.000000  .181636  .293893  .293893  .181636  .000000
DIFF(X,T) 0.000000  .000462  .000748  .000748  .000462  .000000

TIME = .80
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
U(X,T) 0.000000  -.474956  -.768496  -.768496  -.474956  0.000000
TE(X,T) 0.000000  -.475528  -.769421  -.769421  -.475528  .000000
DIFF(X,T) 0.000000  .000572  .000925  .000925  .000572  .000000

TIME = 1.20
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
U(X,T) 0.000000  -.476384  -.770805  -.770805  -.476384  0.000000
TE(X,T) 0.000000  -.475528  -.769421  -.769421  -.475528  .000000
DIFF(X,T) 0.000000  -.000855  -.001384  -.001384  -.000855  .000000

TIME = 1.60
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
U(X,T) 0.000000  .179786  .290900  .290900  .179786  0.000000
TE(X,T) 0.000000  .181636  .293893  .293893  .181636  .000000
DIFF(X,T) 0.000000  -.001849  -.002993  -.002993  -.001849  .000000

TIME = 2.00
      X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
U(X,T) 0.000000  .587780  .951048  .951048  .587780  0.000000
TE(X,T) 0.000000  .587785  .951057  .951057  .587785  .000000
DIFF(X,T) 0.000000  -.000005  -.000008  -.000008  -.000005  .000000

```

Exactly the opposite is true for the hyperbolic PDE [equation (2.22)]. We see from the solution, equation (2.27), that not only does the initial condition not decay; it actually returns at some later time! In other words, the solution at any $t + 2\pi$ is identical to the solution at t [this follows directly from equation (2.27) with $a = L = 1$]. An important consequence of this property of hyperbolic PDEs is that inconsistencies between initial and boundary conditions (or discontinuities in general) are propagated indefinitely and can be expected to have an important effect throughout

the entire solution (up to whatever final value of t is of interest). This has important implications for computing numerical solutions. We offer the opinion that, for this reason, solutions to hyperbolic PDEs are more difficult to compute than those for parabolic PDEs.

Having covered some general features of the solutions to parabolic and hyperbolic PDEs, we can now complete this introductory discussion by considering some elliptic PDEs. A more detailed discussion of the classification of PDEs will be given in Chapter 6.

2.7 Three PDEs Zero-Order in Time: the Elliptic — Laplace, Poisson, and Helmholtz Equations —

So far, we have considered PDEs first-order in t (Fourier's second law) and second-order in time (the wave equation), which are classified geometrically as *parabolic* and *hyperbolic*, respectively. We can also consider PDEs that are zero-order in t ; these are termed *elliptic*. Three well-known examples are Laplace's equation [equation (2.37)], Poisson's equation [equation (2.38)], and Helmholtz's equation [equation (2.39)].

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad (2.37)$$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \quad (2.38)$$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + u = 0 \quad (2.39)$$

We should first note that x and y are boundary-value independent variables; in other words, equations (2.37) to (2.39) do not have an initial-value independent variable, which would suggest that the NUMOL cannot be used to compute a solution to these equations since the NUMOL is based on the use of initial-value ODEs. However, we shall still be able to compute solutions by a method to be discussed.

First, though, we must specify the auxiliary conditions for equations (2.37) to (2.39); since x and y are boundary-value independent variables, we require two boundary conditions for each

$$u(0, y) = 0, \quad u(1, y) = \sinh(a) \sin(by) \quad (2.40) \quad (2.41)$$

$$u(x, 0) = 0, \quad u(x, 1) = 0 \quad (2.42) \quad (2.43)$$

where a and b are constants to be selected.

If we take $b = \pi$ and $a = b$ [equation (2.37)], $a = (b^2 + 1)^{1/2}$ [equation (2.38)], and $a = (b^2 - 1)^{1/2}$ [equation (2.39)], then the solution to equations (2.37), (2.38), and (2.39) is

$$u(x, y) = \sinh(ax) \sin(by) \quad (2.44)$$

We now consider how we can compute a NUMOL solution to equations (2.37) to (2.39) when these equations do not have an initial-value independent variable that can be used in a set of ODEs. The answer is: We will create an initial-value independent variable, which is not part of the original equations, but will have an insignificant effect on the final numerical solutions. Thus, we will write equations (2.37) to (2.39) as parabolic PDEs:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \quad (2.45)$$

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - f(x, y) \quad (2.46)$$

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + u \quad (2.47)$$

Equation (2.45) is Fourier's second law in two-dimensional Cartesian coordinates, and, of course, equations (2.46) and (2.47) are variants of equation (2.45).

Now, we can write a NUMOL code for equations (2.45) to (2.47), subject to boundary conditions (2.40) to (2.43). Then, we will allow the solution to proceed to essentially infinite t , at which point, if the solution is stable (which it should be for a linear parabolic PDE), $\partial u / \partial t = 0$, and we have again equations (2.37) to (2.39). Since t is not part of the original problem, this approach is called the *method of false transients*. However, before we proceed with the NUMOL coding, we must also specify an initial condition for equations (2.45) to (2.47). As we have seen in Section 1.5, the effect of the initial condition in a parabolic problem decays to insignificance, so we can choose any initial condition to start the calculation, since its effect at $t = \infty$ should be inconsequential. We therefore choose something simple, such as

$$u(x, y, t) = 1 \quad (2.48)$$

Note that we have added the argument t to equation (2.48) since it applies to equations (2.45) to (2.47) [but not to equations (2.37) to (2.39)].

Now we can proceed to a NUMOL code. Subroutines INITAL, DERV, and PRINT, plus data, are listed in Program 2.7.

Program 2.7 Subroutines INITIAL, DERV, PRINT, and Data for Equations (2.37) to (2.43)

```

SUBROUTINE INITIAL
C...
C... SUBROUTINE INITIAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C... INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (2.37) TO
C... (2.43) (VIA EQUATIONS (2.45) TO (2.47))
C...
COMMON/T/           T,      NSTOP,      NORUN
1     /Y/   U(21,21)
2     /F/   UT(21,21)
3     /S/UXX(21,21), UYY(21,21),      X(21),      Y(21),
4             US(21),    USX(21),    USY(21),    USXX(21),   USYY(21),
5             PI,       C(3),       XL,        YL
6     /I/   NX,        NY

C...
C... CONSTANTS USED IN THE THREE PDE PROBLEMS
PI=ACOS(-1.)
C(1)=PI
C(2)=SQRT(PI**2+1.0)
C(3)=SQRT(PI**2-1.0)

C...
C... SPATIAL PARAMETERS
NX=21
NY=21
XL=1.0
YL=1.0
DX=XL/FLOAT(NX-1)
DY=YL/FLOAT(NY-1)

C...
C... INITIALIZATION OVER THE SPATIAL GRID
DO 1 I=1,NX
DO 1 J=1,NY

C...
C... SPATIAL GRID
X(I)=DX*FLOAT(I-1)
Y(J)=DY*FLOAT(J-1)

C...
C... INITIAL CONDITION (2.48)
U(I,J)=1.0
1 CONTINUE
RETURN
END

SUBROUTINE DERV
C...
C... SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (2.45) TO (2.47)
C...
COMMON/T/           T,      NSTOP,      NORUN
1     /Y/   U(21,21)
2     /F/   UT(21,21)
3     /S/UXX(21,21), UYY(21,21),      X(21),      Y(21),
4             US(21),    USX(21),    USY(21),    USXX(21),   USYY(21),
5             PI,       C(3),       XL,        YL
6     /I/   NX,        NY

C...
C... BOUNDARY CONDITION AT X = 0, EQUATION (2.40)
DO 1 J=1,NY
1 U(1,J)=0.

C...
C... BOUNDARY CONDITION AT X = 1, EQUATION (2.41)
DO 2 J=1,NY

```

continues

78 2. Some Applications of the Numerical Method of Lines

```

2      U(NX,J)=SINH(C(NORUN)*X(NX))*SIN(PI*Y(J))
C...
C...  BOUNDARY CONDITION AT Y = 0, EQUATION (2.42)
DO 3 I=1,NX
3      U(I,1)=0.
C...
C...  BOUNDARY CONDITION AT Y = 1, EQUATION (2.43)
DO 4 I=1,NX
4      U(I,NY)=0.
C...
C...  UX
C...
C...      0 LE Y LE 1
DO 5 J=1,NY
C...
C...      0 LE X LE 1
DO 6 I=1,NX
C...
C...      TRANSFER U(I,J) TO US(I)
6      US(I)=U(I,J)
C...
C...      USX
CALL DSS002(0.,XL,NX,US,USX)
C...
C...      USXX
CALL DSS002(0.,XL,NX,USX,USXX)
C...
C...      TRANSFER USXX TO UX
DO 7 I=1,NX
7      UX(I,J)=USXX(I)
5      CONTINUE
C...
C...  UY
C...
C...      0 LE X LE 1
DO 8 I=1,NX
C...
C...      0 LE Y LE 1
DO 9 J=1,NY
C...
C...      TRANSFER U(I,J) TO US(I)
9      US(J)=U(I,J)
C...
C...      USY
CALL DSS002(0.,YL,NY,US,USY)
C...
C...      USYY
CALL DSS002(0.,YL,NY,USY,USYY)
C...
C...      TRANSFER USYY TO UY
DO 10 J=1,NY
10     UY(I,J)=USYY(J)
8      CONTINUE
C...
C...  LAPLACE'S EQUATION VIA EQUATION (2.45)
IF(NORUN.EQ.1)THEN
DO 11 I=1,NX
DO 11 J=1,NY
      UT(I,J)=UX(I,J)+UY(I,J)
11     CONTINUE
C...

```

continues

```

C... POISSON'S EQUATION VIA EQUATION (2.46)
      ELSE IF(NORUN.EQ.2)THEN
          DO 12 I=1,NX
          DO 12 J=1,NY
              UT(I,J)=UXX(I,J)+UYY(I,J)-SINH(C(NORUN)*X(I))*SIN(PI*Y(J))
12      CONTINUE
C...
C... HELMHOLTZ'S EQUATION VIA EQUATION (2.47)
      ELSE IF(NORUN.EQ.3)THEN
          DO 13 I=1,NX
          DO 13 J=1,NY
              UT(I,J)=UXX(I,J)+UYY(I,J)+U(I,J)
13      CONTINUE
      END IF
      RETURN
      END

      SUBROUTINE PRINT(NI,NO)
C...
C... SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PRO1P3 TO PRINT THE
C... NUMOL SOLUTION OF EQUATIONS (2.37) TO (2.43)
C...
      COMMON/T/, T, NSTOP, NORUN
      1 /Y/ U(21,21)
      2 /F/ UT(21,21)
      3 /S/ UXX(21,21), UYY(21,21), X(21), Y(21),
      4 US(21), USX(21), USY(21), USXX(21), USYY(21),
      5 PI, C(3), XL, YL
      6 /I/ NX, NY
C...
C... DIMENSION THE ARRAYS FOR THE EXACT SOLUTION
      REAL UE(21,21), DIFF(21,21)
C...
C... CALCULATE THE EXACT SOLUTION, AND THE DIFFERENCE BETWEEN THE
C... NUMOL AND EXACT SOLUTIONS
      DO 4 J=NY,1,-4
      DO 4 I=1,NX,4
          UE(I,J)=EXACT(I,J)
          DIFF(I,J)=U(I,J)-UE(I,J)
4      CONTINUE
C...
C... PRINT THE NUMOL SOLUTION TO EQUATIONS (2.45) TO (2.47)
      WRITE(NO,1)T
1      FORMAT(' TIME' = ',F6.2,/,20X,' X=0',4X,' X=0.2',4X,' X=0.4',4X,
1           ' X=0.6',4X,' X=0.8',4X,' X=1',/)
      DO 3 J=NY,1,-4
          WRITE(NO,2)Y(J),(U(I,J),I=1,NX,4),
1           (UE(I,J),I=1,NX,4),
2           (DIFF(I,J),I=1,NX,4),
3           (UT(I,J),I=1,NX,4)
2      FORMAT(' Y = ',F3.1,' U(X,Y)',6F9.4,/,
1 9X,'UE(X,Y)',6F9.4,/,12X,'DIFF',6F9.4,/,
2 9X,'UT(X,Y)',6F9.1,/)
3      CONTINUE
      RETURN
      END

      REAL FUNCTION EXACT(I,J)
C...
C... FUNCTION EXACT COMPUTES THE EXACT SOLUTION TO EQUATIONS (2.37)
C... TO (2.43), I.E., EQUATION (2.44), AT GRID INDICES I AND J

```

continues

```

C...
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/   U(21,21)
2      /F/   UT(21,21)
3      /S/UXX(21,21),UYY(21,21),      X(21),      Y(21),
4          US(21),    USX(21),    USY(21),    USXX(21),  USYY(21),
5          PI,        C(3),       XL,         YL
6      /I/   NX,        NY

C...
C... EXACT SOLUTION AT X AND Y
EXACT=SINH(C(NORUN)*X(I))*SIN(PI*Y(J))
RETURN
END

LAPLACE'S EQUATION, UXX + UYY = 0
0.      1.0      1.0
441      0.000001
POISSON'S EQUATION, UXX + UYY = F(X,Y)
0.      1.0      1.0
441      0.000001
HELMHOLTZ'S EQUATION, UXX + UYY + U = 0
0.      1.0      1.0
441      0.000001
END OF RUNS

```

Several points should be noted concerning Program 2.7:

(1) Equations (2.37) to (2.39) are accommodated with a single program by using the run counter, NORUN, in COMMON/T/. When the first set of data is read, the main program PRO1P3 sets NORUN = 1, and the code for equation (2.37) is executed. When the solution to equation (2.37) [via equation (2.45)] is complete (at $t = 1$), the second set of data is read, NORUN = 2, and the code for equation (2.38) [via equation (2.46)], is executed, etc. Thus, by using multiple sets of data, the programming (and the data, if desired) can be changed from run to run (in fact, the solution from one run can be used as input to the next run, typically as initial conditions).

(2) The NUMOL solution to equations (2.37) to (2.39) is stored in the two-dimensional array U(21, 21); similarly, the spatial derivatives are stored in UXX(21, 21), and UYY(21, 21), and the "pseudo" time derivative of equations (2.45) to (2.47) is stored in UT (21, 21).

(3) Since subroutine DSS002 can accommodate only one-dimensional arrays, intermediate storage arrays, US(21), USX(21), USXX(21), USY(21), and USYY(21), are used to store the dependent variable and its various spatial derivatives in DERV so that DSS002 can be used to calculate spatial derivatives. This arrangement is somewhat cumbersome, and a better procedure would be to have a spatial differentiation subroutine that handles two-dimensional arrays directly; such a subroutine clearly could easily be written, and we shall, in fact, provide one in later examples (DSS034).

(4) The "pseudo" initial condition (2.48) is implemented in subroutine INITAL.

(5) The numerical and exact initial conditions do not agree (at "TIME" = 0), since the exact "initial condition" is computed by function EXACT, which has no dependence on t [it implements the exact solution to equations (2.37) to (2.39), equation (2.44)].

(6) The data indicate that t starts at $t = 0$ and runs to $t = 1$; the solution is printed only at $t = 0$ and $t = 1$ (the print interval is 1), since we are really interested only in the NUMOL solution at the final value of $t(t = 1)$, which we select to be essentially infinite (some experimentation with this final value of t may be required).

(7) The derivative $\partial u / \partial t (= UT(I, J))$ is printed in subroutine PRINT to indicate how closely the solution has proceeded to the condition $\partial u / \partial t \rightarrow 0$ [for which the solutions to equations (2.45) to (2.47) are also the solutions to equations (2.37) to (2.39)].

(8) By using a 21×21 -point grid in x and y , the NUMOL requires the integration of $21^2 = 441$ ODEs (as specified in the third line of each set of data). Thus the number of ODEs increases very quickly with the number of grid points in each direction of the two-dimensional grid (which is why we did not use a 51×51 -point grid). In order to accommodate 441 ODEs, Program PRO1P3 was modified in two ways:

(8.1) The absolute dimensioning of the arrays was changed from 250 to 450 [this is easily done with an editor that can replace one character string (250) with another character string (450)].

(8.2) The absolute dimensioning of the work array in RKF45, WORK, was increased from WORK(1000) to WORK(3000). This was required in accordance with the sizing formula for WORK in RKF45, $6 * N + 3 = 6 * 441 + 3 = 2649$ (overdimensioning is permissible).

Clearly, higher-dimensional PDEs can quickly lead to large sets of ODEs (for one three-dimensional PDE using a $21 \times 21 \times 21$ -point grid, the total number of ODEs if the PDE is first-order in t is $21^3 = 9261$ ODEs).

The output from Program 2.7 is listed in Table 2.7.

The output in Table 2.7 has two interesting features:

- (1) While the NUMOL solution at $t = 0$ is far removed from the solution given by equation (2.44) [as expected since initial condition (2.48) was chosen arbitrarily, and is just a constant, independent of x and y], the solution at $t = 1$ is relatively close to the exact solution.
- (2) Also, the derivatives in t ($UT(I, J) = u_t(x, y)$) are relatively small at $t = 1$, indicating that the solutions to equations (2.45) to (2.47) are close to the solutions to equations (2.37) to (2.39).

Thus, by converting the original elliptic problems, equations (2.37) to (2.43), to related parabolic equations [equations (2.40) to (2.43), (2.45) to (2.48)], we were able to compute a NUMOL solution to the elliptic problems.

Table 2.7 Numerical Solution from Program 2.7

```

RUN NO. - 1 LAPLACE'S EQUATION, UXX + UYY = 0

INITIAL T - 0.000E+00

FINAL T - .100E+01

PRINT T - .100E+01

NUMBER OF DIFFERENTIAL EQUATIONS - 441

MAXIMUM INTEGRATION ERROR - .100E-05

"TIME" = 0.00
          X=0    X=0.2   X=0.4   X=0.6   X=0.8   X=1
Y = 1.0  U(X,Y)  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      UE(X,Y)  0.00000  .00000   .00000   .00000   .00000   .00000
      DIFF   0.00000  .00000   .00000   .00000   .00000   .00000
      UT(X,Y)  0.00   -500.00  -500.00  -500.00  -500.00  -13.46

Y = .8   U(X,Y)  0.00000  1.00000  1.00000  1.00000  1.00000  6.78818
      UE(X,Y)  0.00000  .39410   .94897   1.89097   3.60438   6.78818
      DIFF   0.00000  .60590   .05103   -.89097  -2.60438   0.00000
      UT(X,Y)  -500.00  0.00     0.00     0.00     0.00     2827.64

Y = .6   U(X,Y)  0.00000  1.00000  1.00000  1.00000  1.00000  10.98350
      UE(X,Y)  0.00000  .63767   1.53547   3.05966   5.83201   10.98350
      DIFF   0.00000  .36233   .53547  -2.05966  -4.83201   0.00000
      UT(X,Y)  -500.00  0.00     0.00     0.00     0.00     4884.24

Y = .4   U(X,Y)  0.00000  1.00000  1.00000  1.00000  1.00000  10.98350
      UE(X,Y)  0.00000  .63767   1.53547   3.05966   5.83201   10.98350
      DIFF   0.00000  .36233   .53547  -2.05966  -4.83201   0.00000
      UT(X,Y)  -500.00  0.00     0.00     0.00     0.00     4884.24

Y = .2   U(X,Y)  0.00000  1.00000  1.00000  1.00000  1.00000  6.78818
      UE(X,Y)  0.00000  .39410   .94897   1.89097   3.60438   6.78818
      DIFF   0.00000  .60590   .05103   -.89097  -2.60438   0.00000
      UT(X,Y)  -500.00  0.00     0.00     0.00     0.00     2827.64

Y = 0.0   U(X,Y)  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      UE(X,Y)  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      DIFF   0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      UT(X,Y)  0.00   -500.00  -500.00  -500.00  -500.00  -13.46

"TIME" = 1.00
          X=0    X=0.2   X=0.4   X=0.6   X=0.8   X=1
Y = 1.0  U(X,Y)  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      UE(X,Y)  0.00000  .00000   .00000   .00000   .00000   .00000
      DIFF   0.00000  .00000   .00000   .00000   .00000   .00000
      UT(X,Y)  0.00   .02     .00     -.17    -1.06   -13.46

Y = .8   U(X,Y)  0.00000  .40063   .96196   1.90916   3.62219   6.78818
      UE(X,Y)  0.00000  .39410   .94897   1.89097   3.60438   6.78818
      DIFF   0.00000  .00653   .01299   .01819   .01781   0.00000
      UT(X,Y)  .14     .00     .00     .00     .00     -1.50

```

continues

Y = .6	U(X,Y)	0.00000	.64823	1.55648	3.08909	5.86082	10.98350
	UE(X,Y)	0.00000	.63767	1.53547	3.05966	5.83201	10.98350
	DIFF	0.00000	.01056	.02101	.02943	.02881	0.00000
	UT(X,Y)	.23	.00	.00	.00	.00	-2.43
Y = .4	U(X,Y)	0.00000	.64823	1.55648	3.08909	5.86082	10.98350
	UE(X,Y)	0.00000	.63767	1.53547	3.05966	5.83201	10.98350
	DIFF	0.00000	.01056	.02101	.02943	.02881	0.00000
	UT(X,Y)	.23	.00	.00	.00	.00	-2.43
Y = .2	U(X,Y)	0.00000	.40063	.96196	1.90916	3.62219	6.78818
	UE(X,Y)	0.00000	.39410	.94897	1.89097	3.60438	6.78818
	DIFF	0.00000	.00653	.01299	.01819	.01781	0.00000
	UT(X,Y)	.14	.00	.00	.00	.00	-1.50
Y = 0.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	DIFF	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UT(X,Y)	0.00	.02	.00	-.17	-1.06	-13.46

RUN NO. - 2 POISSON'S EQUATION, $U_{XX} + U_{YY} = F(X,Y)$

INITIAL T - 0.000E+00

FINAL T - .100E+01

PRINT T - .100E+01

NUMBER OF DIFFERENTIAL EQUATIONS - 441

MAXIMUM INTEGRATION ERROR - .100E-05

"TIME" = 0.00

		X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
Y = 1.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	.00000	.00000	.00000	.00000	.00000
	DIFF	0.00000	.00000	.00000	.00000	.00000	.00000
	UT(X,Y)	0.00	-500.00	-500.00	-500.00	-500.00	-15.72
Y = .8	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	7.93273
	UE(X,Y)	0.00000	.41628	1.02020	2.08399	4.08718	7.93273
	DIFF	0.00000	.58372	-.02020	-1.08399	-3.08718	0.00000
	UT(X,Y)	-500.00	-.42	-1.02	-2.08	-4.09	3380.78
Y = .6	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	12.83543
	UE(X,Y)	0.00000	.67355	1.65072	3.37197	6.61319	12.83543
	DIFF	0.00000	.32645	-.65072	-2.37197	-5.61319	0.00000
	UT(X,Y)	-500.00	-.67	-1.65	-3.37	-6.61	5779.24
Y = .4	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	12.83543
	UE(X,Y)	0.00000	.67355	1.65072	3.37197	6.61319	12.83543
	DIFF	0.00000	.32645	-.65072	-2.37197	-5.61319	0.00000
	UT(X,Y)	-500.00	-.67	-1.65	-3.37	-6.61	5779.24
Y = .2	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	7.93273
	UE(X,Y)	0.00000	.41628	1.02020	2.08399	4.08718	7.93273
	DIFF	0.00000	.58372	-.02020	-1.08399	-3.08718	0.00000
	UT(X,Y)	-500.00	-.42	-1.02	-2.08	-4.09	3380.78

continues

84 2. Some Applications of the Numerical Method of Lines

Y = 0.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	DIFF	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UT(X,Y)	0.00	-500.00	-500.00	-500.00	-500.00	-500.00	-15.72
"TIME" = 1.00								
	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1		
Y = 1.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	.00000	.00000	.00000	.00000	.00000	.00000
	DIFF	0.00000	.00000	.00000	.00000	.00000	.00000	.00000
	UT(X,Y)	0.00	.03	.01	-.17	-1.21	-15.72	
Y = .8	U(X,Y)	0.00000	.42418	1.03604	2.10640	4.10938	7.93273	
	UE(X,Y)	0.00000	.41628	1.02020	2.08399	4.08718	7.93273	
	DIFF	0.00000	.00791	.01584	.02241	.02220	0.00000	
	UT(X,Y)	.18	.00	.00	.00	.00	-2.07	
Y = .6	U(X,Y)	0.00000	.68634	1.67634	3.40822	6.64912	12.83543	
	UE(X,Y)	0.00000	.67355	1.65072	3.37197	6.61319	12.83543	
	DIFF	0.00000	.01279	.02563	.03625	.03593	0.00000	
	UT(X,Y)	.29	.00	.00	.00	.00	-3.36	
Y = .4	U(X,Y)	0.00000	.68634	1.67634	3.40822	6.64912	12.83543	
	UE(X,Y)	0.00000	.67355	1.65072	3.37197	6.61319	12.83543	
	DIFF	0.00000	.01279	.02563	.03625	.03593	0.00000	
	UT(X,Y)	.29	.00	.00	.00	.00	-3.36	
Y = .2	U(X,Y)	0.00000	.42418	1.03604	2.10640	4.10938	7.93273	
	UE(X,Y)	0.00000	.41628	1.02020	2.08399	4.08718	7.93273	
	DIFF	0.00000	.00791	.01584	.02241	.02220	0.00000	
	UT(X,Y)	.18	.00	.00	.00	.00	-2.07	
Y = 0.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	DIFF	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UT(X,Y)	0.00	.03	.01	-.17	-1.21	-15.72	

RUN NO. - 3 HELMHOLTZ'S EQUATION, UXX + UYY + U = 0

INITIAL T - 0.000E+00

FINAL T - .100E+01

PRINT T - .100E+01

NUMBER OF DIFFERENTIAL EQUATIONS - 441

MAXIMUM INTEGRATION ERROR - .100E-05

"TIME" = 0.00	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1		
Y = 1.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
	UE(X,Y)	0.00000	.00000	.00000	.00000	.00000	.00000	
	DIFF	0.00000	.00000	.00000	.00000	.00000	.00000	
	UT(X,Y)	0.00	-500.00	-500.00	-500.00	-500.00	-11.42	
Y = .8	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	5.76068	
	UE(X,Y)	0.00000	.37118	.87799	1.70561	3.15646	5.76068	
	DIFF	0.00000	.62882	.12201	-.70561	-2.15646	0.00000	
	UT(X,Y)	-500.00	1.00	1.00	1.00	1.00	2329.71	

continues

$Y = .6$	$U(X, Y)$	0.00000	1.00000	1.00000	1.00000	1.00000	9.32097
	$UE(X, Y)$	0.00000	.60058	1.42061	2.75974	5.10726	9.32097
	$DIFF$	0.00000	.39942	-.42061	-1.75974	-4.10726	0.00000
	$UT(X, Y)$	-500.00	1.00	1.00	1.00	1.00	4078.56
$Y = .4$	$U(X, Y)$	0.00000	1.00000	1.00000	1.00000	1.00000	9.32097
	$UE(X, Y)$	0.00000	.60058	1.42061	2.75974	5.10726	9.32097
	$DIFF$	0.00000	.39942	-.42061	-1.75974	-4.10726	0.00000
	$UT(X, Y)$	-500.00	1.00	1.00	1.00	1.00	4078.56
$Y = .2$	$U(X, Y)$	0.00000	1.00000	1.00000	1.00000	1.00000	5.76068
	$UE(X, Y)$	0.00000	.37118	.87799	1.70561	3.15646	5.76068
	$DIFF$	0.00000	.62882	.12201	-.70561	-2.15646	0.00000
	$UT(X, Y)$	-500.00	1.00	1.00	1.00	1.00	2329.71
$Y = 0.0$	$U(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UE(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$DIFF$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UT(X, Y)$	0.00	-500.00	-500.00	-500.00	-500.00	-11.42
"TIME" = 1.00		X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
$Y = 1.0$	$U(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UE(X, Y)$	0.00000	.00000	.00000	.00000	.00000	.00000
	$DIFF$	0.00000	.00000	.00000	.00000	.00000	.00000
	$UT(X, Y)$	0.00	.02	.01	-.14	-.91	-11.42
$Y = .8$	$U(X, Y)$	0.00000	.37692	.88925	1.72106	3.17121	5.76068
	$UE(X, Y)$	0.00000	.37118	.87799	1.70561	3.15646	5.76068
	$DIFF$	0.00000	.00574	.01126	.01545	.01475	0.00000
	$UT(X, Y)$.12	.00	.00	.00	.00	-1.04
$Y = .6$	$U(X, Y)$	0.00000	.60986	1.43883	2.78474	5.13112	9.32097
	$UE(X, Y)$	0.00000	.60058	1.42061	2.75974	5.10726	9.32097
	$DIFF$	0.00000	.00928	.01822	.02500	.02386	0.00000
	$UT(X, Y)$.19	.00	.00	.00	.00	-1.68
$Y = .4$	$U(X, Y)$	0.00000	.60986	1.43883	2.78474	5.13112	9.32097
	$UE(X, Y)$	0.00000	.60058	1.42061	2.75974	5.10726	9.32097
	$DIFF$	0.00000	.00928	.01822	.02500	.02386	0.00000
	$UT(X, Y)$.19	.00	.00	.00	.00	-1.68
$Y = .2$	$U(X, Y)$	0.00000	.37692	.88925	1.72106	3.17121	5.76068
	$UE(X, Y)$	0.00000	.37118	.87799	1.70561	3.15646	5.76068
	$DIFF$	0.00000	.00574	.01126	.01545	.01475	0.00000
	$UT(X, Y)$.12	.00	.00	.00	.00	-1.04
$Y = 0.0$	$U(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UE(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$DIFF$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UT(X, Y)$	0.00	.02	.01	-.14	-.91	-11.42

We could possibly conclude that the agreement between the NUMOL and exact solutions at $t = 1$ is not acceptable. This difference is due to the differentiation in space through DSS002 since the integration in t is performed to an accuracy of $t = 0.000001$ (see the error tolerance in the data of Program 2.7; we did confirm that $t = .1$ is essentially infinite since a solution to $t = 2$ did not produce much change in the solution). However, to get better accuracy from DSS002, we must increase the number of

grid points, which, as we have seen, increases very quickly for two-dimensional problems. Therefore, about the only way left to improve the accuracy of the NUMOL solution is to improve the accuracy of the spatial differentiation formulas (in DSS0002). This has already been done, and the improved differentiation formulas are in a subroutine named DSS004. We can call DSS004 merely by replacing CALL DSS002(. . .) with CALL DSS004(. . .) in subroutine DERV of Program 2.7 (the arguments of the two subroutines are the same, and everything else in the programming remains the same). The output produced by the use of subroutine DSS004 is listed in Table 2.8. Clearly the use of DSS004 has made a substantial

Table 2.8 Numerical Solution from Program 2.7 with Subroutine DSS004 Used in Place of DSS002

RUN NO.	-	1	LAPLACE'S EQUATION, $U_{XX} + U_{YY} = 0$					
INITIAL T	-	0.000E+00						
FINAL T	-	.100E+01						
PRINT T	-	.100E+01						
NUMBER OF DIFFERENTIAL EQUATIONS	-	441						
MAXIMUM INTEGRATION ERROR	-	.100E-05						
"TIME"	=	0.00	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
Y = 1.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	
	UE(X,Y)	0.00000	.00000	.00000	.00000	.00000	.00000	
	DIFF	0.00000	.00000	.00000	.00000	.00000	.00000	
	UT(X,Y)	0.00	-1236.11	-1236.11	-1236.11	-1236.11	-1236.11	.29
Y = .8	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	1.00000	6.78818
	UE(X,Y)	0.00000	.39410	.94897	1.89097	3.60438	6.78818	
	DIFF	0.00000	.60590	.05103	-.89097	-2.60438	0.00000	
	UT(X,Y)	-1236.11	-5.56	-2.78	-2.78	13.30	7087.84	
Y = .6	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	1.00000	10.98350
	UE(X,Y)	0.00000	.63767	1.53547	3.05966	5.83201	10.98350	
	DIFF	0.00000	.36233	-.53547	-2.05966	-4.83201	0.00000	
	UT(X,Y)	-1236.11	-2.78	0.00	0.00	27.73	12232.32	
Y = .4	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	1.00000	10.98350
	UE(X,Y)	0.00000	.63767	1.53547	3.05966	5.83201	10.98350	
	DIFF	0.00000	.36233	-.53547	-2.05966	-4.83201	0.00000	
	UT(X,Y)	-1236.11	-2.78	0.00	0.00	27.73	12232.32	
Y = .2	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	1.00000	6.78818
	UE(X,Y)	0.00000	.39410	.94897	1.89097	3.60438	6.78818	
	DIFF	0.00000	.60590	.05103	-.89097	-2.60438	0.00000	
	UT(X,Y)	-1236.11	-5.56	-2.78	-2.78	13.30	7087.84	

continues

$Y = 0.0$	$U(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UE(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$DIFF$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UT(X, Y)$	0.00	-1236.11	-1236.11	-1236.11	-1236.11	.29	
"TIME" = 1.00								
		X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1	
$Y = 1.0$	$U(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UE(X, Y)$	0.00000	.00000	.00000	.00000	.00000	.00000	.00000
	$DIFF$	0.00000	.00000	.00000	.00000	.00000	.00000	.00000
	$UT(X, Y)$	0.00	.00	.00	.00	.03	.29	
$Y = .8$	$U(X, Y)$	0.00000	.39410	.94897	1.89097	3.60439	6.78818	
	$UE(X, Y)$	0.00000	.39410	.94897	1.89097	3.60438	6.78818	
	$DIFF$	0.00000	-.00000	-.00000	-.00000	.00001	0.00000	
	$UT(X, Y)$.01	.00	.00	.00	.00	-.03	
$Y = .6$	$U(X, Y)$	0.00000	.63766	1.53547	3.05967	5.83205	10.98350	
	$UE(X, Y)$	0.00000	.63767	1.53547	3.05966	5.83201	10.98350	
	$DIFF$	0.00000	-.00001	.00000	.00001	.00003	0.00000	
	$UT(X, Y)$.01	.00	.00	.00	.00	-.06	
$Y = .4$	$U(X, Y)$	0.00000	.63766	1.53547	3.05967	5.83205	10.98350	
	$UE(X, Y)$	0.00000	.63767	1.53547	3.05966	5.83201	10.98350	
	$DIFF$	0.00000	-.00001	.00000	.00001	.00003	0.00000	
	$UT(X, Y)$.01	.00	.00	.00	.00	-.06	
$Y = .2$	$U(X, Y)$	0.00000	.39410	.94897	1.89097	3.60439	6.78818	
	$UE(X, Y)$	0.00000	.39410	.94897	1.89097	3.60438	6.78818	
	$DIFF$	0.00000	-.00000	-.00000	-.00000	.00001	0.00000	
	$UT(X, Y)$.01	.00	.00	.00	.00	-.03	
$Y = 0.0$	$U(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UE(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$DIFF$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UT(X, Y)$	0.00	.00	.00	.00	.03	.29	

RUN NO. - 2 POISSON'S EQUATION, $U_{XX} + U_{YY} = F(X, Y)$

INITIAL T - 0.000E+00

FINAL T - .100E+01

PRINT T - .100E+01

NUMBER OF DIFFERENTIAL EQUATIONS - 441

MAXIMUM INTEGRATION ERROR - .100E-05

"TIME" = 0.00								
		X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1	
$Y = 1.0$	$U(X, Y)$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	$UE(X, Y)$	0.00000	.00000	.00000	.00000	.00000	.00000	.00000
	$DIFF$	0.00000	.00000	.00000	.00000	.00000	.00000	.00000
	$UT(X, Y)$	0.00	-1236.11	-1236.11	-1236.11	-1236.11	.34	
$Y = .8$	$U(X, Y)$	0.00000	1.00000	1.00000	1.00000	1.00000	7.93273	
	$UE(X, Y)$	0.00000	.41628	1.02020	2.08399	4.08718	7.93273	
	$DIFF$	0.00000	.58372	-.02020	-1.08399	-3.08718	0.00000	
	$UT(X, Y)$	-1236.11	-5.97	-3.80	-4.86	12.39	8483.41	

continues

88 2. Some Applications of the Numerical Method of Lines

Y = .6	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	12.83543
	UE(X,Y)	0.00000	.67355	1.65072	3.37197	6.61319	12.83543
	DIFF	0.00000	.32645	-.65072	-2.37197	-5.61319	0.00000
	UT(X,Y)	-1236.11	-3.45	-1.65	-3.37	26.26	14490.40
Y = .4	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	12.83543
	UE(X,Y)	0.00000	.67355	1.65072	3.37197	6.61319	12.83543
	DIFF	0.00000	.32645	-.65072	-2.37197	-5.61319	0.00000
	UT(X,Y)	-1236.11	-3.45	-1.65	-3.37	26.26	14490.40
Y = .2	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	7.93273
	UE(X,Y)	0.00000	.41628	1.02020	2.08399	4.08718	7.93273
	DIFF	0.00000	.58372	-.02020	-1.08399	-3.08718	0.00000
	UT(X,Y)	-1236.11	-5.97	-3.80	-4.86	12.39	8483.41
Y = 0.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	DIFF	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UT(X,Y)	0.00	-1236.11	-1236.11	-1236.11	-1236.11	.34
"TIME" = 1.00		X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
Y = 1.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	.00000	.00000	.00000	.00000	.00000
	DIFF	0.00000	.00000	.00000	.00000	.00000	.00000
	UT(X,Y)	0.00	.00	.00	.00	.03	.34
Y = .8	U(X,Y)	0.00000	.41627	1.02019	2.08398	4.08718	7.93273
	UE(X,Y)	0.00000	.41628	1.02020	2.08399	4.08718	7.93273
	DIFF	0.00000	-.00001	-.00001	-.00001	.00000	0.00000
	UT(X,Y)	.01	.00	.00	.00	.00	-.05
Y = .6	U(X,Y)	0.00000	.67354	1.65070	3.37196	6.61322	12.83543
	UE(X,Y)	0.00000	.67355	1.65072	3.37197	6.61319	12.83543
	DIFF	0.00000	-.00001	-.00001	-.00001	.00003	0.00000
	UT(X,Y)	.02	.00	.00	.00	.00	-.09
Y = .4	U(X,Y)	0.00000	.67354	1.65070	3.37196	6.61322	12.83543
	UE(X,Y)	0.00000	.67355	1.65072	3.37197	6.61319	12.83543
	DIFF	0.00000	-.00001	-.00001	-.00001	.00003	0.00000
	UT(X,Y)	.02	.00	.00	.00	.00	-.09
Y = .2	U(X,Y)	0.00000	.41627	1.02019	2.08398	4.08718	7.93273
	UE(X,Y)	0.00000	.41628	1.02020	2.08399	4.08718	7.93273
	DIFF	0.00000	-.00001	-.00001	-.00001	.00000	0.00000
	UT(X,Y)	.01	.00	.00	.00	.00	-.05
Y = 0.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	DIFF	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UT(X,Y)	0.00	.00	.00	.00	.03	.34

RUN NO. - 3 HELMHOLTZ'S EQUATION, $U_{XX} + U_{YY} + U = 0$

INITIAL T - 0.000E+00

FINAL T - .100E+01

PRINT T - .100E+01

NUMBER OF DIFFERENTIAL EQUATIONS - 441

MAXIMUM INTEGRATION ERROR - .100E-05

continues

"TIME" = 0.00		X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
Y = 1.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	.00000	.00000	.00000	.00000	.00000
	DIFF	0.00000	.00000	.00000	.00000	.00000	.00000
	UT(X,Y)	0.00	-1236.11	-1236.11	-1236.11	-1236.11	.25
Y = .8	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	5.76068
	UE(X,Y)	0.00000	.37118	.87799	1.70561	3.15646	5.76068
	DIFF	0.00000	.62882	.12201	-.70561	-2.15646	0.00000
	UT(X,Y)	-1236.11	-4.56	-1.78	-1.78	11.45	5833.63
Y = .6	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	9.32097
	UE(X,Y)	0.00000	.60058	1.42061	2.75974	5.10726	9.32097
	DIFF	0.00000	.39942	-.42061	-1.75974	-4.10726	0.00000
	UT(X,Y)	-1236.11	-1.78	1.00	1.00	24.11	10202.97
Y = .4	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	9.32097
	UE(X,Y)	0.00000	.60058	1.42061	2.75974	5.10726	9.32097
	DIFF	0.00000	.39942	-.42061	-1.75974	-4.10726	0.00000
	UT(X,Y)	-1236.11	-1.78	1.00	1.00	24.11	10202.97
Y = .2	U(X,Y)	0.00000	1.00000	1.00000	1.00000	1.00000	5.76068
	UE(X,Y)	0.00000	.37118	.87799	1.70561	3.15646	5.76068
	DIFF	0.00000	.62882	.12201	-.70561	-2.15646	0.00000
	UT(X,Y)	-1236.11	-4.56	-1.78	-1.78	11.45	5833.63
Y = 0.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	DIFF	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UT(X,Y)	0.00	-1236.11	-1236.11	-1236.11	-1236.11	.25
"TIME" = 1.00		X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
Y = 1.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	.00000	.00000	.00000	.00000	.00000
	DIFF	0.00000	.00000	.00000	.00000	.00000	.00000
	UT(X,Y)	0.00	.00	.00	.00	.02	.25
Y = .8	U(X,Y)	0.00000	.37118	.87799	1.70562	3.15647	5.76068
	UE(X,Y)	0.00000	.37118	.87799	1.70561	3.15646	5.76068
	DIFF	0.00000	.00000	.00000	.00001	.00001	0.00000
	UT(X,Y)	.01	.00	.00	.00	.00	-.02
Y = .6	U(X,Y)	0.00000	.60058	1.42062	2.75976	5.10730	9.32097
	UE(X,Y)	0.00000	.60058	1.42061	2.75974	5.10726	9.32097
	DIFF	0.00000	.00000	.00001	.00002	.00004	0.00000
	UT(X,Y)	.01	.00	.00	.00	.00	-.04
Y = .4	U(X,Y)	0.00000	.60058	1.42062	2.75976	5.10730	9.32097
	UE(X,Y)	0.00000	.60058	1.42061	2.75974	5.10726	9.32097
	DIFF	0.00000	.00000	.00001	.00002	.00004	0.00000
	UT(X,Y)	.01	.00	.00	.00	.00	-.04
Y = .2	U(X,Y)	0.00000	.37118	.87799	1.70562	3.15647	5.76068
	UE(X,Y)	0.00000	.37118	.87799	1.70561	3.15646	5.76068
	DIFF	0.00000	.00000	.00000	.00001	.00001	0.00000
	UT(X,Y)	.01	.00	.00	.00	.00	-.02
Y = 0.0	U(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UE(X,Y)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	DIFF	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	UT(X,Y)	0.00	.00	.00	.00	.02	.25

improvement in the accuracy of the NUMOL solution. Whereas the solution from DSS002 was correct to two to three significant figures, the solution from DSS004 is correct to five to six significant figures. As we shall see in Chapter 3, this improvement in the accuracy of the NUMOL solution is obtained at little additional computational effort (we shall examine the differentiation formulas in subroutines DSS002 and DSS004 in detail).

We have now applied the NUMOL to PDEs that are zeroth-, first-, and second-order in time, with the geometric classification of elliptic, parabolic, and hyperbolic, respectively, so that we see the NUMOL is, in principle, capable of handling all three major classes of PDEs. Finally, we conclude this chapter with one more example that clearly demonstrates the flexibility and versatility of the NUMOL.

2.8 Two Nonlinear PDEs

Consider the following system of one-dimensional nonlinear PDEs [Madsen (2.1)]

$$u_t = (v - 1)u_x + (16xt - 2t - 16(v - 1))(u - 1) + 10xe^{-4x} \quad (2.49)$$

$$v_t = v_{xx} + u_x + 4u - 4 + x^2 - 2t - 10te^{-4x} \quad (2.50)$$

with the initial conditions

$$u(x, 0) = v(x, 0) = 1 \quad (2.51) \quad (2.52)$$

and the boundary conditions

$$u(0, t) = v(0, t) = 1 \quad (2.53) \quad (2.54)$$

$$3u(1, t) + u_x(1, t) = 3, \quad 5v_x(1, t) = e^4(u(1, t) - 1) \quad (2.55) \quad (2.56)$$

The exact solution to equations (2.49) to (2.56) is

$$u(x, t) = 1 + 10xte^{-4x}, \quad v(x, t) = 1 + x^2t \quad (2.57) \quad (2.58)$$

We can note several interesting points concerning this problem:

(1) Equation (2.49) is nonlinear; note in particular the terms $((v - 1)u_x)_x$ and $(v - 1)(u - 1)$ involving products of the dependent variables and their derivatives.

(2) Equations (2.49) and (2.50) have variable coefficients, that is, coefficients multiplying the dependent variable that are functions of the independent variables x and t , such as $(16xt - 2t - 16(v - 1))(u - 1)$.

(3) Equations (2.49) and (2.50) have *nonhomogeneous* terms, that is, terms that are functions of the independent variables, x and t , such as $10te^{-4x}$.

(4) Equations (2.49) has the second-order spatial derivative u_{xx} [from expanding the term $(v - 1)u_x)_x$] which, in combination with u_t , suggests it is parabolic in u . Equation (2.50) has the second-order spatial derivative v_{xx} and, in combination with v_t , suggests that it is parabolic. Also, equation (2.49) has the first-order spatial derivative v_x [multiplied by u_x from expanding the term $(v - 1)u_x)_x$], and equation (2.50) has the first-order spatial derivative u_x , suggesting that equations (2.49) and (2.50) are *first-order hyperbolic PDEs* [see also problem (2.5)]. Thus, equations (2.49) and (2.50) can be considered as *parabolic-hyperbolic*, that is, of *mixed type*.

(5) Boundary conditions (2.53) and (2.54) are of the Dirichlet type, while boundary conditions (2.55) and (2.56) are of the third type.

Thus, equations (2.49) to (2.56) have a variety of properties; yet, as we shall see, these equations are easily solved by the NUMOL, indicating the flexibility and versatility of the NUMOL approach to PDEs. Also, we offer the opinion that this problem would be considerably more difficult to program and solve by more conventional numerical methods such as finite differences.

Subroutines INITAL, DERV, and PRINT and the data for equations (2.49) to (2.58) are listed in Program 2.8. Two runs are programmed (see the data) for the calculation of the spatial derivatives in equations (2.49) to (2.56) using DSS002 (NORUN = 1) and DSS004 (NORUN = 2). Note in particular the ease of programming the PDEs, equations (2.49) and (2.50) in DO loop 3 of subroutine DERV.

Program 2.8 Subroutines INITAL, DERV, PRINT, and Data for Equations (2.49) to (2.58)

```

SUBROUTINE INITAL
C...
C... SUBROUTINE INITAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C... INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (2.49) TO
C... (2.56)
C...
COMMON/T/      T,    NSTOP,   NORUN
1      /Y/    U(21),   V(21)
2      /F/    UT(21),  VT(21)
3      /SD/   UX(21),UXX(21),  VX(21),VXX(21)
4      /X/    X(21)
5      /P/     N,       E4
C...
C... NUMBER OF GRID POINTS AND SPATIAL INCREMENT
N=21
DX=1./FLOAT(N-1)
C...
C... VALUES OF X ALONG THE SPATIAL GRID
DO 1 I=1,N
X(I)=DX*FLOAT(I-1)
1  CONTINUE

```

continues

```

C...
C... INITIAL CONDITIONS, EQUATIONS (2.51) AND (2.52)
DO 2 I=1,N
U(I)=1.
V(I)=1.
2 CONTINUE
C...
C... CONSTANT E**4 USED IN BOUNDARY CONDITION (2.56)
E4=EXP(1.)**4
RETURN
END

SUBROUTINE DERV
C...
C... SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (2.49) TO (2.56)
C...
COMMON/T/      T,    NSTOP,   NORUN
1      /Y/    U(21),  V(21)
2      /F/    UT(21), VT(21)
3      /SD/   UX(21),UXX(21), VX(21),VXX(21)
4      /X/    X(21)
5      /P/     N,       E4
C...
C... BOUNDARY CONDITIONS AT X = 0, EQUATIONS (2.53) AND (2.54)
U(1)=1.
UT(1)=0.
V(1)=1.
VT(1)=0.
C...
C... FIRST ORDER SPATIAL DERIVATIVES
XL=X(1)
XU=X(N)
C...
C... THREE POINT CENTERED DIFFERENCES
IF(NORUN.EQ.1)CALL DSS002(XL,XU,N,U,UX)
IF(NORUN.EQ.1)CALL DSS002(XL,XU,N,V,VX)
C...
C... FIVE POINT CENTERED DIFFERENCES
IF(NORUN.EQ.2)CALL DSS004(XL,XU,N,U,UX)
IF(NORUN.EQ.2)CALL DSS004(XL,XU,N,V,VX)
C...
C... BOUNDARY CONDITIONS AT X = 1, EQUATIONS (2.55) AND (2.56)
UX(N)=3.-3.*U(N)
VX(N)=E4*(U(N)-1.)/5.
C...
C... SECOND ORDER SPATIAL DERIVATIVES
C...
C... THREE POINT CENTERED DIFFERENCES
IF(NORUN.EQ.1)CALL DSS002(XL,XU,N,VX,VXX)
C...
C... FIVE POINT CENTERED DIFFERENCES
IF(NORUN.EQ.2)CALL DSS004(XL,XU,N,VX,VXX)
C...
C... NOTE THAT ARRAY VX IS USED AS TEMPORARY STORAGE IN THE CALCULATION
C... OF THE TERM ((V-1)*U ) WHICH IS FINALLY STORED IN ARRAY UXX
C...          X X
DO 1 I=1,N
VX(I)=(V(I)-1.)*UX(I)
1 CONTINUE
IF(NORUN.EQ.1)CALL DSS002(XL,XU,N,VX,UXX)
IF(NORUN.EQ.2)CALL DSS004(XL,XU,N,VX,UXX)

```

continues

```

C...
C... PDES, EQUATIONS (2.49) AND (2.50)
DO 3 I=2,N
EX=EXP(-4.*X(I))
UT(I)=UXX(I)+(16.*X(I)*T-2.*T-16.*V(I)-1.)*(U(I)-1.)+10.*X(I)*EX
VT(I)=VXX(I)+UX(I)+4.*U(I)-4.+X(I)**2-2.*T-10.*T*EX
3 CONTINUE
RETURN
END

SUBROUTINE PRINT(NI,NO)
C...
C... SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PRO1P3 TO PRINT THE
C... NUMOL SOLUTION OF EQUATIONS (2.49) TO (2.56)
C...
COMMON/T/      T, NSTOP, NORUN
1      /Y/   U(21), V(21)
2      /F/   UT(21), VT(21)
3      /SD/  UX(21), UXX(21), VX(21), VXX(21)
4      /X/   X(21)
5      /P/   N, E4
6      /A/   UA(21), DU(21), VA(21), DV(21)
C...
C... PRINT A HEADING FOR THE NUMERICAL AND ANALYTICAL SOLUTIONS
IF(T.LT.0.01)WRITE(NO,1)
1 FORMAT(' U = NUMERICAL U(X,T)', '/',
1       ' UA = ANALYTICAL U(X,T)', '/',
2       ' DIFF U = U - UA ', '/',
3       ' V = NUMERICAL V(X,T)', '/',
4       ' VA = ANALYTICAL V(X,T)', '/',
5       ' DIFF V = V - VA ', '/',
6       ' 3X, 'T', 6X, 'X', 'U', 7X, 'UA', 5X, 'DIFF U',
7       ' 8X, 'V', 7X, 'UA', 5X, 'DIFF V')
C...
C... ANALYTICAL SOLUTIONS (2.57) AND (2.58), DIFFERENCES BETWEEN THE
C... NUMERICAL AND ANALYTICAL SOLUTIONS
DO 2 I=1,N,10
UA(I)=1.+10.*X(I)*T*EXP(-4.*X(I))
DU(I)=U(I)-UA(I)
VA(I)=1.+(X(I)**2)*T
DV(I)=V(I)-VA(I)
2 CONTINUE
C...
C... PRINT THE NUMERICAL SOLUTION, ANALYTICAL SOLUTION, DIFFERENCE
C... BETWEEN THE SOLUTIONS
WRITE(NO,3)(T,X(I),U(I),UA(I),DU(I),V(I),VA(I),DV(I),I=1,N,10)
3 FORMAT(F4.1,F7.3,2F9.4,E11.3,2F9.4,E11.3)
WRITE(NO,4)
4 FORMAT(' ')
RETURN
END

MADSEN, ET AL, NUM METH DIFF SYS, PP 238-239, DSS002
0.        2.0      1.0
42          0.00001
MADSEN, ET AL, NUM METH DIFF SYS, PP 238-239, DSS004
0.        2.0      1.0
42          0.00001
END OF RUNS

```

As expected, the numerical solution is closer to the exact solution [equations (2.57) and (2.58) programmed in subroutine PRINT] for DSS004 than for DSS002, as shown in Table 2.9.

We have now developed a series of NUMOL solutions for elliptic,

Table 2.9 Numerical Solution from Program 2.8

RUN NO. - 1 MADSEN, ET AL, NUM METH DIFF SYS, PP 238-239, DSS002

INITIAL T - 0.000E+00

FINAL T - .200E+01

PRINT T - .100E+01

NUMBER OF DIFFERENTIAL EQUATIONS - 42

MAXIMUM INTEGRATION ERROR - .100E-04

U = NUMERICAL U(X,T)

UA = ANALYTICAL U(X,T)

DIFF U = U - UA

V = NUMERICAL V(X,T)

VA = ANALYTICAL V(X,T)

DIFF V = V - VA

T	X	U	UA	DIFF U	V	UA	DIFF V
0.0	0.000	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
0.0	.500	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
0.0	1.000	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
1.0	0.000	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
1.0	.500	1.6723	1.6767	-.438E-02	1.2506	1.2500	.584E-03
1.0	1.000	1.1825	1.1832	-.683E-03	1.9974	2.0000	-.262E-02
2.0	0.000	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
2.0	.500	2.3405	2.3534	-.129E-01	1.5002	1.5000	.158E-03
2.0	1.000	1.3647	1.3663	-.160E-02	2.9917	3.0000	-.828E-02

RUN NO. - 2 MADSEN, ET AL, NUM METH DIFF SYS, PP 238-239, DSS004

INITIAL T - 0.000E+00

FINAL T - .200E+01

PRINT T - .100E+01

NUMBER OF DIFFERENTIAL EQUATIONS - 42

MAXIMUM INTEGRATION ERROR - .100E-04

U = NUMERICAL U(X,T)

UA = ANALYTICAL U(X,T)

DIFF U = U - UA

V = NUMERICAL V(X,T)

VA = ANALYTICAL V(X,T)

DIFF V = V - VA

T	X	U	UA	DIFF U	V	UA	DIFF V
0.0	0.000	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
0.0	.500	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
0.0	1.000	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
1.0	0.000	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
1.0	.500	1.6769	1.6767	.223E-03	1.2500	1.2500	.309E-04
1.0	1.000	1.1831	1.1832	-.408E-04	2.0000	2.0000	.155E-04
2.0	0.000	1.0000	1.0000	0.000E+00	1.0000	1.0000	0.000E+00
2.0	.500	2.3536	2.3534	.283E-03	1.5003	1.5000	.273E-03
2.0	1.000	1.3661	1.3663	-.194E-03	2.9999	3.0000	-.870E-04

parabolic, and hyperbolic PDEs, both linear and nonlinear, in multidimensional space and time. If we consider the essential steps in developing NUMOL solutions, we see that there are essentially two:

- (1) The calculation of the spatial derivatives, which we consider in some detail in Chapter 3.
- (2) The integration of the initial-value ODEs resulting from the algebraic approximation of the spatial derivatives, which we consider in some detail in Chapter 4.

After covering these two steps in the NUMOL programming for a PDE problem, we consider some applications of increasing complexity in Chapters 5 and 6.

References

- (2.1) Madsen, N. K., and R. F. Sincovec, "Software for Partial Differential Equations," in *Numerical Methods for Differential Systems*, L. Lapidus and W. E. Schiesser, eds., Academic Press, New York, 1976.

Problems

- (2.1) Derive equation (2.11) by writing an energy balance on an incremental cylinder of radius r and thickness Δr , followed by the limit $\Delta r \rightarrow 0$. Neglect heat conduction in the axial (z) and angular (θ) directions, so that the model is one-dimensional (in r .)
- (2.2) Consider again equation (2.11) (renumbered here for this problem)

$$T_t = \alpha(T_{rr} + (1/r)T_r) \quad (2.2a)$$

with the initial and boundary conditions

$$T(r, 0) = T_0 \quad (2.2b)$$

$$T(r_1, t) = T_1, \quad T(r_2, t) = T_2 \quad (2.2c) \quad (2.2d)$$

Modify Program 2.4 for initial condition (2.2b) and boundary conditions (2.2c) and (2.2d) (rather than (2.12), (2.13), and (2.14)). Execute the program for $T_0 = T_1 = 25$, $T_2 = 500$, $r_1 = 0.5$, $r_2 = 1$. Compare the long-time NUMOL solution with the infinite-time exact solution.

- (2.3) Modify Program 2.5 so that the conduction term in equation (2.17), $\alpha(T_{rr} + (2/r)T_r)$, and the reaction term, $\beta e^{-E/(RT)}$, are computed and

printed individually, along with the solution $T(r, t)$ (note that everything required to compute these terms is available in COMMON in subroutine PRINT). The intention is to observe the relative contributions of the heat conduction and reaction to the time derivative T_t in equation (2.17), and thereby gain some insight into when the solution appears to become unstable (the heat from the reaction cannot be removed fast enough by the radial conduction, thereby leading to a temperature rise that in turn accelerates the rate of reaction giving more heat, etc.).

- (2.4) Verify that equation (2.27) is the solution to equations (2.22) to (2.26).

- (2.5) Consider the two first-order PDEs

$$u_t = v_x, \quad v_t = u_x \quad (2.5a) \quad (2.5b)$$

Equations (2.5a) and (2.5b) are termed *first-order hyperbolic* PDEs.

Show that equations (2.5a) and (2.5b) can be combined to give the second-order hyperbolic wave equation, equation (2.21) with $a = 1$. This can be done by either eliminating u from equations (2.5a) and (2.5b) or by eliminating v .

If the initial and boundary conditions for equations (2.5a) and (2.5b) are

$$u(x, 0) = 0, \quad v(x, 0) = \cos(\pi x/2) \quad (2.5c) \quad (2.5d)$$

$$u(0, t) = 0, \quad v(1, t) = 1 \quad (2.5e) \quad (2.5f)$$

Show that the exact solution for $u(x, t)$ is

$$u(x, t) = -\sin(\pi x/2) \sin(\pi t/2) \quad (2.5g)$$

What is the corresponding exact solution for $v(x, t)$?

- (2.6) Prepare a NUMOL program for the solution for equations (2.5a) to (2.5f) using DSS004. Compare the NUMOL solution with the exact solution, equation (2.5g).
- (2.7) Prepare a NUMOL program for the solution of equations (2.5a) to (2.5f) after v has been eliminated from these equations, that is, by solving the second-order wave equation (reference to Section 2.6 should help with this problem). Compare the NUMOL solution with the exact solution.

3

Spatial Differentiation

In Chapters 1 and 2 we observed that the calculation of the spatial derivatives (the derivatives with respect to the boundary-value independent variables) is one of two principal steps in the NUMOL integration of PDEs (the other step is the numerical integration of the ODEs in the initial-value independent variable). We also observed that the accuracy of the NUMOL solutions is determined primarily by the accuracy of the spatial differentiation. For example, subroutine DSS002 produced NUMOL solutions of acceptable accuracy until we studied the three elliptic PDEs in Section 2.7; there we observed that the NUMOL solutions from DSS002 were of marginal accuracy, but by switching to subroutine DSS004, which has more accurate differentiation formulas, we were able to obtain NUMOL solutions of good accuracy. We now consider what is in these subroutines, and generally, how we can construct spatial differentiation formulas.

3.1 Polynomial Approximations

We start by considering the variation in a PDE dependent variable $u(x)$ with respect to a spatial independent variable x as (although we are considering here only one independent variable, the results can be applied directly to the case of two or more spatial independent variables, and are

therefore applicable to multidimensional PDEs),

$$u(x) + a_0 + a_1(x - x_i) + a_2(x - x_i)^2 + a_3(x - x_i)^3 + \dots \quad (3.1)$$

where x_i is a value of x to be specified and $a_0, a_1, a_2, a_3, \dots$ are constants to be determined. In other words, we are assuming that u varies as a polynomial in x .

To determine the first constant, we let $x = x_i$ and immediately get $a_0 = u(x_i)$. Next, we differentiate equation (3.1) with respect to x ,

$$du(x)/dx = a_1 + 2a_2(x - x_i) + 3a_3(x - x_i)^2 + \dots$$

from which, with $x = x_i$, we get the second constant $a_1 = du(x_i)/dt$. Successive differentiations followed by $x = x_i$ give $a_2 = (1/2!)d^2u(x_i)/dx^2$, $a_3 = (1/3!)d^3u(x_i)/dx^3$, \dots or in general [with $0! = 1$, $d^0u(x_i)/dx^0 = u(x_i)$]:

$$a_n = (1/n!)d^n u(x_i)/dx^n \quad (3.2)$$

Equations (3.1) and (3.2) (with $n \rightarrow \infty$) are the well-known *Taylor series* (which is the mathematical basis for many of the well-established approximations in numerical analysis). We now consider the use of the Taylor series in the derivation of spatial differentiation formulas.

3.2 Second-Order Formulas for First Derivatives

We first derive an algebraic formula for the approximation of a first-order derivative, $du(x_i)/dx (= u_x(x_i))$, using the values $u(x_{i+1})$, $u(x_i)$ and $u(x_{i-1})$ (defined along a grid in x as we discussed in Chapters 1 and 2). We can consider a Taylor series for the value $u(x_{i+1})$:

$$\begin{aligned} u(x_{i+1}) &= u(x_i) + (du(x_i)/dx)(x_{i+1} - x_i) \\ &\quad + (1/2!)(d^2u(x_i)/dx^2)(x_{i+1} - x_i)^2 + \dots \\ &= u(x_i) + (du(x_i)/dx)(\Delta x) \\ &\quad + (1/2!)(d^2u(x_i)/dx^2)(\Delta x)^2 + \dots \end{aligned} \quad (3.3)$$

where $\Delta x = x_{i+1} - x_i$. Similarly, for $u(x_{i-1})$,

$$\begin{aligned} u(x_{i-1}) &= u(x_i) + (du(x_i)/dx)(-\Delta x) \\ &\quad + (1/2!)(d^2u(x_i)/dx^2)(-\Delta x)^2 + \dots \end{aligned} \quad (3.4)$$

with $-\Delta x = x_{i-1} - x_i$. Note that the derivative of interest is $du(x_i)/dx$, the second right-hand side (RHS) term of equations (3.3) and (3.4). Therefore,

if we subtract equation (3.4) from (3.3) (and include the third derivative terms), we obtain

$$u(x_{i+1}) - u(x_{i-1}) = (2 \Delta x) du(x_i)/dx + (2/3!)(d^3 u(x_i)/dx^3)(\Delta x^3) + \dots$$

or

$$\frac{du(x_i)/dx}{2 \Delta x} = \frac{u(x_{i+1}) - u(x_{i-1})}{2 \Delta x} + O(\Delta x^2) \quad (3.5)$$

Equation (3.5) gives the well-known *second-order central finite difference* approximation for the first derivative $du(x_i)/dx$ (you should verify the origin of the second-order error term, $O(\Delta x^2)$, which follows from the third derivative term). This approximation is termed “central” because the dependent variable is evaluated at points x_{i+1} and x_{i-1} , which are located symmetrically (or centrally) around the point at which the first-order derivative is evaluated, x_i .

Thus, equation (3.5) can be used to calculate a first-order spatial derivative in a NUMOL solution from the two values of the dependent variable, $u(x_{i+1})$ and $u(x_{i-1})$, and in fact, is one of the differentiation formulas used in subroutine DSS002. In other words, equation (3.5) can be applied over the spatial grid at points $i = 2, 3, \dots, N-1$. However, a problem occurs at the endpoints, $i = 1$ and N ; for $i = 1$, $u(x_{i-1}) = u(x_0)$ is required in equation (3.5), which does not exist (it is the dependent variable at a so-called fictitious point); similarly, at $i = N$, equation (3.5) requires $u(x_{N+1})$, which is also nonexistent.

Standard numerical procedures are available involving the use of the boundary conditions (at $i = 1$ and N) to mathematically eliminate the values of the dependent variable at the fictitious points. However, these procedures are rather problem-specific (they generally require a mathematical formulation for each new PDE problem) and are therefore difficult to put into general-purpose programs (such as DSS002). Rather, we would like to have programming that will eliminate the use of fictitious points and make the implementation of boundary conditions straightforward and robust (in the sense that the same procedure can, in principle, be used to handle all types of boundary conditions—Dirichlet, Neumann, and the third type).

To avoid fictitious points, we will develop an approximation for $du(x_1)/dx$ that requires the use of the points $u(x_1)$, $u(x_2)$, and $u(x_3)$ [rather than $u(x_0)$ and $u(x_2)$ required by equation (3.5)]. Thus, we need two Taylor series for $u(x_2)$ and $u(x_3)$:

$$\begin{aligned} u(x_2) &= u(x_1) + (du(x_1)/dx)(\Delta x) \\ &\quad + (1/2!)(d^2 u(x_1)/dx^2)(\Delta x)^2 + \dots \end{aligned} \quad (3.6)$$

$$\begin{aligned} u(x_3) &= u(x_1) + (du(x_1)/dx)(2 \Delta x) \\ &\quad + (1/2!)(d^2u(x_1)/dx^2)(2 \Delta x)^2 + \dots \end{aligned} \quad (3.7)$$

We are interested in an approximation for $du(x_1)/dx$ (the first derivative at the left boundary). Also, we can achieve *maximum accuracy in the approximation by dropping out as many of the higher-order terms* [in equations (3.6) and (3.7)] *as possible*. Thus, if we multiply equation (3.6) by 4, and subtract equation (3.7), we can drop out the second derivative term, to obtain

$$\begin{aligned} 4u(x_2) - 3u(x_1) - u(x_3) &= (2 \Delta x) du(x_1)/dx \\ &\quad - (4/3!)(d^3u(x_1)/dx^3)(\Delta x^3) + \dots \end{aligned}$$

or

$$du(x_1)/dx = \frac{-3u(x_1) + 4u(x_2) - u(x_3)}{2 \Delta x} + O(\Delta x^2) \quad (3.8)$$

Similarly, if we write Taylor series for $u(x_{N-1})$ and $u(x_{N-2})$, and perform the same operations as in deriving equation (3.8), we arrive at an approximation for the first-order derivative at $i = N$:

$$du(x_N)/dx = \frac{3u(x_N) - 4u(x_{N-1}) + u(x_{N-2})}{2 \Delta x} + O(\Delta x^2) \quad (3.9)$$

Now, we have approximations for all of the spatial grid points $i = 1, 2, \dots, N$, equations (3.5), (3.8), and (3.9). In fact, these are the equations used in subroutine DSS002. However, before we go on to the coding in subroutine DSS002, we write equations (3.5), (3.8), and (3.9) in an alternate, more compact, form:

$$d\bar{u}/dx = [1/(2 \Delta x)] \begin{bmatrix} -3 & 4 & -1 \\ -1 & 0 & 1 \\ 1 & -4 & 3 \end{bmatrix} \bar{u} + O(\Delta x^2) \quad (3.10)$$

Equation (3.10) indicates how the derivative vector $d\bar{u}/dx$ is computed from the dependent variable vector \bar{u} . The 3×3 matrix is termed a *differentiation matrix* (or a *computational stencil*); note that the elements of the matrix are just the *weighting coefficients* of equations (3.5), (3.8), and (3.9). Additionally, equation (3.10) can be summarized by specifying the order of the derivative computed (1), the order of the approximation (2), the coefficient multiplying the matrix ($\frac{1}{2}$), and the elements of the matrix. Then, as we develop equations like (3.10), we can summarize them in a table by specifying the preceding information; in fact, extensive tables have been reported by Fornberg (3.1), as well as a concise algorithm for

computing the entries in a table of differentiation matrices. We could even summarize equation (3.10) as (1, 2, $\frac{1}{2}$, 3×3 , us) designating: the order of the derivative (1), the order of the approximation (2), the multiplying constant ($\frac{1}{2}$), the dimensions of the differentiation matrix (3×3), and the type of spacing (uniform spacing with an interval Δx between grid points).

The 3×3 differentiation matrix has several interesting and important properties:

(1) The coefficients in any row sum to zero; this property is required in order for the matrix to differentiate a constant to zero (if it did not differentiate a constant to zero, the differentiator would not be worth much; you should confirm that you understand this point).

(2) The elements of the matrix have an antisymmetric property in the sense that a line drawn through the center element, 0 in this case, connects elements of the same magnitude, but opposite sign. For example, a vertical line through the center connects the elements 4 and -4 . This property is useful in checking the elements of the matrix (to detect errors in programming, for example). It occurs only for odd-order derivatives, a first-order derivative in this case. For even-order derivatives, the differentiation matrix has a symmetry property (lines drawn through the center element connect elements of the same magnitude and sign); this property will be apparent in some examples of differentiation matrices for second-order derivatives in this chapter.

(3) The multiplying constant, $\frac{1}{2}$, also includes the grid spacing, Δx , in the denominator. The need for this is apparent just from the difference in the units between \bar{u} and $d\bar{u}/dx$. This idea can then be extended to higher-order derivatives, i.e., an approximation for a second-order derivative would require multiplication of the differentiation matrix by $1/\Delta x^2$, etc.

Now, we can go to subroutine DSS002 (listed in Program 3.1), which is just an implementation of equations (3.5), (3.8), and (3.9) [or just equation (3.10)].

Program 3.1 Subroutine DSS002

```
SUBROUTINE DSS002(XL,XU,N,U,UX)
C... SUBROUTINE DSS002 COMPUTES THE FIRST DERIVATIVE, U_x, OF A
C... VARIABLE U OVER THE SPATIAL DOMAIN XL LE X LE XU
C... ARGUMENT LIST
C...     XL      LOWER BOUNDARY VALUE OF X (INPUT)
```

continues

```

C...
C...      XU      UPPER BOUNDARY VALUE OF X (INPUT)
C...
C...      N      NUMBER OF GRID POINTS IN THE X DOMAIN INCLUDING THE
C...                  BOUNDARY POINTS (INPUT)
C...
C...      U      ONE DIMENSIONAL ARRAY CONTAINING THE VALUES OF U AT
C...                  THE N GRID POINTS FOR WHICH THE DERIVATIVE IS
C...                  TO BE COMPUTED (INPUT)
C...
C...      UX     ONE DIMENSIONAL ARRAY CONTAINING THE NUMERICAL
C...                  VALUES OF THE DERIVATIVES OF U AT THE N GRID POINTS
C...                  (OUTPUT)
C...
C...      SUBROUTINE DSS002 COMPUTES THE FIRST DERIVATIVE, U , OF A
C...                  X
C...      VARIABLE U OVER THE SPATIAL DOMAIN XL LE X LE XU FROM THE
C...      CLASSICAL THREE POINT, SECOND ORDER FINITE DIFFERENCE APPROXI-
C...      TIONS.
C...
C...      THE DIFFERENTIATION MATRIX IS
C...
C...            -3    4    -1
C...
C...            1/2   -1    0    1
C...
C...            1    -4    3
C...
C...      REAL U(N) ,UX(N)
C...
C...      SPATIAL INCREMENT
DX=(XU-XL)/FLOAT(N-1)
R2FDX=1./(2.*DX)
NM1=N-1
C...
C...      EQUATION (3.8)
UX(1)=R2FDX*
1(      -3.    *U(  1)      +4.    *U(  2)      -1.    *U(  3))
C...
C...      EQUATION (3.5)
DO 1 1=2,NM1
UX(I)=R2FDX*
1(      -1.    *U(I-1)      +0.    *U(  I)      +1.    *U(I+1))
1  CONTINUE
C...
C...      EQUATION (3.9)
UX(N)=R2FDX*
1(      1.    *U(N-2)      -4.    *U(N-1)      +3.    *U(  N))
RETURN
END

```

Note that DSS002 is a direct implementation of equation (3.10) [in fact, the coding contains some “do-nothing” operations such as multiplication by 0 and 1 in order to preserve the correspondence with equation (3.10)]. Clearly, DSS002 is not complicated, particularly in view of its utility in solving a series of PDE problems in Chapters 1 and 2. As we observed there, however, it does have limited accuracy, and we should

therefore consider the derivation and coding of more accurate differentiation formulas, as in DSS004.

However, before we proceed to these more accurate formulas, we can consider more precisely what we mean by “accuracy.” For example, we have observed that equations (3.5), (3.8), and (3.9) are exact for a constant function (this is sometimes referred to as *superaccuracy*). We might also consider how accurate these formulas are for a first-order polynomial

$$u(x) = a_0 + a_1 x \quad (3.11)$$

In other words, it is conceivable that the solution to a PDE varies linearly with the spatial variable, x . We substitute equation (3.11) in equation (3.5)

$$\frac{du(x_i)/dx}{2 \Delta x} = \frac{a_0 + a_1(x_i + \Delta x) - (a_0 + a_1(x_i - \Delta x))}{2 \Delta x} + O(\Delta x^2) = a_1$$

which, again, is exactly correct ($d(a_0 + a_1 x)/dx = a_1$). This is not surprising if we consider that in deriving equation (3.5), the second-order derivative was eliminated; in other words, only the third- and higher-order derivatives of the function to be differentiated will contribute to the final result, and since the linear function, equation (3.11), has zero third- and higher-order derivatives, there is no contribution from these higher-order derivatives to $du(x_i)/dx$ computed from equation (3.5). This conclusion illustrates the significance of the order term $O(\Delta x^2)$ in equation (3.5). The same conclusions also apply to equations (3.8) and (3.9) (the verification is left as problems at the end of the chapter).

Therefore, we might try a second-order polynomial

$$u(x) = a_0 + a_1 x + a_2 x^2 \quad (3.12)$$

Substitution in equation (3.5) gives

$$\begin{aligned} & \frac{du(x_i)/dx}{2 \Delta x} \\ &= \frac{a_0 + a_1(x_i + \Delta x) + a_2(x_i + \Delta x)^2 - (a_0 + a_1(x_i - \Delta x) + a_2(x_i - \Delta x)^2)}{2 \Delta x} \\ &= O(\Delta x^2) = a_1 + 2a_2 x \end{aligned}$$

which again is exactly correct [equation (3.5) is superaccurate for zeroth-, first-, and second-order polynomials; the same conclusions apply to equations (3.8) and (3.9)].

So far, the results have been very encouraging, so we might next consider a third-order polynomial:

$$u(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \quad (3.13)$$

Substitution of equation (3.13) in equation (3.5) gives

$$\begin{aligned} du(x_i)/dx &= \frac{a_0 + a_1(x_i + \Delta x) + a_2(x_i + \Delta x)^2 + a_3(x_i + \Delta x)^3}{2 \Delta x} \\ &\quad - \frac{a_0 + a_1(x_i - \Delta x) + a_2(x_i - \Delta x)^2 + a_3(x_i - \Delta x)^3}{2 \Delta x} + O(\Delta x^2) \\ &= a_1 + 2a_2 x + 3a_3 x^2 + a_3 \Delta x^2 \end{aligned}$$

From this result, we come to two important conclusions:

- (1) Equation (3.5) is exact for second- and lower-order polynomials only. It is not exact for third- and higher-order polynomials [which, again, is expected, since in the derivation of equation (3.5), the third derivative term of the Taylor series remained, and this is nonzero for third- and higher-order polynomials].
- (2) The error in the computed derivative (for third- and higher-order polynomials) decreases with Δx (note the preceding error term $a_3 \Delta x^2$). We observed this in the solution of equations (1.21) to (1.24) [even though the solution varied as $\sin(\pi x/L)$ from equation (1.25) rather than as a polynomial].

If equations (3.8) and (3.9) are applied to equation (3.13), we arrive at the same two conclusions, plus one more:

- (3) The error in the numerical derivative is larger at the endpoints [$i = 1$ where equation (3.8) is applied, and $i = N$ where equation (3.9) is applied], than at the interior points [$i = 2, 3, \dots, N - 1$ where equation (3.5) is applied]. In the case of equation (3.13), application of equations (3.8) and (3.9) gives an error of $-2a_3 \Delta x^2$, which, in absolute value, is twice the interior error of $a_3 \Delta x^2$; however, only the multiplying constant changed (-2 rather than 1) and not the power of Δx [all three errors were proportional to Δx^2 , so all of the formulas, equations (3.5), (3.8), and (3.9), are second-order correct].

We now have a somewhat more precise understanding of the accuracy of the numerical differentiation formulas in DSS002. Remember, though, that the preceding discussion is based on polynomials, and in general, the solution to a PDE will not vary spatially as a polynomial, so we cannot assume that all of the preceding conclusions apply in general. In particular, we hope that the NUMOL solution to a PDE will improve with decreasing Δx (as we use more grid points). This proved to be the case in the solution of equations (1.21) to (1.24), and experience has demonstrated that in general, this will be true; of course, the price we pay for improved

accuracy at small Δx is more computation (more grid points lead to more ODEs to be integrated).

Now, we are ready to consider another approach to improving the NUMOL solution to a PDE (in addition to using a smaller Δx or finer grid): the use of higher-order differentiation formulas.

3.3 Fourth-Order Formulas for First Derivatives

The three-point differentiation formulas in DSS002 are second-order correct, which, in other words, is a specification of their limited accuracy. We could therefore naturally consider how we might increase the order of accuracy; intuitively, it would seem that this could be done by using more points in the differentiation formulas, e.g., five points instead of three. If, for example, we are seeking an approximation for a first derivative, u_x , at grid point i , we could use the dependent variable $u(x)$ at x_{i-2} , x_{i-1} , x_i , x_{i+1} , and x_{i+2} in the approximation. This suggests we write a Taylor series for grid points $i - 2$, $i - 1$, $i + 1$, and $i + 2$:

$$\begin{aligned} u(x_{i-2}) &= u(x_i) + (du(x_i)/dx)(-2 \Delta x) \\ &\quad + (1/2!)(d^2u(x_i)/dx^2)(-2 \Delta x)^2 + \dots \end{aligned} \quad (3.14)$$

$$\begin{aligned} u(x_{i-1}) &= u(x_i) + (du(x_i)/dx)(-\Delta x) \\ &\quad + (1/2!)(d^2u(x_i)/dx^2)(-\Delta x)^2 + \dots \end{aligned} \quad (3.15)$$

$$\begin{aligned} u(x_{i+1}) &= u(x_i) + (du(x_i)/dx)(\Delta x) \\ &\quad + (1/2!)(d^2u(x_i)/dx^2)(\Delta x)^2 + \dots \end{aligned} \quad (3.16)$$

$$\begin{aligned} u(x_{i+2}) &= u(x_i) + (du(x_i)/dx)(2 \Delta x) \\ &\quad + (1/2!)(d^2u(x_i)/dx^2)(2 \Delta x)^2 + \dots \end{aligned} \quad (3.17)$$

Now, proceeding as we did with the three-point approximations, we will take a linear combination of equations (3.14) to (3.17) so as to keep the derivative of interest, $du(x_i)/dx$, and drop as many of the higher derivative terms as possible (to give maximum accuracy in the final differentiation formula). In the case of equations (3.3) and (3.4), this was done by a single subtraction. However, in the case of equations (3.14) to (3.17), it is not so clear how to proceed; we need a systematic procedure rather than just relying on insight.

If we multiply equation (3.14) by a constant a , equation (3.15) by a

constant b , etc.,

$$\begin{aligned} au(x_{i-2}) &= au(x_i) + a(du(x_i)/dx)(-2 \Delta x) \\ &\quad + a(1/2!)(d^2u(x_i)/dx^2)(-2 \Delta x)^2 + \dots \end{aligned} \quad (3.18)$$

$$\begin{aligned} bu(x_{i-1}) &= bu(x_i) + b(du(x_i)/dx)(-\Delta x) \\ &\quad + b(1/2!)(d^2u(x_i)/dx^2)(-\Delta x)^2 + \dots \end{aligned} \quad (3.19)$$

$$\begin{aligned} cu(x_{i+1}) &= cu(x_i) + c(du(x_i)/dx)(\Delta x) \\ &\quad + c(1/2!)(d^2u(x_i)/dx^2)(\Delta x)^2 + \dots \end{aligned} \quad (3.20)$$

$$\begin{aligned} du(x_{i+2}) &= du(x_i) + d(du(x_i)/dx)(2 \Delta x) \\ &\quad + d(1/2!)(d^2u(x_i)/dx^2)(2 \Delta x)^2 + \dots \end{aligned} \quad (3.21)$$

then sum the resulting equations, we can retain the first derivative term, $du(x_i)/dx$, by imposing the condition

$$-2a - b + c + 2d = 1 \quad (3.22)$$

The 1 in the RHS of equation (3.22) ensures that when equations (3.18) to (3.21) are added with the numerical values of a , b , c , and d (still to be determined), the first derivative term will remain. Similarly, in order to eliminate the second derivative term, $d^2u(x_i)/dx^2$, we can impose the condition

$$4a + b + c + 4d = 0 \quad (3.23)$$

The 0 in the RHS of equation (3.23) ensures that when equations (3.18) to (3.21) are added with the numerical values of a , b , c , and d , the second derivative term will be eliminated.

In the same way, to eliminate the third- and fourth-order derivative terms, we can impose the conditions

$$-8a - b + c + 8d = 0 \quad (3.24)$$

$$16a + b + c + 16d = 0 \quad (3.25)$$

We now have four linear algebraic equations [(3.22) to (3.25)] for the four constants, a to d . Simultaneous solution gives $a = 2/4!$, $b = -16/4!$, $c = 16/4!$, $d = -2/4!$. Then, if equations (3.18) to (3.21) are added with these numerical values for a to d , we obtain for the derivative of interest $du(x_i)/dx$:

$$\begin{aligned} du(x_i)/dx &= (1/(4! \Delta x))(2u(x_{i-2}) - 16u(x_{i-1}) + 0u(x_i) + 16u(x_{i+1}) \\ &\quad - 2u(x_{i+2})) + O(\Delta x^4) \end{aligned} \quad (3.26)$$

Note that equation (3.26) is fourth-order correct. Also, the weighting coefficients, 2, -16, 16, and -2 sum to zero, again, as required for equation (3.26) to correctly differentiate a constant to zero (this check, which can be made by inspection, is quite useful when deriving a new differentiation formula). Based on the analysis of equation (3.5) applied to polynomials of various orders, what conclusions can you draw about equation (3.26) applied to polynomials of various orders?

Equation (3.26) is a central difference approximation since values of $u(x)$ located symmetrically around $u(x_i)$ are used to calculate $du(x_i)/dx$. As with the corresponding three-point approximation, equation (3.5), equation (3.26) cannot be applied at grid points $i = 1$ or 2 [because $u(x_{-1})$ and $u(x_0)$ are required, which are fictitious]; similarly, equation (3.26) cannot be applied at $i = N - 1$ and N [since $u(x_{N+1})$ and $u(x_{N+2})$ are required]. Therefore, we must use four Taylor series that involve only interior values of $u(x)$. For example, to obtain an approximation for $du(x_1)/dx$, we use the Taylor series for $u(x_2)$, $u(x_3)$, $u(x_4)$, and $u(x_5)$

$$\begin{aligned} au(x_2) &= au(x_1) + a(du(x_1)/dx)(\Delta x) \\ &\quad + a(1/2!)(d^2u(x_1)/(dx^2))(\Delta x)^2 + \dots \end{aligned} \quad (3.27)$$

$$\begin{aligned} bu(x_3) &= bu(x_1) + b(du(x_1)/dx)(2 \Delta x) \\ &\quad + b(1/2!)(d^2u(x_1)/(dx^2))(2 \Delta x)^2 + \dots \end{aligned} \quad (3.28)$$

$$\begin{aligned} cu(x_4) &= cu(x_1) + c(du(x_1)/(dx))(3 \Delta x) \\ &\quad + c(1/2!)(d^2u(x_1)/(dx^2))(3 \Delta x)^2 + \dots \end{aligned} \quad (3.29)$$

$$\begin{aligned} du(x_5) &= du(x_1) + d(du(x_1)/dx)(4 \Delta x) \\ &\quad + d(1/2!)(d^2u(x_1)/(dx^2))(4 \Delta x)^2 + \dots \end{aligned} \quad (3.30)$$

To retain the first derivative, $du(x_1)/dx$, we impose the condition

$$a + 2b + 3c + 4d = 1 \quad (3.31)$$

To delete the second, third, and fourth derivatives, we impose the conditions

$$a + 4b + 9c + 16d = 0 \quad (3.32)$$

$$a + 8b + 27c + 64d = 0 \quad (3.33)$$

$$a + 16b + 81c + 256d = 0 \quad (3.34)$$

Simultaneous solution of equations (3.31) to (3.34) gives $a = 96/4!$, $b = -72/4!$, $c = 32/4!$, and $d = -6/4!$. Substitution of these values into equations (3.31) to (3.34), followed by the addition of these equations and

the algebraic solution for $du(x_1)/dx$, gives

$$\begin{aligned} du(x_1)/dx = & (1/(4! \Delta x))(-50u(x_1) + 96u(x_2) - 72u(x_3) + 32u(x_4) \\ & - 6u(x_5)) + O(\Delta x^4) \end{aligned} \quad (3.35)$$

Again, the weighting coefficients sum to zero.

Similarly, for an approximation of $du(x_2)/dx$, we use four Taylor series for $u(x_1)$, $u(x_3)$, $u(x_4)$, and $u(x_5)$. The corresponding algebraic equations are

$$-a + b + 2c + 3d = 1 \quad (3.36)$$

$$a + b + 4c + 9d = 0 \quad (3.37)$$

$$-a + b + 8c + 27d = 0 \quad (3.38)$$

$$a + b + 16c + 81d = 0 \quad (3.39)$$

Simultaneous solution of these equations gives $a = -6/4!$, $b = 36/4!$, $c = -12/4!$, and $d = 2/4!$. The corresponding differentiation formula is

$$\begin{aligned} du(x_2)/dx = & (1/(4! \Delta x))(-6u(x_1) - 20u(x_2) + 36u(x_3) - 12u(x_4) \\ & + 2u(x_5)) + O(\Delta x^4) \end{aligned} \quad (3.40)$$

Similar derivations for approximations of $du(x_{N-1})/dx$ and $du(x_N)/dx$ give

$$\begin{aligned} du(x_{N-1})/dx = & (1/(4! \Delta x))(-2u(x_{N-4}) + 12u(x_{N-3}) - 36u(x_{N-2}) \\ & + 20u(x_{N-1}) + 6u(x_N)) + O(\Delta x^4) \end{aligned} \quad (3.41)$$

$$\begin{aligned} du(x_N)/dx = & (1/(4! \Delta x))(6u(x_{N-4}) - 32u(x_{N-3}) + 72u(x_{N-2}) \\ & - 96u(x_{N-1}) + 50u(x_N)) + O(\Delta x^4) \end{aligned} \quad (3.42)$$

We again summarize equations (3.26), (3.35), (3.40), (3.41), and (3.42) in terms of a differentiation matrix

$$d\bar{u}/dx = (1/(4! \Delta x)) \begin{bmatrix} -50 & 96 & -72 & 32 & -6 \\ -6 & -20 & 36 & -12 & 2 \\ 2 & -16 & 0 & 16 & -2 \\ -2 & 12 & -36 & 20 & 6 \\ 6 & -32 & 72 & -96 & 50 \end{bmatrix} \bar{u} + O(\Delta x^4) \quad (3.43)$$

Note again, as in the case of equation (3.10), that the differentiation matrix is antisymmetric, and the elements sum to zero in each row.

Equation (3.43) is implemented in subroutine DSS004, listed in Program 3.2.

Program 3.2 Subroutine DSS004

```

SUBROUTINE DSS004(XL,XU,N,U,UX)
C... SUBROUTINE DSS004 COMPUTES THE FIRST DERIVATIVE, U , OF A
C... VARIABLE U OVER THE SPATIAL DOMAIN XL LE X LE XU FROM CLASSICAL
C... FIVE POINT, FOURTH ORDER FINITE DIFFERENCE APPROXIMATIONS
C... ARGUMENT LIST
C...
C...     XL      LOWER BOUNDARY VALUE OF X (INPUT)
C...     XU      UPPER BOUNDARY VALUE OF X (INPUT)
C...     N       NUMBER OF GRID POINTS IN THE X DOMAIN INCLUDING THE
C...             BOUNDARY POINTS (INPUT)
C...
C...     U       ONE-DIMENSIONAL ARRAY CONTAINING THE VALUES OF U AT
C...             THE N GRID POINT POINTS FOR WHICH THE DERIVATIVE IS
C...             TO BE COMPUTED (INPUT)
C...
C...     UX      ONE-DIMENSIONAL ARRAY CONTAINING THE NUMERICAL
C...             VALUES OF THE DERIVATIVES OF U AT THE N GRID POINTS
C...             (OUTPUT)
C...
C... THE DIFFERENTIATION MATRIX IS
C...
C...          -50    96   -72    32    -6
C...          -6   -20    36   -12     2
C...
C...          1/24    2   -16     0    16   -2
C...
C...          -2    12   -36    20     6
C...
C...          6   -32    72   -96    50
C...
REAL U(N) ,UX(N)
C...
C... COMPUTE THE SPATIAL INCREMENT
DX=(XU-XL)/FLOAT(N-1)
R4FDX=1./(24.*DX)
NM2=N-2
C...
C... EQUATION (3.35)
UX(-1)=R4FDX*
1(-50.*U(-1) +96.*U(-2) -72.*U(-3) +32.*U(-4) -6.*U(-5))
C...
C... EQUATION (3.40)
UX( 2)=R4FDX*
1(-6.*U( 1) -20.*U( 2) +36.*U( 3) -12.*U( 4) +2.*U( 5))
C...
C... EQUATION (3.26)
DO 1 I=3,NM2
UX( I)=R4FDX*
1(+2.*U(I-2) -16.*U(I-1) +0.*U( I) +16.*U(I+1) -2.*U(I+2))
1 CONTINUE
C...
C... EQUATION (3.41)
UX(N-1)=R4FDX*
1(-2.*U(N-4) +12.*U(N-3) -36.*U(N-2) +20.*U(N-1) +6.*U( N))

```

continues

```

C...
C... EQUATION (3.42)
UX( N)=R4FDX*
1( 6.*U(N-4) -32.*U(N-3) +72.*U(N-2) -96.*U(N-1) +50.*U( N))
RETURN
END

```

Clearly, we can continue this process of using additional terms to derive higher-order differentiation formulas. In fact, this has been done, as summarized below:

- DSS006 Seven-point, sixth-order formulas
- DSS008 Nine-point, eighth-order formulas
- DSS010 Eleven-point, tenth-order formulas

We can naturally ask Why not use the 11-point formulas in all NUMOL applications? This would be logical if we could be assured that the spatial variation of the PDE will always be a polynomial, but in general, this will not be the case, e.g., $\sin(\pi x/L)$ in equation (1.25). Also, all of the preceding approximations for the derivative $d\bar{u}/dx$ are based on polynomials, i.e., truncated Taylor series (recall Section 3.1). However, polynomials of increasing order have derivatives with an increasing number of roots. For example, the fourth-order polynomial, which is the basis of equation (3.43), differentiates once to a third-order polynomial that has three roots; at each of these three roots, the polynomial will therefore have a maximum or a minimum, suggesting that it can oscillate between three maximum and minimum values. Similarly, a tenth-order polynomial will have nine maxima and minima, and it will in general oscillate between these nine values. In other words, as the order of the approximating polynomial increases, the possibility of unrealistic oscillation in the NUMOL solution of a PDE also increases, and this is frequently observed. We have therefore concluded, as a result of experience, that the fourth-order formulas of equation (3.43) are a good compromise between accuracy (defined, in this case, as fourth-order) and the minimization of oscillation. Admittedly, this is a rather loose statement. Certainly, other approximations can be used that might be better behaved (not have the oscillation of polynomials).

In fact, essentially any approximation can be considered for the PDE spatial derivatives, and some approximations will generally be found to be better than others. Thus, the NUMOL is really open-ended, and can be implemented in many ways. Commonly used approximations for PDE spatial derivatives include splines, finite elements, and weighted residual methods. We could obviously devote much more discussion to the development of spatial derivative approximations, but in order to keep the discussion to reasonable length, we shall consider a few other selected polynomial approximations that have been useful in the solution of a range

of PDE problems. We do not wish to suggest, however, that the following treatment is at all comprehensive, and we encourage you to try other approaches, particularly if they are readily implemented, e.g., using a subroutine for spline approximations from which derivatives are easily computed; also, NUMOL software is available that could possibly be used immediately after reading the documentation, for example, in commercial libraries such as IMSL and NAG, and in the open literature [e.g., Madsen and Sincovec (3.2), (3.3)].

3.4 Fourth-Order Formulas for Second Derivatives

Several examples in Chapters 1 and 2 involved PDEs with second-order spatial derivatives, e.g., Fourier's second law, equation (1.4); also, PDEs with fourth-order spatial derivatives are not uncommon. Depending on the application, PDEs might contain combinations of first-, second-, and higher-order spatial derivatives in both linear and nonlinear terms. Therefore, we should develop methods that will accommodate all of these possibilities, and in fact, one of the strengths of the NUMOL is its flexibility in the ability to accommodate virtually any combination of spatial derivatives with a straightforward application of some basic programming constructs.

For example, if we are interested in approximating a second spatial (parabolic) derivative in a PDE, one approach is to compute the derivative of the first-order derivative as we did in previous examples, with *stagewise differentiation*. Our experience has generally been good with this approach, although some difficulties may arise when the PDE also has first-order spatial (hyperbolic) derivatives. In this section, we consider the alternative of computing the second-order derivative directly from the dependent variable (rather than via the first derivative). This approach has the advantages of

- (1) Greater robustness (it will handle a broader class of problems than stagewise differentiation).
- (2) All three types of boundary conditions (Dirichlet, Neumann, and the third type) can be included without approximation (the boundary conditions are not approximated).

The disadvantages are

- (1) The second-order derivative cannot include a nonlinear term within the derivative, e.g., $(k(u)u_x)_x$.

- (2) The implementation of the NUMOL on two- and three-dimensional domains is somewhat more complicated than when using stagewise differentiation (as in Section 2.7, for example).

We shall essentially follow the approach in the preceding Section 3.3 to derive fourth-order correct approximations, e.g., for the interior grid points i , $i = 3, 4, \dots, N - 2$, we use a Taylor series for the dependent variable $u(x)$ at x_{i-2} , x_{i-1} , x_{i+1} , and x_{i+2} in the approximation

$$\begin{aligned} au(x_{i-2}) &= au(x_i) + a(du(x_i)/dx)(-2 \Delta x) \\ &\quad + a(1/2!)(d^2u(x_i)/dx^2)(-2 \Delta x)^2 + \dots \end{aligned} \quad (3.44)$$

$$\begin{aligned} bu(x_{i-1}) &= bu(x_i) + b(du(x_i)/dx)(-\Delta x) \\ &\quad + b(1/2!)(d^2u(x_i)/dx^2)(-\Delta x)^2 + \dots \end{aligned} \quad (3.45)$$

$$\begin{aligned} cu(x_{i+1}) &= cu(x_i) + c(du(x_i)/dx)(\Delta x) \\ &\quad + c(1/2!)(d^2u(x_i)/dx^2)(\Delta x)^2 + \dots \end{aligned} \quad (3.46)$$

$$\begin{aligned} du(x_{i+2}) &= du(x_i) + d(du(x_i)/dx)(2 \Delta x) \\ &\quad + d(1/2!)(d^2u(x_i)/dx^2)(2 \Delta x)^2 + \dots \end{aligned} \quad (3.47)$$

We retain the second derivative, $d^2u(x_i)/dx^2$, through the algebraic equation

$$4a + b + c + 4d = 2 \quad (3.48)$$

[the 2 in equation (3.48) will conveniently cancel the $2!$ in equations (3.44) to (3.47)].

Also, the third, fourth, and fifth derivatives are dropped through the three algebraic equations

$$-8a - b + c + 8d = 0 \quad (3.49)$$

$$16a + b + c + 16d = 0 \quad (3.50)$$

$$-32a - b + c + 32d = 0 \quad (3.51)$$

We now have four linear algebraic equations [(3.48) to (3.51)] for the four constants, a to d . Simultaneous solution gives $a = -2/4!$, $b = 32/4!$, $c = 32/4!$, and $d = -2/4!$. Then, if equations (3.44) to (3.47) are added with these numerical values for a to d , we obtain for the derivative of interest $du^2(x_i)/dx^2$:

$$\begin{aligned} du^2(x_i)/dx^2 &= (1/(4! \Delta x^2))(-2u(x_{i-2}) + 32u(x_{i-1}) - 60u(x_i) \\ &\quad + 32u(x_{i+1}) - 2u(x_{i+2})) + O(\Delta x^4) \end{aligned} \quad (3.52)$$

Note the fortuitous cancellation of the first derivative terms since

$$-2a - b + c + 2d = -2(-2) - (32) + (32) + 2(-2) = 0 \quad (3.53)$$

For an approximation of $d^2u(x_2)/dx^2$, we take a linear combination of the Taylor series for $u(x)$ at $x = x_1, x_3, x_4, x_5$, and x_6

$$au(x_1) + bu(x_3) + cu(x_4) + du(x_5) + eu(x_6)$$

To drop $du(x_2)/dx$,

$$-a + b + 2c + 3d + 4e = 0 \quad (3.54)$$

Similarly, to drop $d^3u(x_2)/dx^3$, $d^4u(x_2)/dx^4$ and $d^5u(x_2)/dx^5$

$$-a + b + 8c + 27d + 64e = 0 \quad (3.55)$$

$$a + b + 16c + 81d + 256e = 0 \quad (3.56)$$

$$-a + b + 32c + 243d + 1024e = 0 \quad (3.57)$$

Finally, to retain $d^2u(x_2)/dx^2$

$$a + b + 4c + 9d + 16e = 2 \quad (3.58)$$

Solution of equations (3.54) to (3.58) gives $a = 20/4!$, $b = -8/4!$, $c = 28/4!$, $d = -12/4!$, and $e = 2/4!$. Substitution of these values in the five Taylor series, followed by solution for $d^2u(x_2)/dx^2$, gives

$$\begin{aligned} du^2(x_2)/dx^2 &= (1/(4! \Delta x^2))(20u(x_1) - 30u(x_2) - 8u(x_3) \\ &\quad + 28u(x_4) - 12u(x_5) + 2u(x_6)) + O(\Delta x^4) \end{aligned} \quad (3.59)$$

Similarly, to obtain an approximation for $du^2(x_{N-1})/dx^2$, we take a linear combination of the Taylor series for $u(x)$ at $x_{N-5}, x_{N-4}, x_{N-3}, x_{N-2}$, and x_N . The final result is

$$\begin{aligned} du^2(x_{N-1})/dx^2 &= (1/(4! \Delta x^2))(20u(x_N) - 30u(x_{N-1}) - 8u(x_{N-2}) \\ &\quad + 28u(x_{N-3}) - 12u(x_{N-4}) \\ &\quad + 2u(x_{N-5})) + O(\Delta x^4) \end{aligned} \quad (3.60)$$

Finally, we need approximations for $d^2u(x_1)/dx^2$ and $d^2u(x_N)/dx^2$; as indicated previously, this will be done so as to include boundary conditions at $x = x_1$ and x_N . For $du^2(x_1)/dx^2$ with $du(x_1)/dx$ included as a boundary condition, we take a linear combination of $u(x)$ at $x = x_2, x_3, x_4$, and x_5 , plus $du(x)/dx$ at $x = x_1$

$$au(x_2) + bu(x_3) + cu(x_4) + du(x_5) + e du(x_1)/dx$$

To drop $du(x_1)/dx$,

$$a + 2b + 3c + 4d + e = 0 \quad (3.61)$$

Similarly, to drop $d^3u(x_1)/dx^3$, $d^4u(x_1)/dx^4$, and $d^5u(x_1)/dx^5$

$$a + 8b + 27c + 64d = 0 \quad (3.62)$$

$$a + 16b + 81c + 256d = 0 \quad (3.63)$$

$$a + 32b + 243c + 1024d = 0 \quad (3.64)$$

Finally, to retain $d^2u(x_1)/dx^2$

$$a + 4b + 9c + 16d = 2 \quad (3.65)$$

Solution of equations (3.61) to (3.65) gives $a = 192/4!$, $b = -72/4!$, $c = 64/(3(4!))$, $d = -3/4!$, and $e = -100/4!$. Substitution of these values in the five Taylor series, followed by solution for $d^2u(x_1)/dx^2$, gives

$$\begin{aligned} du^2(x_1)/dx^2 &= (1/(4! \Delta x^2))((-415/3)u(x_1) + 192u(x_2) - 72u(x_3) \\ &\quad + (64/3)u(x_4) - 3u(x_5) - 100(du(x_1)/dx) \Delta x) \\ &\quad + O(\Delta x^4) \end{aligned} \quad (3.66)$$

Similarly, for $du^2(x_N)/dx^2$ with $du(x_N)/dx$ as a boundary condition, we take a linear combination of $u(x)$ at $x = x_{N-1}$, x_{N-2} , x_{N-3} , and x_{N-4} , plus $du(x)/dx$ at $x = x_N$

$$au(x_{N-1}) + bu(x_{N-2}) + cu(x_{N-3}) + du(x_{N-4}) + e du(x_N)/dx$$

Applying the conditions to drop $du(x_N)/dx$, $d^3u(x_N)/dx^3$, $d^4u(x_N)/dx^4$, and $d^5u(x_N)/dx^5$, while retaining $du^2(x_N)/dx^2$, we arrive finally at

$$\begin{aligned} du^2(x_N)/dx^2 &= (1/(4! \Delta x^2))((-415/3)u(x_N) \\ &\quad + 192u(x_{N-1}) - 72u(x_{N-2}) + (64/3)u(x_{N-3}) \\ &\quad - 3u(x_{N-4}) + 100(du(x_N)/dx) \Delta x) + O(\Delta x^4) \end{aligned} \quad (3.67)$$

Equations (3.66) and (3.67) can be used to calculate $du^2(x_i)/dx^2$ at grid points $i = 1$ and N with Neumann boundary conditions [since $du(x_i)/dx$ at $i = 1$ and N are also included].

To calculate $du^2(x_i)/dx^2$ at $i = 1$ and N with Dirichlet boundary conditions, we take linear combinations of the Taylor series at five adjacent grid points starting at $i = 1$ or N . Again, dropping the first and third to fifth derivatives, while retaining the second derivative, gives

$$\begin{aligned}
 du^2(x_1)/dx^2 = & (1/(4! \Delta x^2))(90u(x_1) \\
 & - 308u(x_2) + 428u(x_3) - 312u(x_4) \\
 & + 122u(x_5) - 20u(x_6)) + O(\Delta x^4)
 \end{aligned} \tag{3.68}$$

$$\begin{aligned}
 du^2(x_N)/dx^2 = & (1/(4! \Delta x^2))(90u(x_N) \\
 & - 308u(x_{N-1}) + 428u(x_{N-2}) - 312u(x_{N-3}) \\
 & + 122u(x_{N-4}) - 20u(x_{N-5})) + O(\Delta x^4)
 \end{aligned} \tag{3.69}$$

We now have all of the formulas required to calculate a fourth-order approximation of $du^2(x)/dx^2$ with Dirichlet and Neumann boundary conditions. These equations are programmed in subroutine DSS044 listed in Program 3.3. The weighting coefficients in subroutine DSS044 have been reduced by a factor of 2 since the coefficient multiplying the RHS of each equation is $1/12$ rather than $1/4! = 1/24$. Otherwise, the coding is a straightforward implementation of equations (3.52), (3.59), (3.60), (3.66), (3.67), (3.68), and (3.69). Note in particular that arguments NL and NU are inputs to DSS044 that specify either Dirichlet or Neumann boundary conditions at grid points $i = 1$ and N .

Program 3.3 Subroutine DSS044

```

SUBROUTINE DSS044(XL,XU,N,U,UX,UXX,NL,NU)
C...
C... SUBROUTINE DSS044 COMPUTES A FOURTH ORDER APPROXIMATION OF A
C... SECOND ORDER DERIVATIVE, WITH OR WITHOUT THE NORMAL DERIVATIVE
C... AT THE BOUNDARY.
C...
C...      SINGLE PRECISION
C...
C... ARGUMENT LIST
C...
C...      XL      LEFT VALUE OF THE SPATIAL INDEPENDENT VARIABLE (INPUT)
C...
C...      XU      RIGHT VALUE OF THE SPATIAL INDEPENDENT VARIABLE (INPUT)
C...
C...      N       NUMBER OF SPATIAL GRID POINTS, INCLUDING THE END
C...              POINTS (INPUT)
C...
C...      U       ONE-DIMENSIONAL ARRAY OF THE DEPENDENT VARIABLE TO BE
C...              DIFFERENTIATED (INPUT)
C...
C...      UX      ONE-DIMENSIONAL ARRAY OF THE FIRST DERIVATIVE OF U.
C...              THE END VALUES OF UX, UX(1) AND UX(N), ARE USED IN
C...              NEUMANN BOUNDARY CONDITIONS AT X = XL AND X = XU,
C...              DEPENDING ON THE ARGUMENTS NL AND NU (SEE THE DE-
C...              Scription OF NL AND NU BELOW)
C...
C...      UXX     ONE-DIMENSIONAL ARRAY OF THE SECOND DERIVATIVE OF U
C...              (OUTPUT)

```

continues

```

C...
C...      NL      INTEGER INDEX FOR THE TYPE OF BOUNDARY CONDITION AT
C...          X = XL (INPUT).  THE ALLOWABLE VALUES ARE
C...
C...          1 - DIRICHLET BOUNDARY CONDITION AT X = XL
C...              (UX(1) IS NOT USED)
C...
C...          2 - NEUMANN BOUNDARY CONDITION AT X = XL
C...              (UX(1) IS USED)
C...
C...      NU      INTEGER INDEX FOR THE TYPE OF BOUNDARY CONDITION AT
C...          X = XU (INPUT).  THE ALLOWABLE VALUES ARE
C...
C...          1 - DIRICHLET BOUNDARY CONDITION AT X = XU
C...              (UX(N) IS NOT USED)
C...
C...          2 - NEUMANN BOUNDARY CONDITION AT X = XU
C...              (UX(N) IS USED)
C...
C...      REAL U(N), UX(N), UXX(N)
C...
C...      GRID SPACING
C...      DX=(XU-XL)/FLOAT(N-1)
C...
C...      1/(12*DX**2) FOR SUBSEQUENT USE
C...      R12DXS=1./(12.0E0*DX**2)
C...
C...      UXX AT THE LEFT BOUNDARY
C...
C...      WITHOUT UX (EQUATION (3.68))
C...      IF(NL.EQ.1)THEN
C...          UXX(1)=R12DXS*
C...
C...          1      ( 45.0E0*U(1)
C...          2      -154.0E0*U(2)
C...          3      +214.0E0*U(3)
C...          4      -156.0E0*U(4)
C...          5      +61.0E0*U(5)
C...          6      -10.0E0*U(6) )
C...
C...      WITH UX (EQUATION (3.66))
C...      ELSE IF(NL.EQ.2)THEN
C...          UXX(1)=R12DXS*
C...
C...          1      (-415.0E0/6.0E0*U(1)
C...          2      +96.0E0*U(2)
C...          3      -36.0E0*U(3)
C...          4      +32.0E0/3.0E0*U(4)
C...          5      -3.0E0/2.0E0*U(5)
C...          6      -50.0E0*UX(1)*DX)
C...
C...      END IF
C...
C...      UXX AT THE RIGHT BOUNDARY
C...
C...      WITHOUT UX (EQUATION (3.69))
C...      IF(NU.EQ.1)THEN
C...          UXX(N)=R12DXS*
C...
C...          1      ( 45.0E0*U(N  )
C...          2      -154.0E0*U(N-1)
C...          3      +214.0E0*U(N-2)
C...          4      -156.0E0*U(N-3)
C...          5      +61.0E0*U(N-4)
C...          6      -10.0E0*U(N-5) )

```

continues

```

C...
C...      WITH UX (EQUATION (3.67))
C...      ELSE IF(NU.EQ.2)THEN
UXX(N)=R12DXS*
1          (-415.OEO/6.OEO*U(N   ))
2          +96.OEO*U(N-1)
3          -36.OEO*U(N-2)
4          +32.OEO/3.OEO*U(N-3)
5          -3.OEO/2.OEO*U(N-4)
6          +50.OEO*UX(N   )*DX)
END IF
C...
C...      UX AT THE INTERIOR GRID POINTS
C...
C...      I = 2 (EQUATION (3.59))
UXX(2)=R12DXS*
1          ( 10.OEO*U(1)
2          -15.OEO*U(2)
3          -4.OEO*U(3)
4          +14.OEO*U(4)
5          -6.OEO*U(5)
6          +1.OEO*U(6))
C...
C...      I = N-1 (EQUATION (3.60))
UXX(N-1)=R12DXS*
1          ( 10.OEO*U(N   )
2          -15.OEO*U(N-1)
3          -4.OEO*U(N-2)
4          +14.OEO*U(N-3)
5          -6.OEO*U(N-4)
6          +1.OEO*U(N-5))
C...
C...      I = 3, 4, . . . , N-2 (EQUATION (3.52))
DO 1 I=3,N-2
UXX(I)=R12DXS*
1          ( -1.OEO*U(I-2)
2          +16.OEO*U(I-1)
3          -30.OEO*U(I   )
4          +16.OEO*U(I+1)
5          -1.OEO*U(I+2))
1      CONTINUE
RETURN
END

```

The use of subroutine DSS044 is illustrated by the coding of equations (1.21) to (1.24), and (1.28) to (1.31), as listed in Program 3.4.

The main program that calls subroutines INITAL, DERV, and PRINT also calls RKF45 to integrate the 51 ODEs. Both Dirichlet bound-

Program 3.4 Subroutines INITAL, DERV, PRINT, and Data for Equations (1.21) to (1.24) and (1.28) to (1.31)

```

SUBROUTINE INITAL
C...
C...      SUBROUTINE INITAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C...      INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (1.21) TO
C...      (1.24) (NORUN = 1) OR EQUATIONS (1.28) TO (1.31) (NORUN = 2)

```

continues

```

C...
COMMON/T/ TIME, NSTOP, NORUN
1 /Y/ T(51)
2 /F/ TT(51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L, X(51)
C...
C... CALCULATE PI FOR USE IN DO LOOP 1
PI=ACOS(-1.0)
C...
C... LENGTH
L=1.0
C...
C... GRID SPACING
N=51
DX=L/FLOAT(N-1)
C...
C... INITIAL CONDITION (1.22)
IF(NORUN.EQ.1)THEN
DO 1 I=1,N
X(I)=FLOAT(I-1)*DX
T(I)=SIN(PI*X(I)/L)
1 CONTINUE
C...
C... INITIAL CONDITION (1.29)
ELSE IF(NORUN.EQ.2)THEN
DO 2 I=1,N
X(I)=FLOAT(I-1)*DX
T(I)=COS(PI*X(I)/L)
2 CONTINUE
END IF
RETURN
END

SUBROUTINE DERV
C...
C... SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24) (NORUN
C... = 1) OR EQUATIONS (1.28) TO (1.31) (NORUN = 2)
C...
COMMON/T/ TIME, NSTOP, NORUN
1 /Y/ T(51)
2 /F/ TT(51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L, TX(51), TX(51)
C...
C... LENGTH AND NUMBER OF GRID POINTS
L=1.
N=51
C...
C... DIRICHLET BOUNDARY CONDITIONS (1.23) AND (1.24)
IF(NORUN.EQ.1)THEN
NL=1
NU=1
T(1)=0.
T(N)=0.
TT(1)=0.
TT(N)=0.

```

continues

```

C...
C... NEUMANN BOUNDARY CONDITIONS (1.30) AND (1.31)
ELSE IF(NORUN.EQ.2)THEN
    NL=2
    NU=2
    TX(1)=0.
    TX(N)=0.
END IF
C...
C... DERIVATIVE TXX
CALL DSS044(0.,L,N,T,TXX,NL,NU)
C...
C... EQUATION (1.21)
IF(NORUN.EQ.1)THEN
    DO 1 I=2,N-1
    TT(I)=TXX(I)
1   CONTINUE
C...
C... EQUATION (1.28)
ELSE IF(NORUN.EQ.2)THEN
    DO 2 I=1,N
    TT(I)=TXX(I)
2   CONTINUE
2
END IF
RETURN
END

SUBROUTINE PRINT(NI,NO)
C...
C... SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PRO1P3 TO PRINT THE
C... NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24) AND (1.28) TO (1.31)
C...
COMMON/T/ TIME, NSTOP, NORUN
1      /Y/ T(51)
2      /F/ TT(51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL TE(51), DIFF(51)
C...
C... CALCULATE THE EXACT SOLUTION, AND THE DIFFERENCE BETWEEN THE
C... NUMOL AND EXACT SOLUTIONS
N=51
DO 3 I=1,N,10
IF(NORUN.EQ.1)TE(I)=EXACT1(I,TIME)
IF(NORUN.EQ.2)TE(I)=EXACT2(I,TIME)
DIFF(I)=T(I)-TE(I)
3   CONTINUE
C...
C... PRINT THE NUMOL SOLUTION TO EQUATIONS (1.21) TO (1.24), OR (1.28)
C... TO (1.31)
WRITE(NO,2)TIME,(T(I),I=1,N,10),
1           (TE(I),I=1,N,10),
2           (DIFF(I),I=1,N,10)
2   FORMAT(' TIME = ',F6.2,/,15X,', X=0',5X,'X=0.2',5X,'X=0.4',5X,
1           ', X=0.6',5X,'X=0.8',5X,' X=1',/,,
2           ', T(X,T)',6F10.6,/,
3           ', TE(X,T)',6F10.6,/,
4           ', DIFF(X,T)',6F10.6,/)
4
RETURN
END

```

continues

```

REAL FUNCTION EXACT1(I,T)
C...
C... FUNCTION EXACT1 COMPUTES THE EXACT SOLUTION TO EQUATIONS (1.21)
C... TO (1.24), I.E., EQUATION (1.25), AT GRID INDEX I AND TIME T
C...
C... NUMBER OF GRID POINTS
C... PARAMETER (N=51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C... PI FOR USE IN THE EXACT SOLUTION
PI=ACOS(-1.)
C...
C... LENGTH
L=1.
C...
C... GRID SPACING
DX=L/FLOAT(N-1)
C...
C... X FOR WHICH THE EXACT SOLUTION IS TO BE COMPUTED
X=FLOAT(I-1)*DX
C...
C... EXACT SOLUTION AT X AND T
EXACT1=EXP((-PI**2/L**2)*T)*SIN(PI*X/L)
RETURN
END

REAL FUNCTION EXACT2(I,T)
C...
C... FUNCTION EXACT2 COMPUTES THE EXACT SOLUTION TO EQUATIONS (1.28)
C... TO (1.31), I.E., EQUATION (1.32), AT GRID INDEX I AND TIME T
C...
C... NUMBER OF GRID POINTS
C... PARAMETER (N=51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C... PI FOR USE IN THE EXACT SOLUTION
PI=ACOS(-1.)
C...
C... LENGTH
L=1.
C...
C... GRID SPACING
DX=L/FLOAT(N-1)
C...
C... X FOR WHICH THE EXACT SOLUTION IS TO BE COMPUTED
X=FLOAT(I-1)*DX
C...
C... EXACT SOLUTION AT X AND T
EXACT2=EXP((-PI**2/L**2)*T)*COS(PI*X/L)
RETURN
END

NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24)
0.          0.5          0.1
      51          0.0001
NUMOL SOLUTION OF EQUATIONS (1.28) TO (1.31)
0.          0.5          0.1
      51          0.0001
END OF RUNS

```

ary conditions, equations (1.23) and (1.24), and Neumann boundary conditions, equations (1.30) and (1.31), are accommodated by the single call to DSS044 in DERV using the run counter NORUN (the third element in COMMON/T/). The two runs therefore required two sets of data.

The output from the two runs is listed in Table 3.1 (for only $t = 0.3$ to save space). A comparison of this numerical output with that in Tables 1.1 and 1.3 indicates that the latter is considerably less accurate, which is to be expected since the numerical output in Tables 1.1 and 1.3 was produced with the second-order differentiation formulas in subroutine DSS002 (in contrast to the fourth-order formulas in DSS044). However, the better accuracy of DSS044 was produced at a price; the maximum number of calls to subroutine DERV, as originally discussed in Sections 1.6 and 1.7, had to be increased to 15,000 to complete the run, i.e., MAXFCN = 15,000. This increase in the computational effort is due to an *increase in the stiffness* of the 51 ODEs; this numerical condition (stiffness) will be discussed in Chapters 4 and 5.

Clearly, the general procedure we have considered for deriving differ-

Table 3.1 Numerical Output from Program 3.4

```
RUN NO. - 1 NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24)

INITIAL T - 0.000E+00
FINAL T - .500E+00
PRINT T - .100E+00
NUMBER OF DIFFERENTIAL EQUATIONS - 51
MAXIMUM INTEGRATION ERROR - .100E-03
TIME = .30
X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T)  0.000000  .030431  .049239  .049239  .030431  0.000000
TE(X,T) 0.000000  .030432  .049239  .049239  .030432  .000000
DIFF(X,T) 0.000000  -.000000  -.000000  -.000000  -.000000  .000000

RUN NO. - 2 NUMOL SOLUTION OF EQUATIONS (1.28) TO (1.31)

INITIAL T - 0.000E+00
FINAL T - .500E+00
PRINT T - .100E+00
NUMBER OF DIFFERENTIAL EQUATIONS - 51
MAXIMUM INTEGRATION ERROR - .100E-03
TIME = .30
X=0      X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T)  .051787  .041885  .015999  -.015999  -.041885  -.051787
TE(X,T)  .051773  .041885  .015999  -.015999  -.041885  -.051773
DIFF(X,T) .000014  .000000  .000000  .000000  .000000  -.000014
```

entiation formulas can be used to derive a variety of approximations. Also, differentiation formulas can be derived by this method for *nonuniformly spaced grids* (Δx does not have to be constant along the grid from $i = 1$ to N). In fact, Fornberg (3.1) has presented a particularly simple algorithm for computing the weighting coefficients of differentiation formulas on uniform and nonuniform grids. Subroutine DSS032 contains approximations for fourth-order centered and noncentered approximations of first derivatives, which permits the user to concentrate the NUMOL grid points in regions where the solution changes rapidly in space. The derivation of the approximations, based on the Lagrange interpolation polynomial, is given as documentation comments in DSS032. Also, these formulas can be derived by the method proposed by Fornberg (3.1). Other possibilities for spatial differentiation on nonuniform grids includes the use of (a) cubic splines as implemented in subroutines DSS038 and DSS040 and (b) orthogonal collocation as implemented in subroutines DSS046, DSS048, and DSS050 for one, two, and three dimensions, respectively.

We could go on indefinitely discussing the derivation of differentiation formulas to use in the NUMOL. However, because of the limitations of space, we shall consider one final type of problem and the approximations that have been found to be effective for it. Our hope is that the preceding discussion presents the necessary basic ideas for the reader to derive new approximations as required for PDE problems of interest.

3.5 First-Order Hyperbolic PDEs

We conclude this chapter with a discussion of approximations for a group of partial derivatives frequently encountered in physical problems, and which generally require special treatment to produce NUMOL solutions of acceptable accuracy. So far, in this and the preceding chapters, we have considered examples of all of the major geometric classes of PDEs, e.g., equation (1.4), which is parabolic; equations (2.37), (2.38), and (2.39), which are elliptic; and equation (2.21), which is second-order hyperbolic. Now we consider *first-order hyperbolic PDEs*.

To understand how first-order hyperbolic PDEs occur in physical problems, consider the flow of a fluid through a tube at velocity v , as depicted in Figure 3.1. Again, as in Section 1.2, we write an energy balance on a section of the fluid of length Δx .

$$\begin{aligned} \text{Rate of accumulation of energy} &= \text{rate of heat flowing in} \\ &\quad - \text{rate of heat flowing out} \end{aligned} \quad (3.70)$$

[note the similarity of equations (1.1) and (3.70)]. In mathematical terms

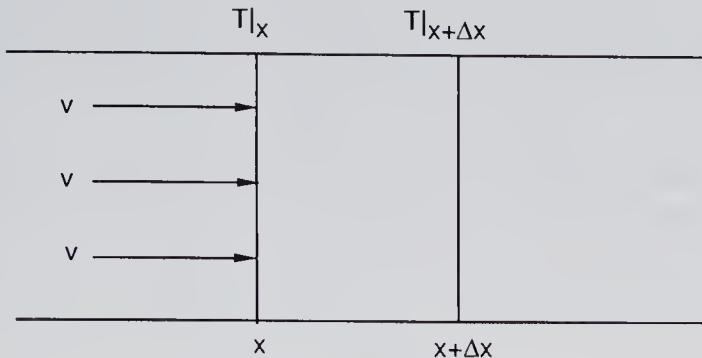


Figure 3.1 Flow of a fluid through a tube.

$$A \Delta X \rho C_p \frac{\partial T}{\partial t} = Av\rho C_p T|_x - Av\rho C_p T|_{x+\Delta x} \quad (3.71)$$

Division by the coefficient of the t derivative, followed by $\Delta x \rightarrow 0$, gives

$$\frac{\partial T}{\partial t} = -v \frac{\partial T}{\partial x} \quad (3.72)$$

Equation (3.72) is termed the *advection equation* since it basically models flow. In the usual subscript notation, equation (3.72) can be written as

$$T_t + vT_x = 0 \quad (3.73)$$

Since equation (3.73) is first-order in t and x , it requires one initial condition and one boundary condition:

$$T(0, t) = f(t), \quad T(x, 0) = g(x) \quad (3.74)(3.75)$$

We now consider some NUMOL approximations for equations (3.73) to (3.75).

Initially, we could reasonably conclude we already have all of the necessary approximations and NUMOL software for equations (3.73) to (3.75). In fact, equation (3.73) is deceptively simple; it is *one of the most difficult PDEs to integrate numerically*. To illustrate this point, we first consider the analytical solution to equations (3.73) to (3.75) for the special case $g(x) = 0$ and $f(t) = 0$ for $t < 0$:

$$T(x, t) = f(t - x/v) \quad (3.76)$$

This solution can be verified easily. If we define $\lambda = t - x/v$, then from equation (3.76), $T(x, t) = f(\lambda)$, i.e., T is a function of only λ . The terms in equation (3.72) become

$$T_t = (dT/d\lambda)(\partial\lambda/\partial t) = dT/d\lambda$$

$$vT_x = v(dT/d\lambda)(\partial\lambda/\partial x) = v(dT/d\lambda)(-1/v)$$

and, on addition of these two equations [and recalling equation (3.73)], $0 = 0$. Also, equation (3.76) satisfies the initial and boundary conditions

$$T(x, 0) = 0, \quad T(0, t) = f(t)$$

The initial condition follows since $f(0 - x/v) = 0$ ($-x/v < 0$ and therefore f has a negative argument, which, as we specified previously, gives f a value of zero).

Now, consider the additional special case $f(t) = 1$, $t > 0$, so that at $x = 0$, the entering temperature undergoes a unit step change at $t = 0$:

$$\begin{aligned} f(t) &= 0, & t < 0 \\ &= 1, & t > 1 \end{aligned} \tag{3.77}$$

The function $f(t)$ defined by equation (3.77) is the *unit step function or Heaviside function, $u(t)$* .

In physical terms, the temperature in the tube is initially zero since $g(x) = 0$ (this does not imply absolute zero, but rather, that the datum on which the temperature is defined is zero, e.g., zero degrees centigrade). Then, at $t = 0$, the entering temperature (at $x = 0$) jumps from zero to one according to equation (3.77);

$$T(0, t) = u(t)$$

It then follows from equation (3.76) that

$$T(x, t) = u(t - x/v) \tag{3.78}$$

i.e., the unit step in temperature at $x = 0$ propagates along the tube at velocity v ; and eventually, at $t = L/v$, the unit step will reach the other end of the tube (L is the length of the tube). At any position along the tube when $t = x/v$, an observer would see a unit step function pass by. In other words, equation (3.73) propagates a finite discontinuity along the tube for this special case.

Why does this cause a problem in computing a NUMOL solution? Consider the slope of the solution at any point x along the tube when $t = x/v$; the slope T_x in equation (3.73) is infinite [which follows from equation (3.78)]. Clearly, any numerical procedure based on well-behaved functions will fail under this condition. In a sense, we are asking the impossible of the numerical approximations we have considered so far for calculating spatial derivatives such as T_x in equation (3.73). With this rather sobering conclusion in mind, we can now consider what happens when we use the previously developed methods.

To summarize our findings so far, we are now dealing with first-order hyperbolic PDEs, such as equation (3.73), which can propagate discontinuities or shocks. Parabolic PDEs, such as Fourier's second law, equation

(1.4), fortunately, do not have this property (as we saw when we considered the damping of a discontinuity due to inconsistent initial and boundary conditions in Section 2.2). Also, elliptic PDEs such as equations (2.37), (2.38), and (2.39), of course, have no time dependence, and therefore the propagation of discontinuities in time has no meaning. However, second-order hyperbolic PDEs, such as equation (2.21), can propagate discontinuities (we considered a smooth case defined by initial conditions (2.23) and (2.24), and boundary conditions (2.25) and (2.26), so this did not happen). Thus, in general, we can conclude that *hyperbolic PDEs are the most difficult class of PDEs to integrate numerically*.

Now, if we proceed as before to develop a NUMOL solution to equations (3.73), (3.74), and (3.75) for the special case $g(x) = 0$ and equation $f(t)$ given by equation (3.77), the coding is straightforward, as shown in Program 3.5. The implementation of the PDE, equation (3.73), in subroutine DERV is for the case $v = L = 1$. Thus, in subroutine PRINT, the exact solution, equation (3.78), for $x = 1$ is just $T(1, t) = 0, t < 1$ and $T(1, t) = 1, t > 1$. The difference between the NUMOL and exact solutions is computed in PRINT (DIFF), and the sum of squares of the differences between the two solutions (SSE) is printed at the end of the two runs to give a single figure of merit for the NUMOL solutions.

Program 3.5 Subroutines INITIAL, DERV, PRINT, and Data for Equations (3.73) to (3.75)

```

SUBROUTINE INITIAL
C...
C...  SUBROUTINE INITIAL IS CALLED BY MAIN PROGRAM PRO1P3 TO DEFINE THE
C...  INITIAL CONDITION IN THE NUMOL SOLUTION OF EQUATIONS (3.73) TO
C...  (3.75)
C...
COMMON/T/    TIME,   NSTOP,   NORUN
1           /Y/   T(51)
2           /F/   TT(51)
C...
C...  TYPE SELECTED VARIABLES AS REAL
REAL L
C...
C...  LENGTH
L=1.0
C...
C...  GRID SPACING
N=51
C...
C...  INITIAL CONDITION (3.75)
DO 1 I=1,N
T(I)=0.
1      CONTINUE
      RETURN
      END

SUBROUTINE DERV
C...
C...  SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
continues

```

```

C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (3.73) TO (3.75) USING
C... SECOND ORDER APPROXIMATIONS (DSS002 CALLED WHEN NORUN = 1) OR
C... FOURTH ORDER APPROXIMATIONS (DSS004 CALLED WHEN NORUN = 2)
C...
C... COMMON/T/    TIME,  NSTOP,  NORUN
1      /Y/  T(51)
2      /F/  TT(51)
C...
C... TYPE SELECTED VARIABLES AS REAL
REAL L, TX(51)
C...
C... LENGTH AND NUMBER OF GRID POINTS
L=1.
N=51
C...
C... BOUNDARY CONDITION (3.74)
T(1)=1.
TT(1)=0.
C...
C... DERIVATIVE TX
IF(NORUN.EQ.1)THEN
    CALL DSS002(0.,L,N,T,TX)
ELSE IF(NORUN.EQ.2)THEN
    CALL DSS004(0.,L,N,T,TX)
END IF
C...
C... EQUATION (3.73)
DO 1 I=2,N
    TT(I)=-TX(I)
1 CONTINUE
RETURN
END

SUBROUTINE PRINT(NI,NO)
C...
C... SUBROUTINE PRINT IS CALLED BY MAIN PROGRAM PRO1P3 TO PRINT THE
C... NUMOL SOLUTION OF EQUATIONS (3.73) TO (3.75)
C...
C... COMMON/T/    TIME,  NSTOP,  NORUN
1      /Y/  T(51)
2      /F/  TT(51)
C...
C... PRINT A HEADING FOR THE SOLUTION AND INITIALIZE THE SUM OF
C... SQUARES OF THE ERRORS
IF(TIME.LT.0.001)THEN
    WRITE(NO,1)
    1 FORMAT(' ADVECTION EQUATION',//,
    1      3X,' TIME          T(0,t)      T(1,t)      TE(1,t)      DIFF')
    SSE=0.
    END IF
C...
C... CALCULATE THE EXACT SOLUTION, AND THE DIFFERENCE BETWEEN THE
C... NUMOL AND EXACT SOLUTIONS
N=51
IF(TIME.LT.1.)TE=0.
IF(TIME.EQ.1.)TE=0.5
IF(TIME.GT.1.)TE=1.
DIFF=T(N)-TE
C...
C... PRINT THE NUMOL AND EXACT SOLUTIONS
WRITE(NO,2)TIME,T(1),T(N),TE,DIFF
2 FORMAT(F10.1,4F12.4)

```

continues

```

C...
C... UPDATE THE SUM OF SQUARES OF THE ERRORS
SSE=SSE+DIFF**2
C...
C... PRINT THE SUM OF SQUARES OF THE ERRORS AT THE END OF THE RUN
IF(TIME.GT.1.999)WRITE(NO,3)SSE
3  FORMAT(//, ' SSE = ',F7.3,/)
      RETURN
    END

NUMOL SOLUTION OF EQUATIONS (3.73) TO (3.75) - DSS002
0.          2.0          0.1
      51          0.0001
NUMOL SOLUTION OF EQUATIONS (3.73) TO (3.75) - DSS004
0.          2.0          0.1
      51          0.0001
END OF RUNS

```

Table 3.2 Numerical Output from Program 3.5

```

RUN NO. -     1 NUMOL SOLUTION OF EQUATIONS (3.73) TO (3.75) - DSS002

INITIAL T -   0.000E+00

FINAL T -    .200E+01

PRINT T -    .100E+00

NUMBER OF DIFFERENTIAL EQUATIONS -  51

MAXIMUM INTEGRATION ERROR -    .100E-03

ADVECTION EQUATION

      TIME      T(0,t)      T(1,t)      TE(1,t)      DIFF
      0.0      1.0000      0.0000      0.0000      0.0000
      .1      1.0000      0.0000      0.0000      0.0000
      .2      1.0000      .0000      0.0000      .0000
      .3      1.0000      .0000      0.0000      .0000
      .4      1.0000      .0000      0.0000      .0000
      .5      1.0000      .0000      0.0000      .0000
      .6      1.0000      .0000      0.0000      .0000
      .7      1.0000      .0000      0.0000      .0000
      .8      1.0000      .0010      0.0000      .0010
      .9      1.0000      .0327      0.0000      .0327
      1.0      1.0000      .3434      0.0000      .3434
      1.1      1.0000      1.1141      1.0000      .1141
      1.2      1.0000      1.0333      1.0000      .0333
      1.3      1.0000      .9642      1.0000      -.0358
      1.4      1.0000      1.0005      1.0000      .0005
      1.5      1.0000      1.0425      1.0000      .0425
      1.6      1.0000      .9306      1.0000      -.0694
      1.7      1.0000      1.0573      1.0000      .0573
      1.8      1.0000      .9921      1.0000      -.0079
      1.9      1.0000      .9610      1.0000      -.0390
      2.0      1.0000      1.0380      1.0000      .0380

SSE =     .147

```

continues

```
RUN NO. - 2 NUMOL SOLUTION OF EQUATIONS (3.73) TO (3.75) - DSS004
INITIAL T - 0.000E+00
FINAL T - .200E+01
PRINT T - .100E+00
NUMBER OF DIFFERENTIAL EQUATIONS - 51
MAXIMUM INTEGRATION ERROR - .100E-03
ADVECTION EQUATION
TIME      T(0,t)      T(1,t)      TE(1,t)      DIFF
0.0       1.0000     0.0000     0.0000     0.0000
.1        1.0000     .0000      0.0000     .0000
.2        1.0000     .0000      0.0000     .0000
.3        1.0000     .0000      0.0000     .0000
.4        1.0000     .0000      0.0000     .0000
.5        1.0000     .0000      0.0000     .0000
.6        1.0000     .0000      0.0000     .0000
.7        1.0000     .0000      0.0000     .0000
.8        1.0000     .0002      0.0000     .0002
.9        1.0000    -.0113     0.0000    -.0113
1.0       1.0000     .3973     0.0000     .3973
1.1       1.0000     1.0883     1.0000     .0883
1.2       1.0000     1.1298     1.0000     .1298
1.3       1.0000     1.0318     1.0000     .0318
1.4       1.0000     .9574     1.0000    -.0426
1.5       1.0000     .9211     1.0000    -.0789
1.6       1.0000     .9080     1.0000    -.0920
1.7       1.0000     .9058     1.0000    -.0942
1.8       1.0000     .9076     1.0000    -.0924
1.9       1.0000     .9105     1.0000    -.0895
2.0       1.0000     .9131     1.0000    -.0869
SSE =   .233
```

The output from the execution of Program 3.5 is listed in Table 3.2.

The output in Table 3.2 indicates that the numerical solution does not closely follow the exact solution. Note in particular the oscillatory behavior in which the numerical solution goes above the exact solution of $T_{\text{exact}}(1, t) = 1$ for $t > 1$ [e.g., $T(1, 1.1) = 1.1141$ in Run 1, an 11.41% overshoot] followed by a drop below the exact solution [e.g., $T(1, 1.6) = 0.9306$, a 6.94% undershoot]. This *numerical oscillation* is evident in the plot of the two solutions in Figure 3.2 (for DSS002) and Figure 3.3 (for DSS004) [note that the exact solution, equation (3.78), is superimposed on Figures 3.2 and 3.3]. We might then consider how the numerical solution can be improved.

One approach for improving the numerical solution is based partly on physical reasoning. Since equation (3.73) models the flow in Figure 3.1, the values of $T(x, t)$ to the left of the point where the derivative T_x is to be approximated may be more influential than those to the right (since the flow is left to right in Figure 3.1); in other words, we might consider using

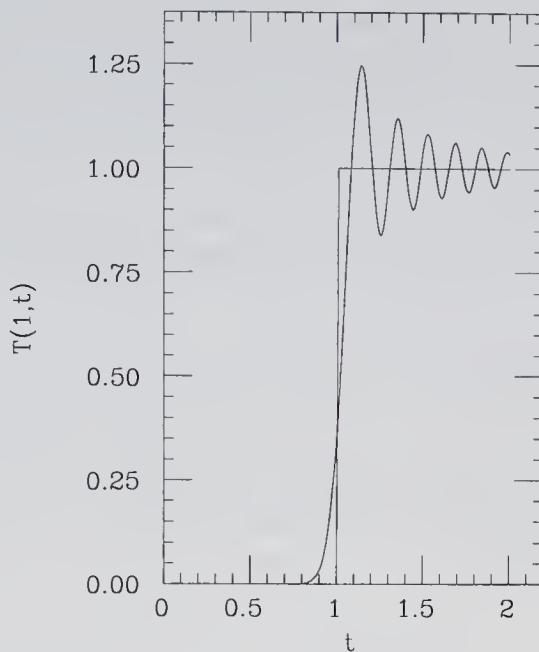


Figure 3.2 NUMOL and exact solutions to equations (3.73) to (3.75) from Subroutine DSS002.

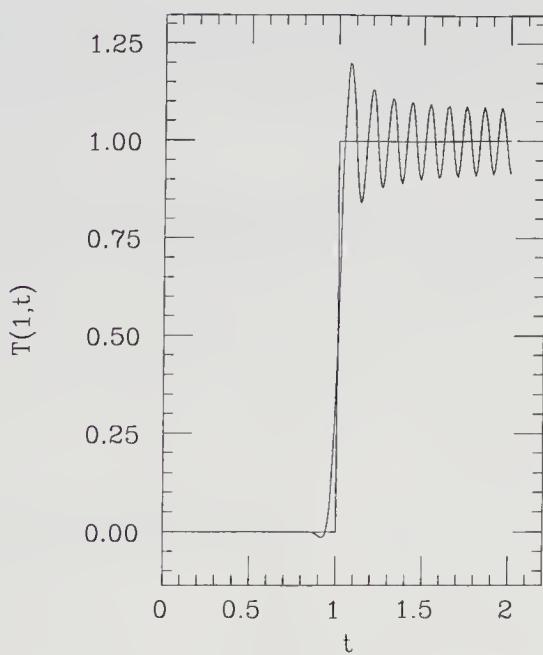


Figure 3.3 NUMOL and exact solutions to equations (3.73) to (3.75) from Subroutine DSS004.

upwind points in the calculation of T_x . The simplest approximation that meets this requirement is the *first-order two-point upwind approximation*

$$T_x(x, t) = \frac{T(x, t) - T(x - \Delta x, t)}{\Delta x} + O(\Delta x) \quad (3.79)$$

Equation (3.79) is easily implemented in subroutine DERV for the calculation of the spatial derivative in equation (3.73), T_x ; in fact, this has

Program 3.6 Subroutine DSS012

```

SUBROUTINE DSS012(XL,XU,N,U,UX,V)
C...
C... SUBROUTINE DSS012 IS AN APPLICATION OF FIRST-ORDER DIRECTIONAL
C... DIFFERENCING IN THE NUMERICAL METHOD OF LINES. IT IS INTENDED
C... SPECIFICALLY FOR THE ANALYSIS OF CONVECTIVE SYSTEMS MODELLED BY
C... FIRST ORDER HYPERBOLIC PARTIAL DIFFERENTIAL EQUATIONS WITH THE
C... SIMPLEST FORM
C...
C... THE FIRST FIVE PARAMETERS, XL, XU, N, U AND UX, ARE THE SAME
C... AS FOR SUBROUTINES DSS002 TO DSS010 AS DEFINED IN THOSE ROUTINES.
C... THE SIXTH PARAMETER, V, MUST BE PROVIDED TO DSS012 SO THAT THE
C... DIRECTION OF FLOW IN EQUATION (1) CAN BE USED TO SELECT THE
C... APPROPRIATE FINITE DIFFERENCE APPROXIMATION FOR THE FIRST ORDER
C... SPATIAL DERIVATIVE IN EQUATION (1), U . THE CONVENTION FOR THE
C... SIGN OF V IS
C...
C...      FLOW LEFT TO RIGHT          V GT 0
C...      (I.E., IN THE DIRECTION    (I.E., THE SIXTH ARGUMENT IS
C...      OF INCREASING X)           POSITIVE IN CALLING DSS012)
C...
C...      FLOW RIGHT TO LEFT        V LT 0
C...      (I.E., IN THE DIRECTION    (I.E., THE SIXTH ARGUMENT IS
C...      OF DECREASING X)           NEGATIVE IN CALLING DSS012)
C...
C...      REAL U(N),UX(N)
C...
C...      COMPUTE THE SPATIAL INCREMENT, THEN SELECT THE FINITE DIFFERENCE
C...      APPROXIMATION DEPENDING ON THE SIGN OF V IN EQUATION (1).
DX=(XU-XL)/FLOAT(N-1)
IF(V.LT.0.)GO TO 10
C...
C...      (1)  FINITE DIFFERENCE APPROXIMATION FOR POSITIVE V
UX(1)=(U(2)-U(1))/DX
DO 1 I=2,N
UX(I)=(U(I)-U(I-1))/DX
CONTINUE
RETURN
C...
C...      (2)  FINITE DIFFERENCE APPROXIMATION FOR NEGATIVE V
10 NM1=N-1
DO 2 I=1,NM1
UX(I)=(U(I+1)-U(I))/DX
CONTINUE
UX(N)=(U(N)-U(N-1))/DX
2 RETURN
END

```

already been done in subroutine DSS012, which is listed in Program 3.6. Note that subroutine DSS012 has a sixth argument defining the direction of flow; if this argument is positive (and real), the two-point upwind approximation for flow in the positive direction is used [via DO loop 1 according to equation (3.79)]; conversely, if the sixth argument is negative, a two-point upwind approximation in the other direction is used (via DO loop 2). Also, at the first grid point for positive flow, $i = 1$, a downwind approximation is used since an upwind approximation would require a fictitious value, $U(0)$; similarly at the first grid point for negative flow, $i = N$, a downwind approximation is used since an upwind approximation would require a fictitious value, $U(N + 1)$.

The use of DSS012 is similar to DSS002 and DSS004 in Program 3.5; only subroutine DERV is listed in Program 3.7 for equations (3.73) to (3.75) (all of the other coding in Program 3.5 remains unchanged).

The numerical output from Program 3.7 is listed in Table 3.3. Note that the numerical solution does not oscillate as in the case of the solutions from DSS002 and DSS004 (Table 3.2). However, the numerical solution lags behind the exact solution, i.e., it is spread out in time [as well as in space, which would be evident if the spatial profiles were plotted as $T(x, t)$]

Program 3.7 Subroutine DERV with a Call to DSS012 for Equations (3.73) to (3.75)

```

SUBROUTINE DERV
C...
C... SUBROUTINE DERV IS CALLED BY INTEGRATOR RKF45 TO DEFINE THE
C... ODES IN THE NUMOL SOLUTION OF EQUATIONS (3.73) TO (3.75) USING
C... FIRST ORDER UPWIND APPROXIMATIONS (IN DSS012)
C...
      COMMON/T/    TIME,   NSTOP,   NORUN
      1          /Y/    T(51)
      2          /F/    TT(51)
C...
C... TYPE SELECTED VARIABLES AS REAL
      REAL L, TX(51)
C...
C... LENGTH AND NUMBER OF GRID POINTS
      L=1.
      N=51
C...
C... BOUNDARY CONDITION (3.74)
      T(1)=1.
      TT(1)=0.
C...
C... DERIVATIVE TX
      V=1.
      CALL DSS012(0.,L,N,T,TX,V)
C...
C... EQUATION (3.73)
      DO 1 I=2,N
      TT(I)=-TX(I)
1     CONTINUE
      RETURN
      END

```

Table 3.3 Numerical Output from Program 3.7

```

RUN NO. -      1 NUMOL SOLUTION OF EQUATIONS (3.73) TO (3.75) - DSS012
INITIAL T -   0.000E+00
FINAL T -    .200E+01
PRINT T -    .100E+00
NUMBER OF DIFFERENTIAL EQUATIONS -  51
MAXIMUM INTEGRATION ERROR -   .100E-03
ADVECTION EQUATION

TIME      T(0,t)      T(1,t)      TE(1,t)      DIFF
0.0       1.0000      0.0000      0.0000      0.0000
.1        1.0000      .0000       0.0000      .0000
.2        1.0000      .0000       0.0000      .0000
.3        1.0000      .0000       0.0000      .0000
.4        1.0000      .0000       0.0000      .0000
.5        1.0000      .0000       0.0000      .0000
.6        1.0000      .0005       0.0000      .0005
.7        1.0000      .0099       0.0000      .0099
.8        1.0000      .0703       0.0000      .0703
.9        1.0000      .2468       0.0000      .2468
1.0       1.0000      .5188       0.0000      .5188
1.1       1.0000      .7678       1.0000     -.2322
1.2       1.0000      .9156       1.0000     -.0844
1.3       1.0000      .9765       1.0000     -.0235
1.4       1.0000      .9949       1.0000     -.0051
1.5       1.0000      .9991       1.0000     -.0009
1.6       1.0000      .9999       1.0000     -.0001
1.7       1.0000      1.0000       1.0000     -.0000
1.8       1.0000      1.0000       1.0000     .0000
1.9       1.0000      1.0000       1.0000     .0000
2.0       1.0000      1.0000       1.0000     .0000

SSE =   .397

```

vs. x for a given t]. This distortion of the solution is termed *numerical diffusion*, and is clearly evident in Figure 3.4.

To this point, we have observed that centered differences, as in DSS002 to DSS004, produce excessive numerical oscillation in the solution to equation (3.73), while the two-point upwind approximations in DSS012 eliminated the oscillation, but produced excessive numerical diffusion. Therefore, we might again consider the upwind approximation, but with more grid points. This can easily be done by reprogramming subroutine DSS002 (in Program 3.1) so that a three-point upwind approximation is used at grid points $i = 3, 4, \dots, N$, which is immediately available from equation (3.9) for a positive velocity. Also, to avoid fictitious values, a three-point centered approximation is used at grid point $i = 2$ [from equation (3.5)] and a three-point downwind approximation is used at grid point $i = 1$ [from equation (3.8)]. This reprogramming of equations (3.5), (3.8), and (3.9) is implemented in subroutine DSS014 listed in Program 3.8, for

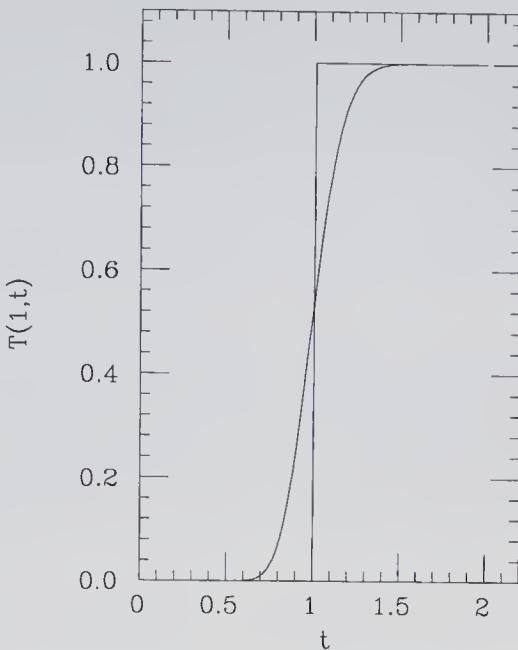


Figure 3.4 NUMOL and exact solutions to equations (3.73) to (3.75) from Subroutine DSS012.

Program 3.8 Subroutine DSS014

```

SUBROUTINE DSS014(XL,XU,N,U,UX,V)
C...
C... SUBROUTINE DSS014 IS AN APPLICATION OF SECOND ORDER DIRECTIONAL
C... DIFFERENCING IN THE NUMERICAL METHOD OF LINES. IT IS INTENDED
C... SPECIFICALLY FOR THE ANALYSIS OF CONVECTIVE SYSTEMS MODELLED BY
C... FIRST ORDER HYPERBOLIC PARTIAL DIFFERENTIAL EQUATIONS AS DIS-
C... CUSSED IN SUBROUTINE DSS012.
C...
REAL U(N),UX(N)
C...
C... COMPUTE THE COMMON FACTOR FOR EACH FINITE DIFFERENCE APPROXIMATION
C... CONTAINING THE SPATIAL INCREMENT, THEN SELECT THE FINITE DIFFER-
C... ENCE APPROXIMATION DEPENDING ON THE SIGN OF V (SIXTH ARGUMENT).
DX=(XU-XL)/FLOAT(N-1)
R2FDX=1./(2.*DX)
IF(V.LT.0.)GO TO 10
C...
C...      (1) FINITE DIFFERENCE APPROXIMATION FOR POSITIVE V
UX(1)=R2FDX*
1(   -3.    *U(  1)      +4.    *U(  2)      -1.    *U(  3))
UX(2)=R2FDX*
1(   -1.    *U(  1)      +0.    *U(  2)      +1.    *U(  3))
DO 1 I=3,N
UX(I)=R2FDX*
1(   1.    *U(I-2)      -4.    *U(I-1)      +3.    *U(  I))
1
CONTINUE
RETURN

```

continues

```

C...
C...      (2) FINITE DIFFERENCE APPROXIMATION FOR NEGATIVE V
10     NM2=N-2
      DO 2 I=1,NM2
      UX(I)=R2FDX*
      1( -3. *U(I)      +4. *U(I+1)      -1. *U(I+2))
2     CONTINUE
      UX(N-1)=R2FDX*
      1( -1. *U(N-2)    +0. *U(N-1)      +1. *U(N))
      1( 1. *U(N-2)     -4. *U(N-1)      +3. *U(N))
      RETURN
      END

```

both positive and negative velocities. The numerical solution from Program 3.7 with a call to DSS014 in place of DSS012 is plotted in Figure 3.5 (the numerical output is not printed to save space). Unfortunately, the numerical solution again oscillates, as evident from Figure 3.5. Note that the oscillation from DSS014 (three-point upwind) is to the left of the step change in the exact solution (Figure 3.5) while the numerical oscillation from DSS002 (three-point centered) and DSS004 (five-point centered) is to the right of the step change (Figures 3.2 and 3.3). Thus, perhaps a combination of centered and upwind approximations will produce less oscillation on both sides of the step change, as proposed by Carver and Hinds (3.4); fortunately, this turns out to be the case.

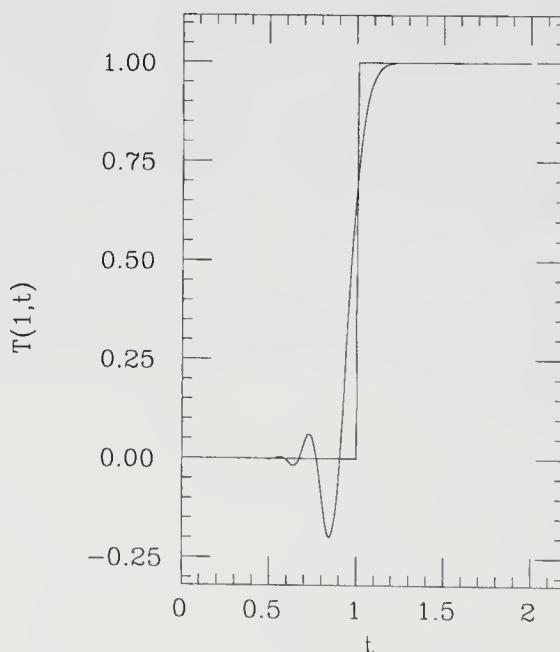


Figure 3.5 NUMOL and exact solutions to equations (3.73) to (3.75) from Subroutine DSS014.

To explore this approach we can use equation (3.41) for positive velocity or equation (3.40) for negative velocity, as programmed in subroutine DSS020, listed in Program 3.9. Note that again in order to avoid fictitious values when the derivative is computed near the boundaries, equations (3.35), (3.40), (3.26), and (3.42) must be used at grid points

Program 3.9 Subroutine DSS020

```

SUBROUTINE DSS020(XL,XU,N,U,UX,V)
C...
C... SUBROUTINE DSS020 IS AN APPLICATION OF FOURTH ORDER DIRECTIONAL
C... DIFFERENCING IN THE NUMERICAL METHOD OF LINES. IT IS INTENDED
C... SPECIFICALLY FOR THE ANALYSIS OF CONVECTIVE SYSTEMS MODELLED BY
C... FIRST ORDER HYPERBOLIC PARTIAL DIFFERENTIAL EQUATIONS AS DIS-
C... CUSED IN SUBROUTINE DSS012. THE USE OF FIVE POINT BIASED
C... UPWIND APPROXIMATIONS WAS PROPOSED AND DESCRIBED IN THE PAPER
C... M. B. CARVER AND H. W. HINDS, THE METHOD OF LINES AND THE
C... ADVECTION EQUATION, SIMULATION, VOL. 31, NO. 2, PP. 59-69,
C... AUGUST, 1978
C...
REAL U(N),UX(N)
C...
C... COMPUTE THE COMMON FACTOR FOR EACH FINITE DIFFERENCE APPROXIMATION
C... CONTAINING THE SPATIAL INCREMENT, THEN SELECT THE FINITE DIFFER-
C... ENCE APPROXIMATION DEPENDING ON THE SIGN OF V (SIXTH ARGUMENT).
DX=(XU-XL)/FLOAT(N-1)
R4FDX=1./(12.*DX)
IF(V.LT.0.)GO TO 10
C...
C... (1) FINITE DIFFERENCE APPROXIMATION FOR POSITIVE V
UX( 1)=R4FDX*
1( -25.*U( 1) +48.*U( 2) -36.*U( 3) +16.*U( 4) -3.*U( 5))
UX( 2)=R4FDX*
1( -3.*U( 1) -10.*U( 2) +18.*U( 3) -6.*U( 4) +1.*U( 5))
UX( 3)=R4FDX*
1( +1.*U( 1) -8.*U( 2) +0.*U( 3) +8.*U( 4) -1.*U( 5))
NM1=N-1
DO 1 I=4,NM1
UX(  I)=R4FDX*
1( -1.*U(I-3) +6.*U(I-2) -18.*U(I-1) +10.*U( I) +3.*U(I+1))
CONTINUE
1( 3.*U(N-4) -16.*U(N-3) +36.*U(N-2) -48.*U(N-1) +25.*U( N))
RETURN
C...
C... (2) FINITE DIFFERENCE APPROXIMATION FOR NEGATIVE V
10 UX( 1)=R4FDX*
1( -25.*U( 1) +48.*U( 2) -36.*U( 3) +16.*U( 4) -3.*U( 5))
NM3=N-3
DO 2 I=2,NM3
UX(  I)=R4FDX*
1( -3.*U(I-1) -10.*U( I) +18.*U(I+1) -6.*U(I+2) +1.*U(I+3))
CONTINUE
UX(N-2)=R4FDX*
1( +1.*U(N-4) -8.*U(N-3) +0.*U(N-2) +8.*U(N-1) -1.*U( N))
UX(N-1)=R4FDX*
1( -1.*U(N-4) +6.*U(N-3) -18.*U(N-2) +10.*U(N-1) +3.*U( N))
UX(  N)=R4FDX*
1( 3.*U(N-4) -16.*U(N-3) +36.*U(N-2) -48.*U(N-1) +25.*U( N))
RETURN
END

```

$i = 1, 2, 3$, and N for positive velocity, respectively, and equations (3.42), (3.41), (3.26), and (3.35) must be used at grid points $i = N, N - 1, N - 2$, and 1 for negative velocity, respectively. The numerical solution from Program 3.7 with a call to DSS020 in place of DSS012 is plotted in Figure 3.6 (the numerical output is not printed to save space). Although the numerical solution has some numerical diffusion and oscillation, this distortion is considerably less than in the cases of subroutines DSS002, DSS004, DSS012, and DSS014. For this reason, we recommend the use of DSS020 for the NUMOL solution of first-order hyperbolic PDEs.

Recall, also, that we are attempting to solve essentially an impossible problem when $f(t)$ is given by equation (3.77) since u_x computed approximately by DSS020 is really infinite at any point along the system when $t = x/v$; thus, if $x = v = 1$, at $t = 1$ the solution has an infinite slope in t (and x), as indicated by the exact solution in Figure 3.6. In most physical problems, we would not expect a dependent variable to change with an infinite slope in time or space. Of course, there are exceptions (e.g., the shock wave or front created when an aircraft exceeds the speed of sound, as manifest by a “sonic boom,” shocks created by explosions, etc.), but typically in a realistic model, we would not expect propagating discontinuities; generally, there will be some physical phenomenon that will produce

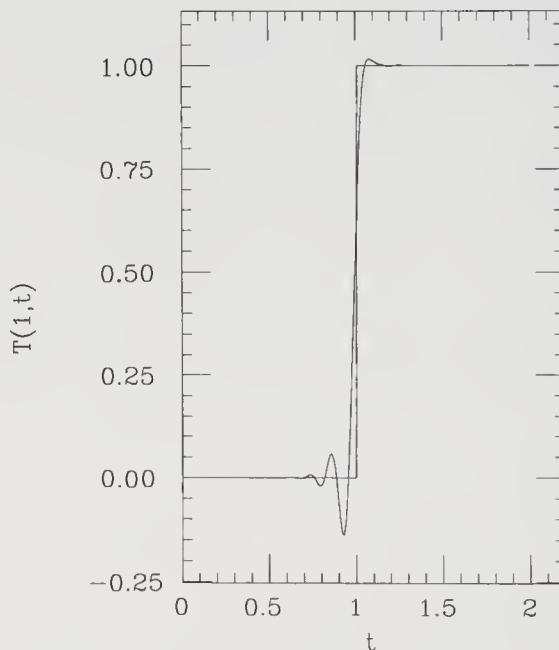


Figure 3.6 NUMOL and exact solutions to equations (3.73) to (3.75) from Subroutine DSS020.

a finite rate of change, e.g., molecular diffusion of temperature, mass, or momentum. Therefore, we might consider a function $f(t)$ that is less stringent, and more typical, than the unit step, $u(t)$, of equation (3.77). We therefore replace the unit step function with a *ramp function* $r(t) = T(0, t)$, which has a steep, but finite, slope

$$r(t) = \begin{cases} 0, & t < 0 \\ st, & 0 < t < 1/s \\ 1, & t > 1/s \end{cases} \quad (3.80)$$

where we take $s = 5$.

The exact solution for this *boundary condition function* follows immediately from equation (3.76) since $f(t) = r(t)$

$$T(x, t) = r(t - x/v) \quad (3.81)$$

Equation (3.80) is easily programmed as the boundary condition at $x = 0$ in subroutine DERV of Program 3.7 [in place of the coding for $u(t)$]

```
C...
C... TRUNCATED RAMP FUNCTION
S=5.
IF(TIME.LT.0.)T(1)=0.
IF(TIME.GT.(1./S))T(1)=1.
IF((TIME.GE.0.).AND.(TIME.LE.(1./S)))T(1)=S*TIME
TT(1)=0.
```

Similarly, the coding of equation (3.81), used in subroutine PRINT of Program 3.7 to print and plot the exact solution (TE), is

```
C...
C... COMPUTE THE EXACT SOLUTION FOR PRINTING AND PLOTTING
L=1.
V=1.
S=5.
TXV=TIME-L/V
IF(TXV.LT.0.)TE=0.
IF(TXV.GT.(1./S))TE=1.
IF((TXV.GE.0.).AND.(TXV.LE.(1./S)))TE=S*TXV
```

The numerical and exact solutions for the truncated ramp using subroutine DSS012 is plotted in Figure 3.7. Again, excessive numerical diffusion occurs, as in Figure 3.4.

However, if subroutine DSS020 is used with $r(t)$ programmed as above, the numerical and exact solutions are in much better agreement, as indicated in Figure 3.8.

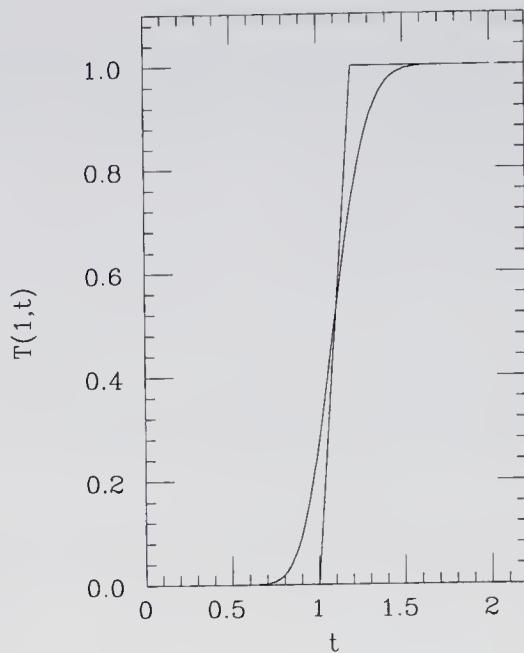


Figure 3.7 NUMOL and exact solutions to equations (3.73) to (3.75) from Subroutine DSS012 for a truncated ramp boundary condition.

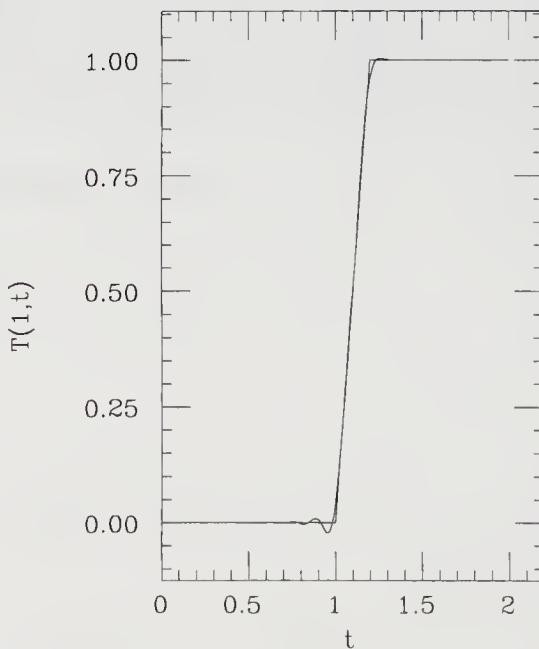


Figure 3.8 NUMOL and exact solutions to equations (3.73) to (3.75) from Subroutine DSS020 for a truncated ramp boundary condition.

As one final example, we consider another boundary condition function, a *cosine pulse*, $p(t)$:

$$p(t) = \begin{cases} 0, & \omega t < 0 \\ 1 - \cos(\omega t), & 0 < \omega t < \pi/2 \\ 1 + \cos(\omega t), & \pi/2 < \omega t < \pi \\ 0, & \omega t > \pi \end{cases} \quad (3.82)$$

The exact solution for this boundary condition function again follows immediately from equation (3.76) since $f(t) = p(t)$:

$$T(x, t) = p(t - x/v) \quad (3.83)$$

Equation (3.82) is also easily programmed as the boundary condition at $x = 0$ in subroutine DERV of Program 3.7 [in place of the coding for $u(t)$]:

```
C...
C...      COSINE PULSE FUNCTION
W=5.
PI=3.1415927
IF(TIME.LT.0.)T(1)=0.
IF((W*TIME).GT.(PI))T(1)=0.
IF(((W*TIME).GE.(0.0)).AND.((W*TIME).LE.(PI/2.)))
1   T(1)=1.-COS(W*TIME)
IF(((W*TIME).GT.(PI/2.)).AND.((W*TIME).LE.(PI)))
1   T(1)=1.+COS(W*TIME)
TT(1)=0.
```

The coding of equation (3.83), used in subroutine PRINT of Program 3.7 to print and plot the exact solution (TE), is

```
C...
C...      COMPUTE THE EXACT SOLUTION FOR PRINTING AND PLOTTING
L=1.
V=1.
W=5.
PI=3.1415927
TXV=TIME-L/V
IF(TXV.LT.0.)TE=0.
IF((W*TXV).GT.(PI))TE=0.
IF(((W*TXV).GE.(0.0)).AND.((W*TXV).LE.(PI/2.)))TE=1.-COS(W*TXV)
IF(((W*TXV).GT.(PI/2.)).AND.((W*TXV).LE.(PI))) TE=1.+COS(W*TXV)
```

The numerical solution from DSS012 again has excessive numerical diffusion, as indicated in Figure 3.9. Subroutine DSS020 gives much better agreement between the numerical and exact solutions, as indicated in Figure 3.10.

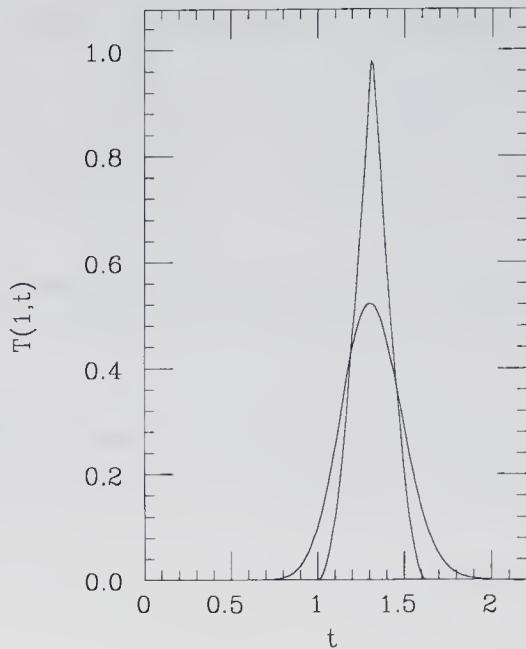


Figure 3.9 NUMOL and exact solutions to equations (3.73) to (3.75) from Subroutine DSS012 for a cosine pulse boundary condition.

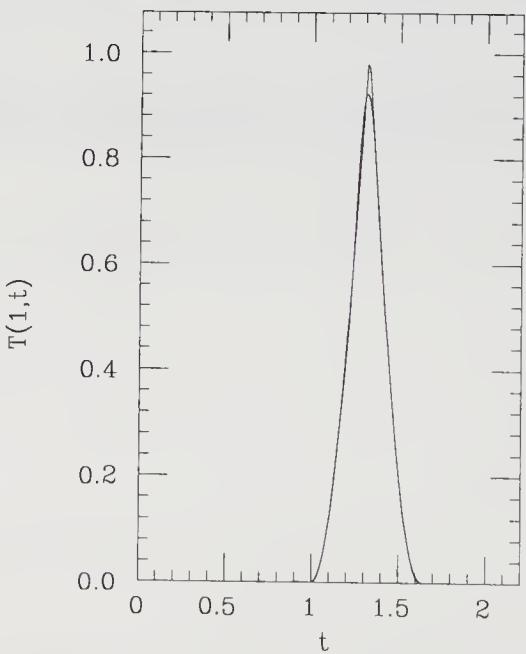


Figure 3.10 NUMOL and exact solutions to equations (3.73) to (3.75) from Subroutine DSS020 for a cosine pulse boundary condition.

Equation (3.82) for $p(t)$ represents a rather important application, that of a pulse moving along a flowing system; typical applications include the response of adsorption and chromatography systems. If these systems are to be simulated realistically, the shape of the pulse that leaves the system (at $x = L$) must be computed accurately. The numerical diffusion of Figure 3.9 would invalidate the simulation, while the numerical solution of Figure 3.10 is quite acceptable.

In summary, we have found the five-point biased upwind approximations proposed by Carver and Hinds (3.4) quite effective in the simulation of convective systems modeled by first-order hyperbolic PDEs, at least when compared with other standard finite difference approximations for spatial derivatives implemented on a fixed grid (e.g., the grid of 51 points in the preceding examples, which did not change as the solution evolved through time). Two relatively complex convective systems are simulated in Chapters 5 and 6 based on the use of these approximations, followed by some discussion of methods that can be used when the fixed-grid NUMOL cannot be used.

References

- (3.1) Fornberg, Bengt, "Generation of Finite Difference Formulas on Arbitrarily Spaced Grids," *Math. Computation*, **51** (184), 699–706 (1988).
- (3.2) Sincovec, Richard F., and Neil K. Madsen, "Software for Nonlinear Partial Differential Equations," *A.C.M. Trans. Math. Software*, **1**, 232–260 (1975).
- (3.3) Madsen, Neil K., and Richard F. Sincovec, "Software for Partial Differential Equations," in *Numerical Methods for Differential Systems*, L. Lapidus and W. E. Schiesser, eds., Academic Press, New York, 1976.
- (3.4) Carver, Michael B., and H. W. Hinds, "The Method of Lines and the Advection Equation," *Simulation* **31**, (2), 59–69 (1978).

Problems

- (3.1) What is the $O(\Delta x^2)$ term in equation (3.8)?
- (3.2) Derive equation (3.9)
- (3.3) Derive equation (3.26) using the given values of a to d .
- (3.4) Apply equation (3.8) to equation (3.11). What can you conclude from the result?
- (3.5) Apply equation (3.9) to equation (3.11). What can you conclude from the result?

- (3.6) Apply equation (3.8) to equation (3.12). What do you conclude from the result?
- (3.7) Apply equation (3.9) to equation (3.12). What do you conclude from the result?
- (3.8) Apply equation (3.8) to equation (3.13). What do you conclude from the result, particularly with regard to the result of applying equation (3.5) to (3.13)?
- (3.9) Apply equation (3.9) to equation (3.13). What do you conclude from the result, particularly with regard to the result of applying equation (3.5) to (3.13)?
- (3.10) Apply equation (3.26) to equation (3.11).
- (3.11) Apply equation (3.35) to equation (3.11).
- (3.12) Derive equations (3.36) to (3.39) and equation (3.40).
- (3.13) Derive equation (3.41).
- (3.14) Derive equation (3.42).
- (3.15) Use subroutine DSS002 to calculate the first and second derivatives of $\sin(\pi x)$ at $x = 0$ and $x = 0.4$ (where x covers the spatial interval $0 \leq x \leq 1$). Plot the log of the absolute value of the error in these derivatives vs. $\log(1/N)$ where N is the total number of grid points; use $N = 11, 21, 31, 41, 51$. What can you conclude from this plot concerning (a) the variation of the errors with N , (b) the relative accuracy of the numerical first and second derivatives, and (c) the accuracy of the numerical differentiation at the end of the spatial interval ($x = 0$) vs. the accuracy in the interior of the spatial interval ($x = 0.4$)?
- (3.16) Do Problem (3.15) using subroutine DSS004.
- (3.17) Derive equation (3.60).
- (3.18) Why are the weighting coefficients of equations (3.66) and (3.67) the same except for the first derivative (for which they are opposite in sign)?
- (3.19) Verify that equations (3.66) and (3.67) differentiate a constant correctly.

- (3.20) Verify that equations (3.66) and (3.67) differentiate $u(x) = a_0 + a_1x$ correctly. Will equations (3.66) and (3.67) differentiate a fourth-order polynomial correctly? A fifth-order polynomial?
- (3.21) Derive equation (3.79).
- (3.22) Plot the spatial profiles [$T(x, t)$ vs. x] for $t = 0.5, 1.0, 1.5$, and 2.0 corresponding to the variation of the numerical solution with t in Figure 3.4.
- (3.23) Use the following coding for the truncated ramp, $f(t) = r(t)$, defined by equation (3.80) to compute the solution to equations (3.73) to (3.75)

```
C...
C...      TRUNCATED RAMP FUNCTION
S=5.
IF(TIME.LT.0.)T(1)=0.
IF(TIME.GT.(1./S))T(1)=1.
IF((TIME.GE.0.).AND.(TIME.LE.(1./S)))T(1)=S*TIME
TT(1)=0.
```

This coding will appear in subroutine DERV of Program 3.7. Also, $g(x) = 0$ will be used in subroutine INITAL. Perform the spatial differentiation (calculation of T_x) in equation (3.73), using

- (a) Subroutine DSS002
- (b) Subroutine DSS012
- (c) Subroutine DSS020

In each case, compare the NUMOL solution with the exact solution, equation (3.81), programmed in subroutine PRINT using the coding

```
C...
C...      COMPUTE THE EXACT SOLUTION FOR PRINTING AND PLOTTING
L=1.
V=1.
S=5.
TXV=TIME-L/V
IF(TXV.LT.0.)TE=0.
IF(TXV.GT.(1./S))TE=1.
IF((TXV.GE.0.).AND.(TXV.LE.(1./S)))TE=S*TXV
```

- (3.24) Use the following coding for the cosine pulse $f(t) = p(t)$, defined by equation (3.82) to compute the solution to equations (3.73) to (3.75):

```
C...
C... COSINE PULSE FUNCTION
W=5.
PI=3.1415927
IF(TIME.LT.0.)T(1)=0.
IF((W*TIME).GT.(PI))T(1)=0.
IF(((W*TIME).GE.(0.0)).AND.((W*TIME).LE.(PI/2.)))
1 T(1)=1.-COS(W*TIME)
IF(((W*TIME).GT.(PI/2.)).AND.((W*TIME).LE.(PI)))
1 T(1)=1.+COS(W*TIME)
TT(1)=0.
```

This coding will appear in subroutine DERV of Program 3.7. Also, $g(x) = 0$ will be used in subroutine INITAL. Perform the spatial differentiation (calculation of T_x) in equation (3.73) using

- (a) Subroutine DSS002
- (b) Subroutine DSS012
- (c) Subroutine DSS020

In each case, compare the NUMOL solution with the exact solution, equation (3.83), programmed in subroutine PRINT using the coding

```
C...
C... COMPUTE THE EXACT SOLUTION FOR PRINTING AND PLOTTING
L=1.
V=1.
W=5.
PI=3.1415927
TXV=TIME-L/V
IF(TXV.LT.0.)TE=0.
IF((W*TXV).GT.(PI))TE=0.
IF(((W*TXV).GE.(0.0)).AND.((W*TXV).LE.(PI/2.)))TE=1.-COS(W*TXV)
IF(((W*TXV).GT.(PI/2.)).AND.((W*TXV).LE.(PI)))TE=1.+COS(W*TXV)
```

- (3.25) Consider the extension of the advection equation (3.72) to include *axial diffusion or dispersion*

$$\frac{\partial T}{\partial t} = -v \frac{\partial T}{\partial x} + D \frac{\partial^2 T}{\partial x^2} \quad (3.25a)$$

Equation (3.25a) is a *convective diffusion equation* or *parabolic-hyperbolic PDE*; D is an *axial diffusivity* or *dispersion coefficient*.

Since equation (3.25a) is second-order in x , it requires another boundary condition [in addition to boundary condition (3.74)]:

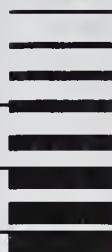
$$\frac{\partial T(L, t)}{\partial x} = 0$$

or in the subscript notation

$$T_x(L, t) = 0 \quad (3.25b)$$

Equation (3.25b) specifies only convection takes place at the exit of the tube in Figure 3.1 (note that equation (3.25b) follows from equation (3.25a) by dropping all but the convective terms, and then letting $x = L$). Compute a NUMOL solution to equations (3.25a), (3.74), (3.25b), and (3.75) for $f(t) = 1$, $g(x) = 0$, $v = D = L = 1$. Use DSS020 to calculate the convective derivative $\partial T / \partial x$ in equation (3.25a) and DSS044 to calculate the diffusion derivative $\partial^2 T / \partial x^2$. Since equation (3.25a) reduces to Fourier's second law for $v = 0$ [equation (1.4) or (1.10)], and the advection equation for $D = 0$ [equation (3.72) or (3.73)], does the NUMOL solution appear to have the properties you would expect (i.e., properties of the solutions for both Fourier's second law and the advection equation)?

4



Initial-Value Integration

In Chapter 3, we considered the first principal step in developing a NUMOL solution to a PDE problem, the calculation of the spatial derivatives. In this chapter, we shall consider the second principal step, the integration of the system of initial-value ODEs. The error analysis in Section 1.6 indicated that we have to be concerned with both the *accuracy* and *stability* of the ODE integration. The accuracy requirement is handled more or less automatically by any quality ODE integrator (e.g., RKF45), which will adjust the integration step size to meet the user-specified error tolerance (via arguments ABSERR and RELERR of subroutine RKF45). Also, quality ODE integrators employ higher-order integration formulas to achieve the required accuracy (RKF45 gives a fifth-order correct result, or in other words, the integration error is $O(\Delta t^5)$, where t is the initial-value independent variable and Δt is the integration step size). We therefore assume that the accuracy requirement is met, and concentrate on the stability requirement (which, ultimately, will determine the selection of an ODE integrator). First, we consider the stability properties of a system of linear, constant-coefficient ODEs.

4.1 The ODE Eigenvalue Problem

The preceding examples indicate that we typically must integrate rather large sets of ODEs (Program 2.7 involved a 21×21 -point spatial grid requiring the solution of 442 ODEs). However, we shall consider just two ODEs for simplicity (fortunately, the conclusions we shall come to also apply to a set of ODEs of any size)

$$\frac{dy_1}{dt} = a_{11}y_1 + a_{12}y_2 \quad (4.1)$$

$$\frac{dy_2}{dt} = a_{21}y_1 + a_{22}y_2 \quad (4.2)$$

Equations (4.1) and (4.2) are linear ODEs with constant coefficients (the constant coefficients are a_{11} , a_{12} , a_{21} , and a_{22}). Equations of this type have *exponential solutions*

$$y_1(t) = C_1 e^{\lambda t} \quad y_2(t) = C_2 e^{\lambda t} \quad (4.3) \quad (4.4)$$

Substitution of equations (4.3) and (4.4) in (4.1) and (4.2) gives

$$\lambda C_1 e^{\lambda t} = a_{11}C_1 e^{\lambda t} + a_{12}C_2 e^{\lambda t} \quad (4.5)$$

$$\lambda C_2 e^{\lambda t} = a_{21}C_1 e^{\lambda t} + a_{22}C_2 e^{\lambda t} \quad (4.6)$$

After canceling the common factor $e^{\lambda t}$ (this cancellation of the exponential term is why the exponential solution works), and rearranging, we arrive at two *homogeneous, linear, algebraic equations* for the two constants C_1 and C_2 :

$$(\lambda - a_{11})C_1 - a_{12}C_2 = 0 \quad (4.7)$$

$$-a_{21}C_1 + (\lambda - a_{22})C_2 = 0 \quad (4.8)$$

Equations (4.7) and (4.8) will have a nontrivial solution ($C_1 = 0$ and $C_2 = 0$ is trivial) if and only if the determinant of the coefficient matrix is zero. Thus

$$(\lambda - a_{11})(\lambda - a_{22}) - a_{12}a_{21} = 0$$

or

$$\lambda^2 - (a_{11} + a_{22})\lambda + a_{11}a_{22} - a_{12}a_{21} = 0 \quad (4.9)$$

Equation (4.9) is the *characteristic equation* for equations (4.1) and (4.2). Note that it is a second-order polynomial in λ . If we were considering a system of n first-order ODEs, the characteristic equation would be an n th-order polynomial.

Equation (4.9) can be factored by the well-known formula for the roots of a quadratic equation to give two roots, λ_1 and λ_2 , which are the

eigenvalues of equations (4.1) and (4.2). The *general solution* to equations (4.1) and (4.2) is therefore

$$y_1 = C_{11}e^{\lambda_1 t} + C_{12}e^{\lambda_2 t} \quad (4.10)$$

$$\lambda_2 = C_{21}e^{\lambda_1 t} + C_{22}e^{\lambda_2 t} \quad (4.11)$$

where C_{11} to C_{22} are constants that can be determined from the two (unspecified) initial conditions of equations (4.1) and (4.2) [in combination with the characteristic equation (4.9) written for each of the eigenvalues, λ_1 and λ_2]. However, for the purpose of considering the stability of ODEs, we do not need to have the values of these constants. Note that equations (4.10) and (4.11) are *linear combinations* of the assumed solutions, equations (4.3) and (4.4). This *superposition* of solutions is possible since equations (4.1) and (4.2) are linear.

For each eigenvalue (λ_1 and λ_2), two constants appear in the general solution [equations (4.10) and (4.11)]; for example, for λ_1 , the two constants are C_{11} and C_{21} , where the first subscript refers to the original constants, C_1 and C_2 in equations (4.3) and (4.4) and the second subscript refers to the first eigenvalue, λ_1 . The constants C_{11} and C_{21} are termed the *eigenvector* for C_{11} ; similarly, C_{12} and C_{22} are the eigenvector for λ_2 .

Finally, in order for the solution, equations (4.10) and (4.11) to be stable, the *eigenvalues*, λ_1 and λ_2 , *must have negative real parts* (we state this stability criterion in terms of real parts since the eigenvalues can be complex). More generally, *for a system of n linear, constant-coefficient ODEs to be stable, the n eigenvalues must be in the left half of the complex plane*. If this condition is not met for all of the eigenvalues, the associated exponentials will grow with t , thereby making the system, e.g., $y_1(t)$ and $y_2(t)$, unstable. Here we are talking about the stability of the original ODE problem, such as equations (4.1) and (4.2), and not the stability of the numerical integration of the ODEs, which is yet to be discussed.

As we just concluded, all of the concepts we have discussed for two linear ODEs with constant coefficients, e.g., equations (4.1) and (4.2), apply as well to a general system of n linear, constant-coefficient ODEs. A system of linear ODEs can be written conveniently in the general matrix form:

$$d\bar{y}/dt = \bar{A}\bar{y} \quad (4.12)$$

where for the 2×2 system of equations (4.1) and (4.2):

$$d\bar{y}/dt = \begin{bmatrix} dy_1/dt \\ dy_2/dt \end{bmatrix} \quad \bar{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \bar{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

The generalization of these matrices to a system of n first-order ODEs, and the solution procedure discussed previously, is straightforward. An exponential solution to equation (4.12) is assumed:

$$\bar{y}(t) = \bar{C} e^{\lambda t} \quad (4.13)$$

Substitution of equation (4.13) in equation (4.12) leads to a system of linear, homogeneous algebraic equations after cancellation of the scalar $e^{\lambda t}$; equations (4.7) and (4.8) are the 2×2 special case of these algebraic equations with

$$\bar{C} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$$

These algebraic equations can be written in general matrix notation as

$$(\bar{A} - \lambda \bar{I}) \bar{C} = \bar{0} \quad (4.14)$$

where, for this 2×2 system:

$$\bar{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{the identity matrix})$$

Equation (4.14) is the *general linear algebraic eigenvalue problem*. Since it is a system of homogeneous, linear algebraic equations, it will have non-trivial solutions ($\bar{C} \neq \bar{0}$) if and only if the determinant of the coefficient matrix is zero, i.e.

$$\det(\bar{A} - \lambda \bar{I}) = 0 \quad (4.15)$$

Equation (4.15) is the characteristic equation for equation (4.12); it is an n th-order polynomial that defines the n eigenvalues, $\lambda_1, \lambda_2, \dots, \lambda_n$ [equation (4.9) is a special case of equation (4.15)]. Again, for a system of n linear, *stable*, first-order differential equations, *all n eigenvalues given by equation (4.15) must have negative real parts*.

Finally, we can begin to relate the preceding ideas to the selection of an ODE integrator. In particular, two properties of the coefficient matrix \bar{A} are major considerations in the selection of an integrator:

- (1) The *separation in the eigenvalues* of \bar{A} [the eigenvalues are defined by equation (4.15)] determines the *stiffness* of the ODE problem. If the eigenvalues vary widely in their magnitude (by many orders of magnitude), then a stiff ODE integrator must be used in an NUMOL code. This point will be illustrated later in terms of some numerical examples.

(2) The *structure* of \bar{A} will determine how the linear algebra in the ODE integrator will be performed, which in turn determines the efficiency of the integrator. For example, \bar{A} may be nearly *full or dense* (most of its elements are nonzero), *banded* (its elements occur mostly along diagonals concentrated around the main diagonal), or *sparse* (its elements are mostly zeros, and the nonzero elements are rather randomly positioned throughout the matrix). In fact, we shall later consider a series of ODE integrators that accommodate these various ODE structures.

All of the preceding discussion is based on linear ODEs [equation (4.12)], but as we have seen in the preceding examples (e.g., Programs 2.5 and 2.8), nonlinear ODEs typically occur in developing an NUMOL solution. We therefore consider how the preceding ideas can be extended to nonlinear ODEs.

4.2 Stability of Nonlinear ODEs

Unfortunately, a general stability analysis of a system of nonlinear ODEs is not possible (we don't know how to do such an analysis). About the best we can do is to apply some of the ideas for linear ODEs in Section 4.1 to nonlinear ODEs, realizing that this will involve approximations; we hope, however, that the conclusions we reach regarding linear ODEs will generally be valid and useful.

We can start with the general system of nonlinear ODEs, equations (1.33) to (1.36):

$$\begin{aligned} dy_1/dt &= f_1(y_1, y_2, \dots, y_n, t) \\ dy_2/dt &= f_2(y_1, y_2, \dots, y_n, t) \\ &\quad \cdot \quad \cdot \\ &\quad \cdot \quad \cdot \\ &\quad \cdot \quad \cdot \\ dy_n/dt &= f_n(y_1, y_2, \dots, y_n, t) \end{aligned} \tag{4.16}$$

$$y_1(t_0) = y_{1,0}, \quad y_2(t_0) = y_{2,0}, \dots, \quad y_n(t_0) = y_{n,0} \tag{4.17}$$

or in matrix form

$$d\bar{y}/dt = \bar{f}(\bar{y}, t) \tag{4.18}$$

$$\bar{y}(t_0) = \bar{y}_0 \tag{4.19}$$

where

$$\begin{aligned}\bar{y} &= [y_1 \quad y_2 \quad \dots \quad y_n]^T \\ \bar{f} &= [f_1 \quad f_2 \quad \dots \quad f_n]^T \\ \bar{y}_0 &= [y_{1,0} \quad y_{2,0} \quad \dots \quad y_{n,0}]^T\end{aligned}$$

If \bar{f} in equation (4.18) is expanded in a Taylor series around the point (\bar{y}_s, t_s) , and then truncated after the linear terms

$$d\bar{y}/dt = \bar{f}(\bar{y}_s, t_s) + \bar{J}(\bar{y}_s, t_s)(\bar{y} - \bar{y}_s) \quad (4.20)$$

where \bar{y}_s is the dependent variable vector \bar{y} at a particular value of t , t_s ; \bar{J} is the ODE *Jacobian matrix*

$$\bar{J} = \begin{bmatrix} f_{11} & f_{12} & \cdot & f_{n1} \\ f_{21} & f_{22} & \cdot & f_{n2} \\ \cdot & \cdot & \cdot & \cdot \\ f_{n1} & f_{n2} & \cdot & f_{nn} \end{bmatrix} \quad (4.21)$$

where $f_{ij} = \partial f_i / \partial y_j$; $i = 1, 2, \dots, n$; $j = 1, 2, \dots, n$. Thus, the Jacobian matrix is the $n \times n$ matrix of all possible first-order derivatives of f_1 to f_n with respect to the dependent (solution) vector, y_1 to y_n . This matrix plays a fundamental role in much of scientific numerical computing.

We can note the following important properties of \bar{J} :

(1) \bar{J} is a constant matrix for linear ODEs (all of its elements are constants). This can be seen by applying (4.20) to equations (4.1) and (4.2); for example, $f_{11} = \partial f_1 / \partial y_1 = \partial(a_{11}y_1 + a_{12}y_2) / \partial y_1 = a_{11}$. Continuing this calculation of the other three elements of \bar{J} , $f_{12} = a_{12}$, $f_{21} = a_{21}$, $f_{22} = a_{22}$.

(2) \bar{J} is a function of \bar{y} for nonlinear ODEs (the elements of \bar{J} are functions of \bar{y}). Thus, \bar{J} can be evaluated at a particular \bar{y} , e.g., $\bar{J}(\bar{y}_s, t_s)$, as in equation (4.20).

(3) If \bar{J} is evaluated as in (2), then equation (4.20) is linear in \bar{y} since $\bar{f}(\bar{y}_s, t_s)$ and $-\bar{J}(\bar{y}_s, t_s)\bar{y}_s$ are just constant vectors. Thus, except for these constant vectors, which have no effect on ODE stability, equations (4.12) and (4.20) are essentially of the same form (linear in \bar{y}). Therefore, \bar{J} in equation (4.20) plays the same role as \bar{A} in equations (4.12) to (4.15), and we can talk about the eigenvalues and eigenvectors of \bar{J} just as well as for \bar{A} . However, the numerical values of the eigenvalues and eigenvectors for \bar{J} will depend on (\bar{y}_s, t_s) since \bar{J} itself depends on (\bar{y}_s, t_s) [the point of expansion of the Taylor series in equation (4.20)]; in other words, the eigenvalues and eigenvectors will not be constant as they were for equation (4.12). In fact, we shall evaluate the eigenvalues of \bar{J} in

subsequent NUMOL solutions as the solutions evolve in t to see how the eigenvalues change along the solutions.

(4) Since \bar{J} plays the same role for nonlinear ODEs [equation (4.18)] as \bar{A} for linear ODEs [equation (4.12)], we shall also be concerned about the separation of the eigenvalues and the structure of \bar{J} in selecting an ODE integrator.

Now that we have covered some of the mathematical preliminaries for the stability of linear and nonlinear ODEs, we can proceed with an analysis of the stability properties of some ODE integrators.

4.3 Stability of the Explicit Euler Method

The explicit Euler method is the simplest of the ODE algorithms. When applied to equation (4.18), it is

$$\bar{y}_{n+1} = \bar{y}_n + (d\bar{y}_n/dt)\Delta t = \bar{y}_n + \bar{f}(\bar{y}_n, t_n)\Delta t \quad (4.22)$$

Equation (4.22) indicates how the ODE solution at $n + 1$ (\bar{y}_{n+1}) is computed from the solution at n (\bar{y}_n) using an integration step Δt . We shall not discuss the explicit Euler method in any detail since it is covered in detail elsewhere [Kahaner et al. (4.1), Lapidus and Seinfeld (4.2), Ortega and Poole (4.3), Shampine and Gordon (4.4)], but rather, just point out that it (a) is based on the Taylor series truncated after the linear term in Δt and therefore gives an ODE solution with accuracy $O(\Delta t)$ and (b) computes the next solution point by a projection from t_n to $t_n + \Delta t$ along the tangent to the solution at t_n ; this projection along the straight-line tangent accounts for its accuracy of $O(\Delta t)$ (linear dependence of the error on Δt).

Before we proceed, we should note a small change in notation. The points along an ODE solution will now be denoted with the index n [see equation (4.22)], and therefore the total number of ODEs will be denoted with N [not n as in equation (4.16)]. This choice of the t index was made because we have already used i for the spatial grid index in PDEs, and several other possibilities such as j and k will be used for other purposes later. To reiterate, n is the grid index for t , and i is the grid index for x (in PDEs); N is the total number of ODEs (which is consistent with the Fortran variable N used for this purpose in previous programs).

Also, we will again investigate the stability of the explicit Euler method using just two ODEs, with the understanding that the conclusions we reach will generally apply to a system of N ODEs. If we apply equation

(4.22) to equations (4.1) and (4.2), we obtain

$$y_{1,n+1} = y_{1,n} + (a_{11}y_{1,n} + a_{12}y_{2,n})\Delta t \quad (4.23)$$

$$y_{2,n+1} = y_{2,n} + (a_{21}y_{1,n} + a_{22}y_{2,n})\Delta t \quad (4.24)$$

Equations (4.23) and (4.24) are two *linear, constant-coefficient difference equations*.

In analogy with differential equations [see equations (4.3) and (4.4)], difference equations (4.23) and (4.24) are assumed to have solutions of the form

$$y_{1,n} = C_1\beta^n, \quad y_{2,n} = C_2\beta^n \quad (4.25)$$

where C_1 , C_2 , and β are constants to be determined (these assumed solutions are analogous to exponential solutions assumed for differential equations, but n is now the independent variable rather than t). If equation (4.25) is substituted in equations (4.23) and (4.24), we obtain

$$C_1\beta^{n+1} = C_1\beta^n + (a_{11}C_1\beta^n + a_{12}C_2\beta^n)\Delta t$$

$$C_2\beta^{n+1} = C_2\beta^n + (a_{21}C_1\beta^n + a_{22}C_2\beta^n)\Delta t$$

Cancellation of the common factor β^n gives two *simultaneous, linear, homogeneous algebraic equations* for the constants C_1 and C_2 (note the analogy with the cancellation of the common exponential $e^{\lambda t}$ when solving ODEs):

$$((\beta - 1) - a_{11}\Delta t)C_1 - a_{12}\Delta t C_2 = 0 \quad (4.26)$$

$$-a_{21}\Delta t C_1 + ((\beta - 1) - a_{22}\Delta t)C_2 = 0 \quad (4.27)$$

Again, as with the case of ODEs [see equations (4.7) and (4.8)], equations (4.26) and (4.27) will have nontrivial solutions ($C_1 \neq 0$ and/or $C_2 \neq 0$) if and only if the determinant of the coefficient matrix is zero. This condition leads to the characteristic equation

$$((\beta - 1) - a_{11}\Delta t)((\beta - 1) - a_{22}\Delta t) - a_{12}a_{21}\Delta t^2 = 0$$

or

$$\begin{aligned} \beta^2 - ((a_{11} + a_{22})\Delta t + 2)\beta + ((a_{11}a_{22} - a_{12}a_{21})\Delta t^2 \\ + (a_{11} + a_{22})\Delta t + 1) = 0 \end{aligned} \quad (4.28)$$

Equation (4.28) is the *characteristic equation* (a second-order polynomial in β) that can be factored to obtain the *eigenvalues* of equations (4.23) and (4.24), β_1 and β_2 . The *general solution* to equations (4.23) and (4.24), in analogy with equations (4.10) and (4.11) for ODEs (4.1) and (4.2), is again

a *superposition of solutions*

$$y_{1,n} = C_{11}\beta_1^n + C_{12}\beta_2^n \quad (4.29)$$

$$y_{2,n} = C_{21}\beta_1^n + C_{22}\beta_2^n \quad (4.30)$$

where C_{11} to C_{22} are constants that can be determined from initial conditions for equations (4.23) and (4.24), e.g., $y_{1,0} = y_{10}$, $y_{2,0} = y_{20}$, where y_{10} and y_{20} are given constants.

With these solutions and eigenvalues, we can now attempt to determine under what conditions the Euler solution will be stable. The solution follows from equations (4.29) and (4.30), which indicate that as we step along the numerical solution, from n to $n + 1$, β is raised to higher powers of n . Thus, if the solution is to remain stable (not become unbounded with increasing n), we require

$$|\beta| \leq 1 \quad (4.31)$$

for *all of the difference equation eigenvalues* [β_1 and β_2 in the case of equations (4.23) and (4.24)]. Inequality (4.31) is the *stability criterion* for the explicit Euler integration of simultaneous ODEs.

Finally, what is the implication of inequality (4.31) for the Euler solution of the two simultaneous ODEs we have been considering, equations (4.1) and (4.2)? The eigenvalues of the ODEs, λ_1 and λ_2 , are given by characteristic equation (4.9), while the eigenvalues of the difference equations, β_1 and β_2 , are given by characteristic equation (4.28). Both these characteristic equations have coefficients that depend on the constants of the original ODEs and difference equations, a_{11} to a_{22} ; additionally, Δt appears in equation (4.28). Thus, we would expect that β_1 and β_2 are related to λ_1 and λ_2 through a_{11} to a_{22} plus Δt . However, this relationship is rather complicated.

Therefore, we will consider a special case of equations (4.1) and (4.2) with $a_{11} = a_{22} = -a$, $a_{12} = a_{21} = b$ (a and b are constants), which leads to real eigenvalues that are simply related to a and b . From equation (4.9), we obtain

$$\lambda^2 + 2a\lambda + (a^2 - b^2) = 0$$

and, from the quadratic formula

$$\lambda_{1,2} = \frac{-2a + (\text{or } -)\sqrt{4a^2 - 4(a^2 - b^2)}}{2}$$

$$\lambda_1 = -(a - b), \quad \lambda_2 = -(a + b) \quad (4.32) \quad (4.33)$$

Also, from equation (4.28),

$$\beta^2 + 2(a \Delta t - 1)\beta + (a^2 - b^2)\Delta t^2 - 2a \Delta t + 1 = 0$$

and from the quadratic formula

$$\beta_1 = 1 - (a - b)\Delta t, \quad \beta_2 = 1 - (a + b)\Delta t$$

or from equations (4.32) and (4.33)

$$\beta_1 = 1 + \lambda_1 \Delta t, \quad \beta_2 = 1 + \lambda_2 \Delta t \quad (4.34) \quad (4.35)$$

Finally, from stability criterion (4.31) applied to equations (4.34) and (4.35), for the Euler solution to remain stable, we require

$$|\beta_1| = |1 + \lambda_1 \Delta t| \leq 1, \quad |1 + \lambda_2 \Delta t| \leq 1$$

or

$$|\lambda \Delta t| \leq 2, \quad |\lambda_2 \Delta t| \leq 2 \quad (4.36) \quad (4.37)$$

In other words, for the Euler integration to remain stable, we require that the *absolute value of the product of each ODE eigenvalue and the integration step be less than or equal to 2*. This is actually a general result for the explicit Euler integration of N ODEs, including the case of complex eigenvalues [for which the absolute values in equations (4.36) and (4.37) should be interpreted for complex numbers]. In other words, *the explicit Euler integration is stable for a system of N first-order, linear, constant-coefficient ODEs if $\lambda_i \Delta t$, $i = 1, 2, \dots, N$ fall within a unit circle centered at the point $(-1, 0j)$ ($j = \sqrt{-1}$) in the complex plane, for all i* . This circle is termed the *stability region* for the explicit Euler method.

We could now ask: What are the practical implications of this result, if any? As we shall see in subsequent sections, this *stability limit* of the explicit Euler method can be a major consideration in the selection of an ODE integrator. To illustrate why this may be the case, consider the special case $a = 500000.5$, $b = 499999.5$ (admittedly a contrived example, but not far removed from practical examples in science and engineering). From equations (4.32) and (4.33),

$$\lambda_1 = -(500000.5 - 499999.5) = -1,$$

$$\lambda_2 = -(500000.5 + 499999.5) = -1,000,000$$

For these eigenvalues, equations (4.10) and (4.11) become

$$y_1 = C_{11}e^{-1t} + C_{12}e^{-1,000,000t} \quad (4.38)$$

$$y_2 = C_{21}e^{-1t} + C_{22}e^{-1,000,000t} \quad (4.39)$$

Thus, the two ODEs are stable (both eigenvalues are real and negative or, in other words, are in the left half of the complex plane), but the eigenvalues are widely separated. This separation has an important implication for the numerical integration of equations (4.1) and (4.2) and the selection of an ODE integrator.

First, we note that the time scale for the problem is approximately $0 \leq t \leq 10$ since for $t = 10$, the first exponential in equations (4.38) and (4.39) has decayed to the small value $e^{-1(10)} = 0.0000454$ (and, of course, the second exponential is far smaller); from this example, we see that, in general, the *smallest eigenvalue determines the ODE time scale*.

However, if we use the explicit Euler method to numerically integrate equations (4.1) and (4.2), the second eigenvalue, $-1,000,000$, determines the maximum step size for stability of the numerical calculation according to inequality (4.37)

$$| -10^6 \Delta t | \leq 2$$

or $\Delta t \leq 2/10^6$. Thus, we will step along the solution from $t = 0$ to $t = 10$ using a step size of no more than $2/10^6$, which will require

$$\frac{10}{2/10^6} = 5 \times 10^6$$

steps; this is a very large calculation for two linear ODEs! If you are not convinced that this is a large-scale calculation (using, for example, the argument that today we have very fast computers), then let $\lambda_2 = -10^9$ or -10^{12} for which 5×10^9 or 5×10^{12} steps would be required; the point is that if the ODE problem is sufficiently stiff (the eigenvalues are sufficiently separated), the *number of integration steps by the explicit Euler method to maintain stability becomes prohibitive*.

We can conclude from this example that (a) the smallest eigenvalue defines the problem time scale, e.g., $\lambda_1 = -1$ defines the time scale for equations (4.1) and (4.2) as approximately $0 \leq t \leq 10$, while (b) the largest eigenvalue defines the maximum step size to maintain stability, e.g., $\lambda_2 = -10^6$ defines the maximum step size as $\Delta t = 2/10^6 = 2 \times 10^{-6}$. It is this combination of a relatively large problem time scale and relatively small maximum integration step size that requires a large number of steps in computing a numerical solution by the explicit Euler method. This is a *problem of stability and not accuracy*; the distinction is often confused. For example, we might conclude that we could take much larger steps (than $\Delta t = 2 \times 10^{-6}$) if we used one of the higher-order ODE integration methods such as the fourth- and fifth-order methods in RKF45. However, this is not the case. For example, the stability criterion for the fourth-order

Runge–Kutta method is

$$|\lambda \Delta t| < 2.785 \quad (4.40)$$

so that the stability interval (2 for Euler's method, 2.785 for fourth-order Runge–Kutta) is only slightly increased by using a higher-order method. Note also that this small gain in the stability interval is achieved at a high price, namely, four derivative evaluations per step (four calls to subroutine DERV per step) for fourth-order Runge–Kutta vs. one derivative evaluation per step for Euler; in effect, the greater accuracy of the higher-order method is effectively lost because of the stability constraint. The stability region of the fourth-order explicit Runge–Kutta (RK) algorithm is indicated in Figure 4.1. Note that since λ can be complex, the horizontal and vertical axes are $\text{Re}(\lambda \Delta t)$ and $\text{Im}(\lambda \Delta t)$, respectively. For the explicit Euler method (which is the first-order Runge–Kutta method), the stability region is a circle of unit radius centered at $(-1, 0j)$, as defined by stability criterion (4.37). The slightly larger stability region of the fourth-order RK method is evident [note, in particular, the point $-2.785 + 0j$ given by inequality (4.40)]

Thus, all of the numerical integration methods we have studied so far have very limited stability intervals and therefore are not effective for stiff ODE problems. We might then ask: What is the property of these algorithms that gives them such limited stability intervals? The answer in general is that they are *explicit methods*; this means that in using the

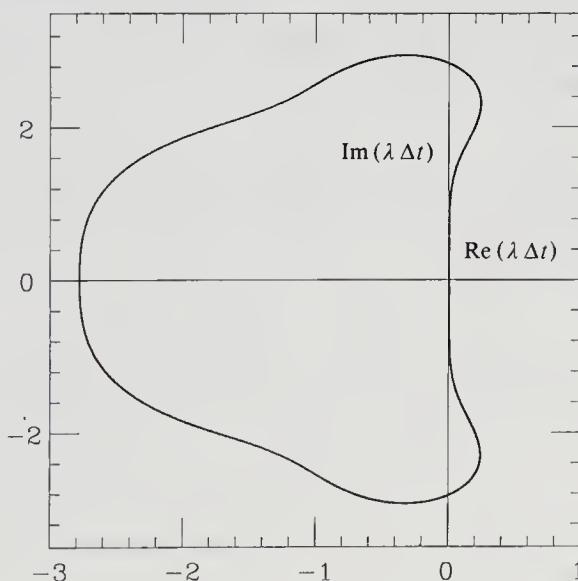


Figure 4.1 Stability region of the fourth-order explicit Runge–Kutta algorithm (the interior of the closed contour is the stable region).

stepping formula to go from y_n to y_{n+1} , such as equation (4.22) for the explicit Euler method, *only the solution at y_n was required, and therefore y_{n+1} could be calculated explicitly*. This is a convenient property for making the calculation of the solution at the advanced point, y_{n+1} , relatively easy, but the price we pay for this convenience is limited stability. Again, though, keep in mind that *we pay this price only for stiff problems*. For nonstiff problems (for which the ODE eigenvalues are separated by no more than a factor of 1000), *explicit algorithms work very well*, and should be used.

We could then ask: What do we do if we have a stiff problem and cannot use an explicit algorithm? The answer in general is that *we should use an implicit algorithm*, and that is the point of the next few sections of this chapter.

4.4 Stability of the Implicit Euler Method

The implicit Euler method is

$$\bar{y}_{n+1} = \bar{y}_n + (d\bar{y}_{n+1}/dt)\Delta t = \bar{y}_n + \tilde{f}(\bar{y}_{n+1}, t_{n+1})\Delta t \quad (4.41)$$

which is similar to equation (4.22) for the explicit Euler method; in fact, the only difference is that the derivative vector, $d\bar{y}/dt$, is evaluated at the advanced point $n+1$, rather than at the base point n , as in the explicit Euler method. While this may appear to be a minor difference, it is, in fact, rather profound, as we shall see.

If equation (4.41) is applied to the 2×2 linear constant-coefficient ODE system, equations (4.1) and (4.2),

$$y_{1,n+1} = y_{1,n} + (a_{11}y_{1,n+1} + a_{12}y_{2,n+1}) \Delta t \quad (4.42)$$

$$y_{2,n+1} = y_{2,n} + (a_{21}y_{1,n+1} + a_{22}y_{2,n+1}) \Delta t \quad (4.43)$$

Again, we assume solutions

$$y_{1,n} = C_1\beta^n, \quad y_{2,n} = C_2\beta^n \quad (4.44)$$

which, when substituted in difference equations (4.42) and (4.43), give

$$C_1\beta^{n+1} = C_1\beta^n + (a_{11}C_1\beta^{n+1} + a_{12}C_2\beta^{n+1}) \Delta t$$

$$C_2\beta^{n+1} = C_2\beta^n + (a_{21}C_1\beta^{n+1} + a_{22}C_2\beta^{n+1}) \Delta t$$

Cancellation of the factor β^n gives two simultaneous homogeneous linear algebraic equations:

$$(1 + (a_{11}\Delta t - 1)\beta)C_1 + a_{12}\Delta t\beta C_2 = 0 \quad (4.45)$$

$$a_{21}\Delta t\beta C_1 + (1 + (a_{22}\Delta t - 1)\beta)C_2 = 0 \quad (4.46)$$

As before, equations (4.45) and (4.46) will have nontrivial solutions if and only if the determinant of the coefficient matrix is zero, which leads to the characteristic equation

$$(1 + (a_{11} \Delta t - 1)\beta)(1 + (a_{22} \Delta t - 1)\beta) - a_{12}a_{21} \Delta t^2 \beta^2 = 0$$

or

$$\begin{aligned} & ((a_{11} \Delta t - 1)(a_{22} \Delta t - 1) - a_{12}a_{21} \Delta t^2)\beta^2 + ((a_{11} \Delta t - 1) \\ & + (a_{22} \Delta t - 1))\beta + 1 = 0 \end{aligned} \quad (4.47)$$

Equation (4.47) can be factored by the quadratic formula to give the eigenvalues of the implicit Euler solution. However, as before, the result is rather complicated, and therefore we will consider the special case: $a_{11} = a_{22} = -a$, $a_{12} = a_{21} = b$, for which equation (4.47) becomes

$$((a \Delta t + 1)^2 - b^2 \Delta t^2)\beta^2 - 2(a \Delta t + 1)\beta + 1 = 0 \quad (4.48)$$

Equation (4.48) can easily be factored by the quadratic formula

$$\beta_1 = \frac{1}{1 - (-a + b) \Delta t} = \frac{1}{1 - \lambda_1 \Delta t} \quad (4.49)$$

$$\beta_2 = \frac{1}{1 - (-a - b) \Delta t} = \frac{1}{1 - \lambda_2 \Delta t} \quad (4.50)$$

Again, as with the explicit Euler method, the general solutions to difference equations (4.42) and (4.43) are

$$y_{1,n} = C_{11}\beta_1^n + C_{12}\beta_2^n \quad (4.51)$$

$$y_{2,n} = C_{21}\beta_1^n + C_{22}\beta_2^n \quad (4.52)$$

Then, for the solution to remain stable, we require that inequality (4.31) be satisfied by all of the difference equation eigenvalues; for the present 2×2 ODE system, this condition is $|\beta_1| \leq 1$ and $|\beta_2| \leq 1$. However, since $\text{Re}(\lambda_1 \Delta t) < 0$ and $\text{Re}(\lambda_2 \Delta t) < 0$ for a stable ODE system (Δt is positive), equations (4.51) and (4.52) indicate that inequality (4.31) is satisfied by both eigenvalues for all values of Δt . Thus, for this 2×2 system of linear ODEs, equations (4.1) and (4.2), the implicit Euler method is unconditionally stable (the stability region is the entire left half of the complex plane). This conclusion will also be true for a system of N ODEs, with real, repeated, or complex eigenvalues provided all of the eigenvalues have negative real parts. In other words, the separation of eigenvalues does not cause a problem with stability when using the implicit Euler method.

To reiterate, for stiff ODEs (with widely separated eigenvalues), the implicit Euler method can be used to compute a numerical solution with the step size unconstrained by stability (only the accuracy will limit the

step size). This conclusion must be qualified, however, in the following ways:

(1) It is valid only for linear, constant-coefficient ODEs (but we hope that the implicit Euler method is also effective in the solution of stiff nonlinear ODEs, which, experience has indicated, is the case).

(2) For the more general nonlinear case, application of the implicit Euler method gives (for two ODEs):

$$y_{1,n+1} = y_{1,n} + f_1(y_{1,n+1}, y_{2,n+1}, t_{n+1}) \Delta t \quad (4.53)$$

$$y_{2,n+1} = y_{2,n} + f_2(y_{1,n+1}, y_{2,n+1}, t_{n+1}) \Delta t \quad (4.54)$$

Note that the solution at the advanced point, $y_{1,n+1}$, $y_{2,n+1}$, appears implicitly in equations (4.53) and (4.54), and since, in general, f_1 and f_2 are nonlinear, we will have to apply a *root-finding* method to equations (4.53) and (4.54) to obtain $y_{1,n+1}$ and $y_{2,n+1}$; generally, this will be *Newton's method*. Note also that in the case of linear ODEs [equations (4.1) and (4.2)], it is possible to solve for $y_{1,n+1}$ and $y_{2,n+1}$ algebraically, as we did in obtaining equations (4.51) to (4.52).

(3) Even when we solve equations (4.53) and (4.54) numerically for $y_{1,n+1}$ and $y_{2,n+1}$, the solution will be relatively inaccurate because of the accuracy limitation of the Euler method (it is only first-order correct, in either its explicit or implicit form). Thus, it would be desirable to have an integration algorithm with good *accuracy and stability*. We shall now consider such an algorithm.

4.5 The BDF Methods

The preceding stepping formula for the implicit Euler method, equation (4.41), can be generalized to

$$y_{n+1} = \sum_{l=0}^{q-1} \alpha_l y_{n-l} + \Delta t \beta_0 dy_{n+1}/dt \quad (4.55)$$

[For simplicity, equation (4.55) is written in scalar form (for a single ODE), but it can also be written in matrix form for a system of N ODEs]; q is the order of the method, and α_l and β_0 are constants for a particular order. Note that for $q = 1$, with $\alpha_0 = \beta_0 = 1$, equation (4.55) reduces to

$$y_{n+1} = y_n + \Delta t dy_{n+1}/dt$$

which is the implicit Euler method [cf. equation (4.41)].

Equation (4.55) is explicit in the solution y_{n-l} since it uses only the past values y_n , y_{n-1} , y_{n-2} , . . . , $y_{n-(q-1)}$, but is *implicit in the one derivative*

tive, dy_{n+1}/dt . Thus, equation (4.55) is termed a *backward differentiation formula* (BDF); it is this implicit derivative term that gives the BDF its good stability properties. Also, since the BDF uses more than one past value of the solution, it is called a *multistep* method, in contrast with the Runge–Kutta methods, which require only the base point value y_n to take the next step to y_{n+1} .

The requirement, now, in using equation (4.55) is to select the constants α_i and β_0 so that the resulting integration algorithm has *both good accuracy and stability properties*. This has been done by Gear (4.5), who assigned the values listed in Table 4.1 to these constants for orders 1 to 6 ($1 \leq q \leq 6$). Note that the coefficients in equation (4.55) are for orders 1 to 6. Thus, in using the $q = 6$ coefficients, the integration from y_n to y_{n+1} is approximated by a sixth-order polynomial (the first to sixth order terms of the Taylor series). This relatively high degree of accuracy is achieved by using $y_n, y_{n-1}, \dots, y_{n-5}$ as well as dy_{n+1}/dt . However, at the beginning of the solution, only the initial condition, y_0 , is available, so the calculation must start with the $q = 1$ formula (the implicit Euler method). Then, as additional points are computed along the solution, the higher-order BDFs can be used. In other words, the computer implementation of the BDFs must not only handle the implicit term dy_{n+1}/dt , and adjust the step size to achieve the required accuracy, but must also vary the order of the method, starting with $q = 1$ and eventually working up to the higher-order formulas. This approach is therefore a *variable-step, variable-order, implicit (stiff) method*.

Additionally, the past values of the solution must be stored for use in the multistep formulas, and they must be available at the current integration step size. This requires interpolation of past values to obtain the necessary values at the current integration step size. Therefore, the computer implementation of the BDF method is relatively complicated; fortunately, all of this has been done, and the BDF method is available in well-established computer codes, which will be discussed in the next section.

Table 4.1 Coefficients for the BDF Formulas

q	β_0	α_0	α_1	α_2	α_3	α_4	α_5
1	1	1					
2	$\frac{2}{3}$	1	$\frac{1}{3}$				
3	$\frac{6}{11}$	1	$\frac{6}{11}$	$\frac{1}{11}$			
4	$\frac{24}{50}$	1	$\frac{35}{50}$	$\frac{1}{5}$	$\frac{1}{50}$		
5	$\frac{120}{274}$	1	$\frac{225}{274}$	$\frac{85}{274}$	$\frac{15}{274}$	$\frac{1}{274}$	
6	$\frac{720}{1764}$	1	$\frac{1624}{1764}$	$\frac{735}{1764}$	$\frac{175}{1764}$	$\frac{21}{1764}$	$\frac{1}{1764}$

The coefficients in Table 4.1 were selected to achieve a compromise between accuracy and stability. This compromise is required because *for methods higher than second order, absolute stability is not possible*, i.e., the entire left half of the complex plane cannot be a stable region for methods above second order. This is demonstrated in Figures 4.2a and 4.2b, where the stability regions of the BDFs for the coefficients in Table 4.1 are plotted. Note that for the orders greater than 2, a portion of the left half of the complex plane is unstable, starting along the imaginary (vertical) axis. The third- and higher-order methods that do not have a stability region encompassing the entire left half plane are termed *stiffly stable* (as opposed to the first- and second-order methods, which are stable over the entire left-hand plane and are therefore *unconditionally or absolutely stable*). As the order increases to 6, more of the left plane is in the unstable region. The coefficients in Table 4.1 were selected so that the *negative real axis remains in the stable region* (real, negative eigenvalues of the ODEs will be handled with stability in the numerical integration, no matter what their separation or stiffness). Fortunately, for many physical problems, the ODE eigenvalues are real, and therefore the stability properties of the BDFs with the coefficients from Table 4.1 will be good. For highly oscillatory systems with eigenvalues along the imaginary axis, the coefficients of Table 4.1 will not give good stability, but revisions of the constants are available for this case.

As noted previously, a requirement of the BDF formulas [cf. equation (4.55)] that must be considered when these formulas are implemented in a computer code is the availability of the series of values $y_n, y_{n-1}, \dots, y_{n-l}$. At the beginning of the solution, only the initial value, y_0 , is available for computing y_1 , and all of the other past values required in equation (4.55) are generally unavailable unless we take $n = 0, q = 1$, i.e., we must start with a *first-order method*, which is the implicit Euler method. Then, as additional values of the dependent variable are computed, this *past history* can be used to include more terms in the sum of equation (4.55). This in effect means that we can begin to use the higher-order formulas (above first-order) once enough of the past history of the solution is known. In other words, the BDF methods are programmed so that they are both *variable-order and variable-step*; the integration step is changed to meet the user-specified error criterion, and the order is simultaneously varied once enough of the solution is available to also meet the error criterion (as expected, when the higher-order formulas can be used, larger steps are possible for a given accuracy in the solution). This feature of variable-order and variable-step makes the programming of the BDF methods relatively complicated as compared with the single-step Runge-Kutta methods. Fortunately, this additional complexity has been studied,

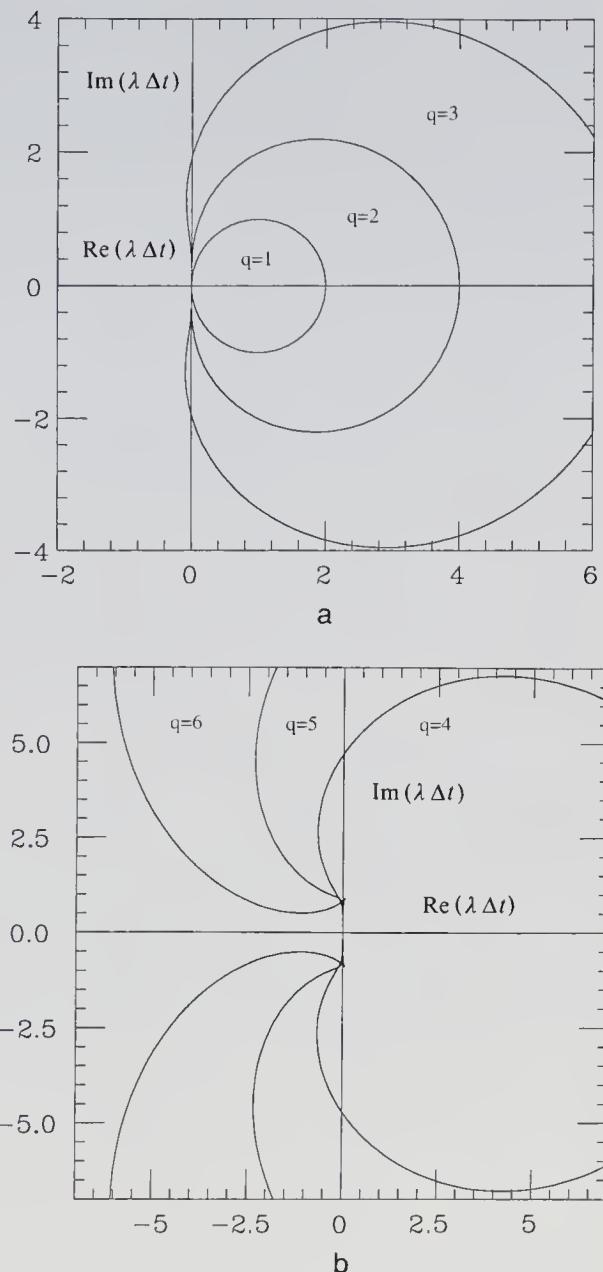


Figure 4.2 Stability regions for the BDFs of (a) orders 1 to 3 (the stable regions are outside the closed contours, or to the left of the open contours); (b) orders 4 to 6 (the stable regions are to the left of the open contours).¹

¹ Fortran programs for calculating the stability boundaries in Figures 4.1, 4.2a, and 4.2b are included with the software available from the author.

implemented, and tested in existing codes so that the user of the BDF methods generally does not have to become involved in the details of order changing as the solution evolves. However, the current order is available as an output from the BDF codes, as well as the current step size, so that the variable-order and variable-step size features can be monitored for any particular problem.

Finally, the implicit nature of the BDFs [due to the dy_{n+1}/dt term in equation (4.55)] requires some additional discussion. To illustrate this point, we can write the implicit Euler method, equation (4.41) or equation (4.55) with $q = 1$, as (again, in scalar form)

$$\begin{aligned} g(y_{n+1}) &= y_{n+1} - y_n - \Delta t \beta_0 dy_{n+1}/dt \\ &= y_{n+1} - y_n - \Delta t \beta_0 f(y_{n+1}, t_{n+1}) \end{aligned} \quad (4.56)$$

We now have to solve the equation

$$g(y_{n+1}) = 0 \quad (4.57)$$

for y_{n+1} . This amounts to root finding, and, generally, this is done by Newton's method, i.e.

$$y_{n+1}^{k+1} = y_{n+1}^k - D^{-1}g(y_{n+1}^k) \quad (4.58)$$

where k is an iteration counter and $D = dg/dy_{n+1}^k = I - \Delta t \beta_0 J$, $I = 1$, $J = df/dy_{n+1}^k$ [from equation (4.56)].

This may seem like a rather complicated way to write Newton's method applied to equation (4.57), but it was done this way so that equations (4.55), (4.56), and (4.57) could be applied directly to the case of N simultaneous ODEs; now I is the identity matrix and, again J is the Jacobian matrix of the ODE system [cf. equation (4.21)].

Equation (4.58) is not used in the form it is stated when N simultaneous ODEs are to be integrated by the BDFs. Rather, it is written in the alternative form

$$(I - \Delta t \beta_0 J)\Delta y_{n+1}^{k+1} = -g(y_{n+1}^k) \quad (4.59)$$

where $\Delta y_{n+1}^{k+1} = y_{n+1}^{k+1} - y_{n+1}^k$ is a vector of *Newton corrections*. Since the Jacobian matrix J is evaluated at y_{n+1}^k , equation (4.59) is a set of *linear algebraic equations for the N Newton corrections*, $\Delta y_{1,n+1}^{k+1}$ to $\Delta y_{N,n+1}^{k+1}$. This vector of corrections is therefore computed by conventional methods for linear algebraic equations, e.g., Gauss row reduction or an iterative method, starting with $k = 0$ and proceeding repetitively until some measure of convergence is achieved. The final vector at which convergence is achieved gives the dependent (solution) vector at the advanced point, $n + 1$, along the solution to the system of N ODEs (since $y_{n+1}^{k+1} = y_{n+1}^k + \Delta y_{n+1}^{k+1}$ for the iteration $k + 1$ for which convergence is

achieved). The iterative calculation requires an initial estimate of the solution, y_{n+1}^0 corresponding to $k = 0$. The usual procedure is to use the solution at the base point, $y_{n+1}^0 = y_n$.

Thus, the *underlying calculation in the computer implementation of the BDF method is linear algebra* (and this is generally true for all implicit integration methods). In fact, this linear algebra is the *major part of the computational effort*. Considering equation (4.59), the efficiency with which this linear algebra will be done is determined by the following factors:

- (1) The rate of convergence of the calculations, i.e., how many iterations must be performed starting with $k = 0$.
- (2) The number of times during the iterative solution of equation (4.59) the Jacobian matrix, is updated, i.e, the number of times J is evaluated at y_{n+1}^k . This is the *single most important factor in determining the total computational effort in using the BDF method* since the Jacobian matrix is $N \times N$, and therefore the matrix grows rapidly with N . (For a mere 100 ODEs, the full Jacobian matrix has $100 \times 100 = 10,000$ elements!) The trick in achieving computational efficiency is to *update the Jacobian matrix as infrequently as possible* to achieve convergence of the Newton iteration. Using a Jacobian matrix that is not completely current *does not introduce any additional error in the numerical solution as long as the Newton iteration converges*. Thus, the BDF integrators we consider subsequently are designed to update the Jacobian matrix only when required to achieve convergence; i.e., a Jacobian update is performed only when the Newton iteration appears not to be converging.
- (3) The number of nonzero elements of the Jacobian matrix.

Typically, in scientific applications, the *Jacobian matrix is not full*, i.e., many of the elements are zero; by taking advantage of the *Jacobian matrix structure*, very substantial savings in computer time can be gained in solving equation (4.59) repetitively. This point can be appreciated by the following example. If we are attempting to integrate a system of N ODEs with a bandwidth of M (all of the nonzero elements of the Jacobian matrix are in a band around the main diagonal with a maximum width of M elements), Gauss row reduction of the full matrix [to solve the N linear equations of equation (4.59)] would require N^3 arithmetic operations [each time equation (4.59) is solved along the ODE solution], while using a version of Gauss row reduction that exploits the banded structure of the Jacobian matrix requires NM^2 operations. Thus, the ratio of these two *operations counts* (full Jacobian matrix/banded Jacoabian matrix) is $N^3/NM^2 = (N^2/M^2)$. If $N = 1000$, $M = 30$ (a rather modest ODE problem), the ratio of computational effort is $(1000/30)^2 = 1111.1$; i.e., the use of a banded solver for equation (4.59) reduces the computational

effort by a factor of approximately 1/1000. This result would, of course, be even more impressive for larger N . Several of the integrators subsequently discussed are designed to exploit the structure of the Jacobian matrix.

(4) The evaluation of the partial derivatives in the Jacobian matrix [see equation (4.21)]. Since for a relatively complicated set of ODEs, calculation of all $N \times N$ partial derivatives by analytical differentiation is impractical (the derivatives are too many in number and too complicated), the partial derivatives are computed numerically; a *numerical Jacobian* is therefore almost always used in large ODE problems, and is typically computed by finite differences.

(5) The number of times $f(y_{n+1}^k, t_{n+1})$ is updated; this is termed the *number of function evaluations*. Actually, the number of Jacobian evaluations (updates) and number of function evaluations are related since the calculation of the numerical Jacobian also requires the evaluation of $f(y_{n+1}^k, t_{n+1})$.

The preceding discussion suggests that the linear algebra of equation (4.59) and the associated Jacobian and function evaluations are the major part of the computational effort in using the BDF methods, and this is, in fact, the case. The BDF integrators we will consider in subsequent sections actually report the *computational statistics* associated with the linear algebra, e.g., the number of Jacobian and function evaluations.

Finally, this discussion of the BDF methods probably gives the impression that this approach is very complicated, at least relative to the explicit methods considered earlier (RKF45). You may naturally ask whether all of this complexity is really necessary. The answer is “yes” for *stiff problems*. In other words, a stiff (implicit) integrator should not be used on a nonstiff problem (for which the eigenvalues are not widely separated) since the linear algebra is not required and is therefore essentially wasted effort. On the other hand, for a stiff problem, the use of a *BDF or other implicit integrator can reduce the computer run times by orders of magnitude*.

Fortunately, all of the preceding mathematics has been implemented in several quality, state-of-the-art subroutines that are easy to use. These subroutines were written by internationally recognized experts who gave special attention to transportability (the coding is in standard Fortran 77); the integrators have been applied to a broad spectrum of problems, and they are now so widely used that they can be considered international standards. Thus, the use of BDF methods is now rather routine for nonexperts, but still, some knowledge of the preceding mathematics is essential for the effective use of these integrators (at least the distinction between stiff and nonstiff ODEs, and the choice of implicit vs. explicit integrators should be appreciated). More detailed discussions of stiff ODEs, and the

theory of stiff integrators, are given in several books listed at the end of this chapter [e.g., Kahaner et al. (4.1), Gear (4.5)].

We now consider the use of several quality integrators for stiff and nonstiff ODEs, starting with SDRV2. We shall see that essentially all of the preceding software for nonstiff ODEs (based on RKF45) can be used with very minor changes (essentially, the calls to RKF45 are replaced with calls to the new integrators).

4.6 SDRV2

Subroutine SDRV2 is described in detail by Kahaner (4.1), so the discussion here will be principally in terms of its application to a previous problem, equations (1.21) to (1.24). SDRV2 is an implementation of the BDF methods discussed in Section 4.5, with options for stiff and nonstiff problems. The use of SDRV2 is illustrated in a main program, PRO4P1, which calls SDRV2 and subroutines INITAL, DERV and PRINT in Programs 1.3a to 1.3c. PRO4P1 is very similar to main program PRO1P3 (Program 1.3d), and essentially differs only in the call to the integrator (the call to RKF45 in PRO1P3 is replaced with a call to SDRV2 in PRO4P1); therefore, to save space, only this call, preceded by the definition of the parameters for SDRV2, is listed in Program 4.1.

The parameters of SDRV2 are described in detail by Kahaner (4.1). Therefore, only a few parameters are discussed here:

(1) Arguments NEQN, TV, YV, FCN, and TOUT serve the same purpose as for RKF45 (see Program 1.3d). Also, MSTATE and IFLAG are essentially the same (note that IFLAG = 1 and MSTATE = 1 initialize RKF45 and SDRV2, respectively, prior to the first call to these subroutines; for a successful integration from TV to TOUT, IFLAG = 2 and

Program 4.1 The Call to SDRV2 in Main Program PRO4P1

```
C...
C...  SET THE PARAMETERS FOR SUBROUTINE SDRV2
      EPS=ERROR
      EWT=ERROR
      MSTATE=1
      NROOT=0
      MINT=1
      LENW=7500
      LENIW=21
      TOUT=T0+TP
C...
C...  CALL SUBROUTINE SDRV2 TO START THE SOLUTION FROM THE INITIAL
C...  CONDITION (MSTATE = 1) OR COMPUTE THE SOLUTION TO THE NEXT PRINT
C...  POINT (MSTATE = 2)
4     CALL SDRV2(NEQN,TV,YV,FCN,TOUT,MSTATE,NROOT,EPS,EWT,MINT,
1              WORK,LENW,LENIW,GFUN)
```

MSTATE = 2, and otherwise, IFLAG and MSTATE are used to print an error message).

(2) NROOT defines the number of equations whose roots are to be computed along the ODE solution. Briefly, SDRV2 has the option of finding the roots of a system of NROOT equations programmed in a user-supplied function that is the last argument of SDRV2 (e.g., GFUN).

(3) EPS and EWT define the ODE integration error tolerances. If EWT = 0, a purely relative error tolerance is used. Thus, these two arguments are analogous to ABSREL and RELERR of RKF45.

(4) MINT is the parameter that distinguishes SDRV2 from RKF45. To quote from the SDRV2 documentation [Kahaner (4.1)]:

MINT = (input) the integration method flag.

MINT = 1 means the Adams methods, and is used for nonstiff problems

MINT = 2 means the stiff methods of Gear (i.e., BDFs) and is used for stiff problems.

MINT = 3 means the program dynamically selects the Adams method when the problem is nonstiff and the Gear methods when the problem is stiff.

Thus, MINT = 2 should be used for stiff ODEs, and MINT = 3 should be used if the ODEs might possibly be stiff for some intervals of the independent variable t and nonstiff for other intervals. This, of course, raises the question of how the user knows ahead of time when an ODE problem has the nonstiff, stiff, or combined nonstiff-stiff characteristics requiring the selection of MINT = 1, 2 and 3, respectively. We shall consider this matter in Chapter 5. For the present, note MINT = 1 in Program 4.1 since the 51 ODEs that approximate the solution to equations (1.21) to (1.24) were integrated successfully with a nonstiff integrator, RKF45 (Program 1.3d).

(5) WORK and LENW define a real work array, WORK, of length LENW. To again quote from the SDRV2 documentation:

The length of WORK should be at least

$16*N + 2*NROOT + 204$ if MINT is 1

$N*N + 10*N + 2*NROOT + 204$ if MINT is 2

$N*N + 17*N + 2*NROOT + 204$ if MINT is 3

and LENW should be set to the value (size of WORK) used.

The differences in these sizing formulas for WORK should be noted. For the nonstiff option, MINT = 1, the size of WORK is proportional to N

(the number of ODEs), while for the stiff options, MINT = 2 and 3, the size of WORK is proportional to N^2 . This is a very significant difference for large N (e.g., $N > 1000$), and reflects the fact that stiff (implicit) integrators require the storage of the ODE Jacobian matrix, which is of size N^2 [see equation (4.59)]. In the present case, SDRV2 stores the full Jacobian matrix (all N^2 elements), and does not take advantage of any structure (e.g., bandedness) to reduce the storage requirement.

(6) IWORK and LENIW define an integer work array, IWORK, of length LENIW. This array is sized as 21 if MINT = 1 and $N + 21$ if MINT = 2 or 3.

(7) GFUN is the subroutine that defines the equations for which roots are to be computed [see the discussion of NROOT in (2) above].

The output from the complete main program PRO4P1 is listed in Table 4.2 (execution of PRO4P1, plus Programs 1.3a to 1.3c and data in program 1.3e). As expected, this output is identical to that from RKF45, Table 1.5b.

The preceding example, of course, does not illustrate the advantage of using an implicit integrator for the solution of stiff ODEs. Therefore, we recommend that you apply SDRV2 to the solution of the 2×2 ODE problem of equations (4.1) and (4.2), as defined in Problems 4.3 and 4.4 at the end of this chapter. These problems will clearly demonstrate the advantages of using an implicit integrator for stiff ODEs.

There are two final points we should mention regarding the preceding use of SDRV2:

(1) The code in Program 4.1 is in single precision (the S in the name SDRV2 denotes single-precision; a double-precision version of this integrator, DDRIV2, is also available). We can then logically ask when either single- or double-precision coding should be used. Of course, it is necessary to have enough precision that round off effects do not degrade the solution (this is true for any numerical calculation). However, within the context of stiff ODEs, we can get some idea of precision requirements by considering again the 2×2 system, equations (4.1) and (4.2), for the particular case of $\lambda_1 = -(a - b) = -1$ and $\lambda_2 = -(a + b) = -1,000,000$. Note that in order to calculate α_1 , the subtraction

$-(500000.5 - 499999.5) = -1$ was required, which could not be done with reasonable accuracy, say to obtain λ_1 to four figures or -1.000 , unless the calculation is to at least 10 significant figures. Therefore 32-bit arithmetic, which corresponds to about seven decimal digits, would not be adequate to compute λ_1 . However, the 64-bit arithmetic used to calculate the output of Table 4.1, which corresponds to about 14 decimal digits, is adequate for the calculation of λ_1 . Of course, we are considering two different calculations: (a) the numerical integration of the stiff ODE

Table 4.2 Numerical Solution from Program 4.1

```

RUN NO. - 1 NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24) BY SDRV2

INITIAL T - 0.000E+00

FINAL T - .500E+00

PRINT T - .100E+00

NUMBER OF DIFFERENTIAL EQUATIONS - 51

MAXIMUM INTEGRATION ERROR - .100E-05

TIME = 0.00
      X=0     X=0.2    X=0.4    X=0.6    X=0.8    X=1
T(X,T) 0.000000  .587785  .951057  .951057  .587785  0.000000
TE(X,T) 0.000000  .587785  .951057  .951057  .587785  .000000
DIFF(X,T) 0.000000  0.000000  0.000000  0.000000  0.000000  .000000

TIME = .10
      X=0     X=0.2    X=0.4    X=0.6    X=0.8    X=1
T(X,T) 0.000000  .219357  .354926  .354926  .219357  0.000000
TE(X,T) 0.000000  .219072  .354466  .354466  .219072  .000000
DIFF(X,T) 0.000000  .000284  .000460  .000460  .000284  .000000

TIME = .20
      X=0     X=0.2    X=0.4    X=0.6    X=0.8    X=1
T(X,T) 0.000000  .081862  .132456  .132456  .081862  0.000000
TE(X,T) 0.000000  .081650  .132112  .132112  .081650  .000000
DIFF(X,T) 0.000000  .000212  .000343  .000343  .000212  .000000

TIME = .30
      X=0     X=0.2    X=0.4    X=0.6    X=0.8    X=1
T(X,T) 0.000000  .030550  .049431  .049431  .030550  0.000000
TE(X,T) 0.000000  .030432  .049239  .049239  .030432  .000000
DIFF(X,T) 0.000000  .000119  .000192  .000192  .000119  .000000

TIME = .40
      X=0     X=0.2    X=0.4    X=0.6    X=0.8    X=1
T(X,T) 0.000000  .011401  .018447  .018447  .011401  0.000000
TE(X,T) 0.000000  .011342  .018352  .018352  .011342  .000000
DIFF(X,T) 0.000000  .000059  .000095  .000095  .000059  .000000

TIME = .50
      X=0     X=0.2    X=0.4    X=0.6    X=0.8    X=1
T(X,T) 0.000000  .004255  .006884  .006884  .004255  0.000000
TE(X,T) 0.000000  .004227  .006840  .006840  .004227  .000000
DIFF(X,T) 0.000000  .000028  .000045  .000045  .000028  .000000

```

problem with $\lambda_1 = -1$ and $\lambda_2 = -1,000,000$ and (b) the direct calculation of λ_1 . However, the two are related. In fact, if the *stiffness ratio* of the ODE system (the ratio of the largest to smallest real parts of the ODE eigenvalues) has m orders of magnitude (powers of 10), then the numerical integration of the ODEs should be done with arithmetic having more than m decimal digits. For example, for the 2×2 system, the stiffness ratio is $-1,000,000/-1 = 10^6$, or six orders of magnitude; therefore the arithmetic should have more than six decimal digits of accuracy, e.g., 10 decimal digits, which precludes 32-bit arithmetic, but includes 64-bit

arithmetic. If a 32-bit computer is to be used for the integration of the 2×2 ODE system, double precision (64-bit arithmetic) should be used. This conclusion has not been proven here in any formal way, but such a proof is available elsewhere.

(2) In implementing SDRIV2 (or DDRIV2) on a particular computer, some machine-dependent constants must be set for the computer. These constants are set in subroutine R1MACH (or D1MACH for DDRIV2); they are already programmed in R1MACH and D1MACH for most of the widely used computers, but may have to be defined by the user. Typically, these constants define the precision (machine epsilon or unit roundoff), and the maximum and minimum numbers for the particular computer. Ideally, computer codes should not have machine-dependent constants, but if they are required, they should be located conveniently for user definition (as in the case of R1MACH and D1MACH), which is generally the case for quality software such as SDRIV2 and DDRIV2.

To conclude this section, SDRIV2 is recommended for nonstiff and stiff problems of modest size for which storage of the ODE Jacobian matrix is not a major concern. Next, we consider integrators LSODE, LSODES, and LSODI that *do* take advantage of the structure of the ODE Jacobian matrix. Again, we will not give all of the coding to use these integrators, but rather, just a representative call as in Program 4.1, and a discussion of some of the key parameters. These integrators are thoroughly documented internally (as an extensive set of comments that constitute a user's manual); our experience has been that they are straightforward to use, and very effective.

4.7 LSODE, LSODES, and LSODI

LSODE, LSODES, and LSODI are three ODE integrators in a set of seven integrators developed by Hindmarsh (4.7, 4.8). The names are abbreviations for the *Livermore Solver for Ordinary Differential Equations*, with variations in some of the letters to reflect basic properties of the various integrators. For example, the basic integrator is LSODE for an ODE system with a full or banded Jacobian matrix, and LSODES is for an ODE system with a sparse Jacobian matrix. LSODI is for implicit ODEs (not to be confused with implicit integrators).

Main programs to call these subroutines are easily written. We consider first LSODE, which is called by the following coding used in place of the call to RKF45 in Program 4.2a. The arguments of

Program 4.2a Call to Subroutine LSODE from Program 1.3d

```

C... SET THE PARAMETERS FOR SUBROUTINE LSODE
C... ITOL=1
C... RTOL=ERROR
C... ATOL=ERROR
C... LRW=7500
C... LIW=470
C... IOPT=0
C... ITASK=1
C... ISTATE=1
C... METH=1
C... MITER=0
C... TOUT=TV+TP
C... THE METHOD FLAG, MF, IS COMPUTED FROM METH AND MITER
C... MF=10*METH+MITER
C... CALL SUBROUTINE LSODE TO COVER ONE PRINT INTERVAL
4 CALL LSODE(FCN,NEQN,YV,TV,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,
1           IOPT,RWORK,LRW,IWORK,LIW,JAC,MF)

```

subroutine LSODE are described in a section of the user's manual in Table 4.5 that follows, so they will not be discussed in detail here. However, the parameter MF (Method Flag) should be noted in particular; it is computed from METH (= 1) and MITER (= 0) as $10(10*1 + 0)$, which corresponds to the nonstiff option of LSODE (the integration is performed by Adams formulas). If METH = 2, the stiff option is used (the integration is performed by the BDF methods).

At the end of the loop in Program 1.3d, i.e., just after the following coding

```

C . . . CHECK FOR THE END OF THE RUN
      TOUT = TV + TP
      IF (TV .LT. (TF - 0.5*TP)) GO TO 4

```

the computational statistics for the execution of an LSODE solution are printed from arrays RWORK and IWORK (see Program 4.2b). This use of the work arrays in various integrators, including SDRIV2, LSODE, LSODI, LSODES, and DASSL, to print the computational statistics gives a

Program 4.2b Printing the Computational Statistics from LSODE

```

C... THE CURRENT RUN IS COMPLETE, SO PRINT THE COMPUTATIONAL STAT-
C... ISTICS FOR LSODE AND GO ON TO THE NEXT RUN
      WRITE(NO,1005)RWORK(11),IWORK(14),IWORK(11),IWORK(12),IWORK(13)
      GO TO 1
1005 FORMAT(1H ,//,' COMPUTATIONAL STATISTICS' ,//,
1   ' LAST STEP SIZE          ', E10.3, //,
2   ' LAST ORDER OF THE METHOD ', I10, //,
3   ' TOTAL NUMBER OF STEPS TAKEN ', I10, //,
4   ' NUMBER OF FUNCTION EVALUATIONS ', I10, //,
5   ' NUMBER OF JACOBIAN EVALUATIONS ', I10, /)

```

Table 4.3 Numerical Solution from Programs 4.2a and 4.2b with MF = 10

```

RUN NO. -      1 NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24) BY LSODE
INITIAL T -    0.000E+00
FINAL T -     .500E+00
PRINT T -     .100E+00
NUMBER OF DIFFERENTIAL EQUATIONS -   51
MAXIMUM INTEGRATION ERROR -     .100E-05
TIME =    0.00
          X=0       X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T)  0.000000  .587785  .951057  .951057  .587785  0.000000
TE(X,T) 0.000000  .587785  .951057  .951057  .587785  .000000
DIFF(X,T) 0.000000  0.000000  0.000000  0.000000  0.000000  .000000

TIME =    .10
          X=0       X=0.2      X=0.4      X=0.6      X=0.8      X=1
T(X,T)  0.000000  .226673  .366765  .366765  .226673  0.000000
TE(X,T) 0.000000  .226389  .366305  .366305  .226389  .000000
DIFF(X,T) 0.000000  .000284  .000460  .000460  .000284  .000000

COMPUTATIONAL STATISTICS
LAST STEP SIZE           .992E-04
LAST ORDER OF THE METHOD 2
TOTAL NUMBER OF STEPS TAKEN 500
NUMBER OF FUNCTION EVALUATIONS 830
NUMBER OF JACOBIAN EVALUATIONS 0
ISTATE =   -1
INDICATING AN INTEGRATION ERROR, SO THE CURRENT RUN
IS TERMINATED. PLEASE REFER TO THE DOCUMENTATION FOR
SUBROUTINE
          LSODE

```

useful indication of how the integrators are performing. Note in particular the last order of the BDF method is printed [q in equation (4.55)] as well as the number of function evaluations [dy/dt in equation (4.55) computed in subroutine DERV] and the number of Jacobian evaluations [J in equation (4.59)].

The output from Programs 4.2a and 4.2b (used in Program 1.3d), subroutines INITAL, DERV, and PRINT in Programs 1.3a, 1.3b, and 1.3c, plus data in Program 1.3e is listed in Table 4.3. Several points should be noted concerning this output:

- (1) The solution progressed to only $t = 0.1$ (rather than the final value of $t = 0.5$).
- (2) The run then terminated with the error indicator $ISTATE = -1$, which, as the excerpt from the user's manual will indicate, means that

LSODE required “excessive work” to reach $t = 0.1$. In other words, LSODE reached the limit of 500 steps along the solution set in LSODE as indicated by the computational statistics. This high level of computational effort is consistent with what occurred in using RKF45 in Chapter 1 when we had to increase the number of function evaluations (calls to DERV) to 4000 in order to complete the solution for 51 spatial grid points (see Section 1.6). Thus, again, the computational effort increases significantly with the number of grid points, which, as we discussed in Chapter 1, is due to the increased stiffness of the ODEs and the limited stability of the ODE integrator (remember that in the current run of LSODE, we used the nonstiff option, MF = 10).

(3) The last integration step size was 0.992E-04, which again indicates that the maximum integration step size was constrained by the stability of the nonstiff integrator, as specified approximately by equations (4.36) and (4.37) (these equations apply to the explicit Euler method—a first-order method, while the computational statistics indicate that the nonstiff second-order Adams formulas were used at the end of the run that have a stability interval slightly greater than 2).

(4) The order of the Adams formulas at the end of the run was only 2, although the variable-order algorithm in LSODE could increase this to a maximum of 13.

(5) The number of derivative evaluations (calls to DERV) was 830 (and, again, the solution progressed to only $t = 0.1$ indicating the large computational effort). This is consistent with the requirement to increase the maximum number of calls to DERV to 4000 for RKF45 in Section 1.6.

(6) As expected, the number of Jacobian evaluations was zero because the nonstiff option, MF = 10, was used, which does not require the use of the Jacobian matrix for the solution of algebraic equations by Newton’s method.

We conclude, in general, that this use of LSODE was not satisfactory (a complete solution was not computed with reasonable effort) because of the stiffness of the 51 ODEs and the limited stability of the nonstiff integrator. The obvious thing to do, then, is to repeat the run using a stiff option, MF = 22. This requires a minor change in the arguments of subroutine LSODE:

```

METH = 2
MITER = 2
TOUT = TV + TP
C . .
C . . . THE METHOD FLAG, MF, IS COMPUTED FROM METH AND MITER
MF = 10*METH + MITER

```

where $\text{METH} = 2$ specifies the BDF (stiff) LSODE option and $\text{MITER} = 2$ specifies evaluation of the Jacobian matrix [J in equation (4.59)] by finite differences.

With this change in the call to LSODE (everything else in Programs 4.2a and 4.2b remains the same), the output of Programs 4.2a, 4.2b, and 1.3a to 1.3e is as shown in Table 4.4.

We note a marked difference between the outputs of Tables 4.3 and 4.4:

- (1) The solution is complete (to $t = 0.5$), and is essentially identical to the output of subroutine RKF45 (Table 1.5b).
- (2) The number of steps is only 55 (for a complete solution vs. 500 for a solution to $t = 0.1$ with $\text{MF} = 10$).
- (3) The number of function evaluations is 628 (for a complete solution vs. 830 for a solution to $t = 0.1$ with $\text{MF} = 10$).
- (4) The final step size is 0.246E-01, so the ratio of final step sizes ($\text{MF} = 22/\text{MF} = 10$) is $0.246\text{E-01}/0.992\text{E-04} = 248$.
- (5) The last BDF order is 5, which is the maximum for the stiff option.
- (6) The number of Jacobian evaluations in equation (4.59) is 11 (vs. zero for the nonstiff option). Therefore, the Jacobian matrix was updated only about every fifth integration step (55/11), but as long as the Newton iteration of the algebraic equations is achieved, the accuracy is unaffected by using an approximate Jacobian matrix (J not based on the current values of y). Therefore, the *additional effort of computing the Jacobian matrix was clearly worthwhile* (since a complete solution to the required accuracy resulted); much larger integration steps could be used [see (4) above], which, in general, is the *advantage of using an implicit integrator*. This is not a particularly stiff problem (or we would not have been able to compute a solution using RKF45). For a highly stiff problem [e.g., a stiffness ratio of 10^6 or more, as in equations (4.38) and (4.39)], the increase in the integration step size can be orders of magnitude. To demonstrate this efficiency of implicit integrators, we suggest doing at least one of Problems 4.4 to 4.8.

To conclude this discussion of LSODE, we list in Table 4.5 the documentation for the arguments of LSODE (taken from the user's manual, an extensive set of comments at the beginning of LSODE). This explanation of the LSODE arguments should be essentially self-explanatory. Only three will be discussed here:

- (1) ATOL is the absolute error tolerance. Note that it can be either a single value applied to all of the ODE dependent variables [y_1 to y_n in

Table 4.4 Numerical Solution from Program 4.2a and 4.2b with MF = 22

RUN NO. - 1 NUMOL SOLUTION OF EQUATIONS (1.21) TO (1.24) BY LSODE

INITIAL T - 0.000E+00

FINAL T - .500E+00

PRINT T - .100E+00

NUMBER OF DIFFERENTIAL EQUATIONS - 51

MAXIMUM INTEGRATION ERROR - .100E-05

TIME = 0.00

	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.587785	.951057	.951057	.587785	0.000000
TE(X,T)	0.000000	.587785	.951057	.951057	.587785	.000000
DIFF(X,T)	0.000000	0.000000	0.000000	0.000000	0.000000	.000000

TIME = .10

	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.219356	.354925	.354925	.219356	0.000000
TE(X,T)	0.000000	.219072	.354466	.354466	.219072	.000000
DIFF(X,T)	0.000000	.000284	.000459	.000459	.000284	.000000

TIME = .20

	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.081863	.132457	.132457	.081863	0.000000
TE(X,T)	0.000000	.081650	.132112	.132112	.081650	.000000
DIFF(X,T)	0.000000	.000213	.000345	.000345	.000213	.000000

TIME = .30

	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.030552	.049434	.049434	.030552	0.000000
TE(X,T)	0.000000	.030432	.049239	.049239	.030432	.000000
DIFF(X,T)	0.000000	.000120	.000195	.000195	.000120	.000000

TIME = .40

	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.011402	.018450	.018450	.011402	0.000000
TE(X,T)	0.000000	.011342	.018352	.018352	.011342	.000000
DIFF(X,T)	0.000000	.000060	.000098	.000098	.000060	.000000

TIME = .50

	X=0	X=0.2	X=0.4	X=0.6	X=0.8	X=1
T(X,T)	0.000000	.004256	.006886	.006886	.004256	0.000000
TE(X,T)	0.000000	.004227	.006840	.006840	.004227	.000000
DIFF(X,T)	0.000000	.000029	.000046	.000046	.000029	.000000

COMPUTATIONAL STATISTICS

LAST STEP SIZE	.246E-01
LAST ORDER OF THE METHOD	5
TOTAL NUMBER OF STEPS TAKEN	55
NUMBER OF FUNCTION EVALUATIONS	628
NUMBER OF JACOBIAN EVALUATIONS	11

Table 4.5 User Information for Subroutine LSODE

C WRITE A MAIN PROGRAM WHICH CALLS SUBROUTINE LSODE ONCE FOR EACH
C POINT AT WHICH ANSWERS ARE DESIRED. THIS SHOULD ALSO PROVIDE
C FOR POSSIBLE USE OF LOGICAL UNIT 6 FOR OUTPUT OF ERROR MESSAGES
C BY LSODE. ON THE FIRST CALL TO LSODE, SUPPLY ARGUMENTS AS FOLLOWS..
C F = NAME OF SUBROUTINE FOR RIGHT-HAND SIDE VECTOR F.
C THIS NAME MUST BE DECLARED EXTERNAL IN CALLING PROGRAM.
C NEQ = NUMBER OF FIRST ORDER ODE-S.
C Y = ARRAY OF INITIAL VALUES, OF LENGTH NEQ.
C T = THE INITIAL VALUE OF THE INDEPENDENT VARIABLE.
C TOUT = FIRST POINT WHERE OUTPUT IS DESIRED (.NE. T).
C ITOL = 1 OR 2 ACCORDING AS ATOL (BELOW) IS A SCALAR OR ARRAY.
C RTOL = RELATIVE TOLERANCE PARAMETER (SCALAR).
C ATOL = ABSOLUTE TOLERANCE PARAMETER (SCALAR OR ARRAY).
C THE ESTIMATED LOCAL ERROR IN Y(I) WILL BE CONTROLLED SO AS
C TO BE ROUGHLY LESS (IN MAGNITUDE) THAN
C EWT(I) = RTOL*ABS(Y(I)) + ATOL IF ITOL = 1, OR
C EWT(I) = RTOL*ABS(Y(I)) + ATOL(I) IF ITOL = 2.
C THUS THE LOCAL ERROR TEST PASSES IF, IN EACH COMPONENT,
C EITHER THE ABSOLUTE ERROR IS LESS THAN ATOL (OR ATOL(I)),
C OR THE RELATIVE ERROR IS LESS THAN RTOL.
C USE RTOL = 0.0 FOR PURE ABSOLUTE ERROR CONTROL, AND
C USE ATOL = 0.0 (OR ATOL(I) = 0.0) FOR PURE RELATIVE ERROR
C CONTROL. CAUTION.. ACTUAL (GLOBAL) ERRORS MAY EXCEED THESE
C LOCAL TOLERANCES, SO CHOOSE THEM CONSERVATIVELY.
C ITASK = 1 FOR NORMAL COMPUTATION OF OUTPUT VALUES OF Y AT T = TOUT.
C ISTATE = INTEGER FLAG (INPUT AND OUTPUT). SET ISTATE = 1.
C IOPT = 0 TO INDICATE NO OPTIONAL INPUTS USED.
C RWORK = REAL WORK ARRAY OF LENGTH AT LEAST..
C 20 + 16*NEQ FOR MF = 10,
C 22 + 9*NEQ + NEQ**2 FOR MF = 21 OR 22,
C 22 + 10*NEQ + (2*ML + MU)*NEQ FOR MF = 24 OR 25.
C LRW = DECLARED LENGTH OF RWORK (IN USER-S DIMENSION).
C IWORK = INTEGER WORK ARRAY OF LENGTH AT LEAST..
C 20 FOR MF = 10,
C 20 + NEQ FOR MF = 21, 22, 24, OR 25.
C IF MF = 24 OR 25, INPUT IN IWORK(1), IWORK(2) THE LOWER
C AND UPPER HALF-BANDWIDTHS ML,MU.
C LIW = DECLARED LENGTH OF IWORK (IN USER-S DIMENSION).
C JAC = NAME OF SUBROUTINE FOR JACOBIAN MATRIX (MF = 21 OR 24).
C IF USED, THIS NAME MUST BE DECLARED EXTERNAL IN CALLING
C PROGRAM. IF NOT USED, PASS A DUMMY NAME.
C MF = METHOD FLAG. STANDARD VALUES ARE..
C 10 FOR NONSTIFF (ADAMS) METHOD, NO JACOBIAN USED.
C 21 FOR STIFF (BDF) METHOD, USER-SUPPLIED FULL JACOBIAN.
C 22 FOR STIFF METHOD, INTERNALLY GENERATED FULL JACOBIAN.
C 24 FOR STIFF METHOD, USER-SUPPLIED BANDED JACOBIAN.
C 25 FOR STIFF METHOD, INTERNALLY GENERATED BANDED JACOBIAN.
C NOTE THAT THE MAIN PROGRAM MUST DECLARE ARRAYS Y, RWORK, IWORK,
C AND POSSIBLY ATOL.
C
C E. THE OUTPUT FROM THE FIRST CALL (OR ANY CALL) IS..
C Y = ARRAY OF COMPUTED VALUES OF Y(T) VECTOR.
C T = CORRESPONDING VALUE OF INDEPENDENT VARIABLE (NORMALLY TOUT).
C ISTATE = 2 IF LSODE WAS SUCCESSFUL, NEGATIVE OTHERWISE.
C -1 MEANS EXCESS WORK DONE ON THIS CALL (PERHAPS WRONG MF).
C -2 MEANS EXCESS ACCURACY REQUESTED (TOLERANCES TOO SMALL).
C -3 MEANS ILLEGAL INPUT DETECTED (SEE PRINTED MESSAGE).
C -4 MEANS REPEATED ERROR TEST FAILURES (CHECK ALL INPUTS).
C -5 MEANS REPEATED CONVERGENCE FAILURES (PERHAPS BAD JACOBIAN
C SUPPLIED OR WRONG CHOICE OF MF OR TOLERANCES).
C -6 MEANS ERROR WEIGHT BECAME ZERO DURING PROBLEM. (SOLUTION
C COMPONENT I VANISHED, AND ATOL OR ATOL(I) = 0.)

equation (4.16)] if ITOL = 1 or an array containing an absolute error criterion for each of the ODE dependent variables if ITOL = 2.

(2) WORK is the real work array. Note in particular the sizing formulas for this array. Again, as in the case of SDRIV2, the nonstiff option (MF = 10) requires an array size proportional to the number of ODEs (NEQ), while the stiff option with a full Jacobian matrix (MF = 21, 22) requires an array size proportional to the square of the number of ODEs. However, if the ODE Jacobian matrix is banded (with ML diagonals below the main diagonal and MU diagonals above the main diagonal), the size of WORK is $22 + 10 * \text{NEQN} + (2 * \text{ML} + \text{MU}) * \text{NEQ}$ (for MF = 24, 25). If $\text{ML} \ll \text{NEQ}$ and $\text{MU} \ll \text{NEQ}$, this results in a very substantial reduction in the size of WORK. For the present case, using subroutine DSS002 (in Program 1.3b), $\text{NEQ} = 51$, $\text{ML} = \text{MU} = 2$. Therefore, WORK could be sized as $22 + 10 * 55 + (2 * 2 + 2) * 55 = 902$ if the banded option is used (MF = 24, 25), while for the full matrix option actually used (MF = 22 in Program 4.2a), the minimum size for WORK is $22 + 9 * 51 + 51 * 2 = 2981$ or a ratio of $902/2981 = 0.303$. This problem is very modest in size (51 ODEs), and the reduction in the size of WORK would be substantially greater with increasing NEQ. In Section 4.5 we observed that the number of arithmetic operations for solving a banded algebraic system [$\text{NM}^2 = 51(2 + 2 + 1)^2 = 1275$ for the present problem] is substantially smaller than for a full algebraic system ($N^3 = 51^3 = 132,651$). Thus, not only is the storage requirement (size of WORK) significantly reduced by exploiting the structure of a banded matrix; the number of arithmetic operations required in handling the Jacobian matrix (J in equation (4.59)] is also reduced each time equation (4.59) is solved (approximately $1275/132,651 = 0.0096$ for the present problem). Clearly it is worthwhile to exploit the structure of the Jacobian matrix to reduce the computational effort (and, in fact, this could have been done in the present case by using MF = 25 rather than MF = 22, but, of course, we would have had to also specify ML and MU as IWORK(1) = 2 and IWORK(2) = 2).

(3) JAC is the user-supplied subroutine for the ODE Jacobian matrix. If MF = 21, the $N \times N$ elements (partial derivatives) of the Jacobian matrix must be programmed in subroutine JAC. (This would be $51 \times 51 = 2601$ elements in the present case!) For MF = 24, the user is required to program only the elements of the Jacobian matrix in the band around the main diagonal [approximately $51 \times (2 + 2 + 1) = 255$ elements in the present case; these would be highly repetitive so that a pair of nested DO loops could be used to good advantage]. This modest problem (51 ODEs) indicates the advantage of having LSODE compute the Jacobian matrix numerically (MF = 22 full or 25 banded), and this is,

in fact, what is usually done for NUMOL solutions; if this option is selected, the user need only provide a dummy subroutine JAC (a subroutine that satisfies the loader but does not have any executable code since it is not actually called if MF = 22 or 25). Also, in the case of SDRV2, the Jacobian matrix is always computed numerically, so a user-supplied Jacobian matrix is not required.

(4) MF is the method flag. This has been discussed in some detail, so further elaboration is not required.

(5) ISTATE is the error flag indicator. Note that ISTATE = 1 to initialize LSODE (before the first call to LSODE), and ISTATE < 0 corresponds to an error condition (ISTATE = -1 was produced by Program 4.2a, indicating excessive work or computational effort).

This concludes the discussion of LSODE. Much more information is available in the user's manual, with an extensive set of comments at the beginning of LSODE, including several worked example applications with output. One important difference we have noted between SDRV2 and LSODE is that the latter can take advantage of the banded structure of the ODE Jacobian matrix, which frequently permits substantial reductions in storage and computational effort, particularly in developing NUMOL solutions since the ODEs are often banded.

This raises the issue, however, of how do we ascertain the actual Jacobian matrix structure of the ODEs in a NUMOL solution. We will take a direct approach to answering this question in Chapter 5, where we will *map the Jacobian matrix*, i.e., we will print a two-dimensional map with the ODE dependent variables, y_1 to y_N , listed across the top, and the ODE derivatives, dy_1/dt to dy_N/dt , listed down the left side. If derivative dy_i/dt is dependent on dependent variable y_j , then the map will have a nonblank character printed in the ij th position. This discussion illustrates why the Jacobian matrix is often banded around the main diagonal; in most physical problems, a particular derivative, dy_i/dt , depends on y_i (thus accounting for the main diagonal elements, with $i = j$), and a few surrounding dependent variables, e.g., y_{i-2} , y_{i+1} , y_{i+1} , and y_{i+2} corresponding to $ML = MU = 2$ as occurs in subroutine DSS002. However, for relatively complicated systems of PDEs, the ODE Jacobian structure is difficult to visualize (which is required so that ML and MU can be selected in programming a NUMOL solution with LSODE), and therefore, a printed map of the Jacobian matrix is quite useful. We shall illustrate this mapping technique in Chapter 5.

Additionally, some physical problems produce an ODE Jacobian matrix without a well-defined structure (the elements appear to be positioned throughout the matrix in a rather random pattern). Often the

matrix has mostly zeros (90 to 95% zeros is not uncommon); matrices with this structure (actually, lack of apparent structure and mostly zeros) are termed *sparse*. Therefore, integrators that accommodate sparse ODEs efficiently are generally useful; in the case of the LSODE library, LSODES accommodates sparse ODEs (the final S denotes *Sparse*).

The programming of an integrator for a sparse ODE system is inherently more difficult than for an ODE system with a well-defined structure; for example, in the case of banded systems, all we had to do was specify ML and MU to accommodate the band around the main diagonal. For the sparse case, the nonzero elements can occur anywhere, and we do not wish to store or process the zero elements (to achieve efficiency). Thus, the integrator must first determine the location of the nonzero elements, the so-called sparsity structure of the ODEs (which, of course, will be different for each new problem, and can even change during the solution to a nonlinear ODE system because the elements of the Jacobian matrix are functions of the dependent variables). Fortunately, all of the details for accommodating sparse ODE systems have been carefully studied and programmed (for example, in LSODES), so that the user need only specify a few additional parameters before calling the ODE integrator.

A call to LSODES for equations (1.21) to (1.24) is listed in Program 4.3 (this call was again put into Program 1.3d in place of the call to RKF45). The numerical solution from the execution of Program 4.3 is identical to Table 4.4. The computational statistics, produced from the coding in Program 4.2b, are as shown in Table 4.6. A comparison of the computational statistics from Table 4.4 for LSODE and Table 4.6 for LSODES indicates LSODES was more efficient (note the difference in the numbers of function evaluations and Jacobian evaluations, which are the major part of the computational effort). These results would suggest that LSODES was effective in exploiting the sparse structure of the 51 ODEs; in

Program 4.3 Call to Subroutine LSODES from Program 1.3d

```
C...
C...  SET THE PARAMETERS FOR SUBROUTINE LSODES
      ITOL=1
      RTOL=ERROR
      ATOL=ERROR
      LRW=9000
      LIW=30
      IOPT=0
      ITASK=1
      ISTATE=1
      MF=222
C...
C...  CALL SUBROUTINE LSODES TO COVER ONE PRINT INTERVAL
4      CALL LSODES(FCN,NEQN,YV,TV,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,
1                  IOPT,RWORK,LRW,IWORK,LIW,JAC,MF)
```

Table 4.6 Computational Statistics from Program 4.3

COMPUTATIONAL STATISTICS

LAST STEP SIZE	.246E-01
LAST ORDER OF THE METHOD	5
TOTAL NUMBER OF STEPS TAKEN	55
NUMBER OF FUNCTION EVALUATIONS	73
NUMBER OF JACOBIAN EVALUATIONS	2

general, we have found this to be the case, particularly for large, complex PDE problems.

The arguments for LSODES closely parallel those for LSODE. In Table 4.7 we list, without comment, the section of the LSODES user's manual describing the arguments. The user information of Table 4.5

Table 4.7 User Information for Subroutine LSODES

```
C THE DESCRIPTIONS OF THE CALL ARGUMENTS ARE AS FOLLOWS.
C
C F      = THE NAME OF THE USER-SUPPLIED SUBROUTINE DEFINING THE
C           ODE SYSTEM. THE SYSTEM MUST BE PUT IN THE FIRST-ORDER
C           FORM DY/DT = F(T,Y), WHERE F IS A VECTOR-VALUED FUNCTION
C           OF THE SCALAR T AND THE VECTOR Y. SUBROUTINE F IS TO
C           COMPUTE THE FUNCTION F. IT IS TO HAVE THE FORM
C           SUBROUTINE F (NEQ, T, Y, YDOT)
C           DIMENSION Y(1), YDOT(1)
C           WHERE NEQ, T, AND Y ARE INPUT, AND THE ARRAY YDOT =
C           F(T,Y) IS OUTPUT. Y AND YDOT ARE ARRAYS OF LENGTH
C           NEQ. (IN THE DIMENSION STATEMENT ABOVE, 1 IS A DUMMY
C           DIMENSION.. IT CAN BE REPLACED BY ANY VALUE.)
C           SUBROUTINE F SHOULD NOT ALTER Y(1),...,Y(NEQ). F
C           MUST BE DECLARED EXTERNAL IN THE CALLING PROGRAM.
C
C           SUBROUTINE F MAY ACCESS USER-DEFINED QUANTITIES IN
C           NEQ(2),... AND Y(NEQ(1)+1),... IF NEQ IS AN ARRAY
C           (DIMENSIONED IN F) AND Y HAS LENGTH EXCEEDING NEQ(1).
C           SEE THE DESCRIPTIONS OF NEQ AND Y BELOW.
C
C NEQ    = THE SIZE OF THE ODE SYSTEM (NUMBER OF FIRST ORDER
C           ORDINARY DIFFERENTIAL EQUATIONS). USED ONLY FOR INPUT.
C           NEQ MAY BE DECREASED, BUT NOT INCREASED, DURING THE PROBLEM.
C           IF NEQ IS DECREASED (WITH ISTATE = 3 ON INPUT), THE
C           REMAINING COMPONENTS OF Y SHOULD BE LEFT UNDISTURBED, IF
C           THESE ARE TO BE ACCESSED IN F AND/OR JAC.
C
C           NORMALLY, NEQ IS A SCALAR, AND IT IS GENERALLY REFERRED TO
C           AS A SCALAR IN THIS USER INTERFACE DESCRIPTION. HOWEVER,
C           NEQ MAY BE AN ARRAY, WITH NEQ(1) SET TO THE SYSTEM SIZE.
C           (THE LSODES PACKAGE ACCESSES ONLY NEQ(1).) IN EITHER CASE,
C           THIS PARAMETER IS PASSED AS THE NEQ ARGUMENT IN ALL CALLS
C           TO F AND JAC. HENCE, IF IT IS AN ARRAY, LOCATIONS
C           NEQ(2),... MAY BE USED TO STORE OTHER INTEGER DATA AND PASS
C           IT TO F AND/OR JAC. SUBROUTINES F AND/OR JAC MUST INCLUDE
C           NEQ IN A DIMENSION STATEMENT IN THAT CASE.
```

continues

C Y = A REAL ARRAY FOR THE VECTOR OF DEPENDENT VARIABLES, OF
C LENGTH NEQ OR MORE. USED FOR BOTH INPUT AND OUTPUT ON THE
C FIRST CALL (ISTATE = 1), AND ONLY FOR OUTPUT ON OTHER CALLS.
C ON THE FIRST CALL, Y MUST CONTAIN THE VECTOR OF INITIAL
C VALUES. ON OUTPUT, Y CONTAINS THE COMPUTED SOLUTION VECTOR,
C EVALUATED AT T. IF DESIRED, THE Y ARRAY MAY BE USED FOR
C OTHER PURPOSES BETWEEN CALLS TO THE SOLVER.

C THIS ARRAY IS PASSED AS THE Y ARGUMENT IN ALL CALLS TO
C F AND JAC. HENCE ITS LENGTH MAY EXCEED NEQ, AND LOCATIONS
C Y(NEQ+1),... MAY BE USED TO STORE OTHER REAL DATA AND PASS
C IT TO F AND/OR JAC. (THE LSODES PACKAGE ACCESSES ONLY
C Y(1),...,Y(NEQ).)

C T = THE INDEPENDENT VARIABLE. ON INPUT, T IS USED ONLY ON THE
C FIRST CALL, AS THE INITIAL POINT OF THE INTEGRATION.
C ON OUTPUT, AFTER EACH CALL, T IS THE VALUE AT WHICH A
C COMPUTED SOLUTION Y IS EVALUATED (USUALLY THE SAME AS TOUT).
C ON AN ERROR RETURN, T IS THE FARDEST POINT REACHED.

C TOUT = THE NEXT VALUE OF T AT WHICH A COMPUTED SOLUTION IS DESIRED.
C USED ONLY FOR INPUT.

C WHEN STARTING THE PROBLEM (ISTATE = 1), TOUT MAY BE EQUAL
C TO T FOR ONE CALL, THEN SHOULD .NE. T FOR THE NEXT CALL.
C FOR THE INITIAL T, AN INPUT VALUE OF TOUT .NE. T IS USED
C IN ORDER TO DETERMINE THE DIRECTION OF THE INTEGRATION
C (I.E. THE ALGEBRAIC SIGN OF THE STEP SIZES) AND THE ROUGH
C SCALE OF THE PROBLEM. INTEGRATION IN EITHER DIRECTION
C (FORWARD OR BACKWARD IN T) IS PERMITTED.

C IF ITASK = 2 OR 5 (ONE-STEP MODES), TOUT IS IGNORED AFTER
C THE FIRST CALL (I.E. THE FIRST CALL WITH TOUT .NE. T).
C OTHERWISE, TOUT IS REQUIRED ON EVERY CALL.

C IF ITASK = 1, 3, OR 4, THE VALUES OF TOUT NEED NOT BE
C MONOTONE, BUT A VALUE OF TOUT WHICH BACKS UP IS LIMITED
C TO THE CURRENT INTERNAL T INTERVAL, WHOSE ENDPOINTS ARE
C TCUR - HU AND TCUR (SEE OPTIONAL OUTPUTS, BELOW, FOR
C TCUR AND HU).

C ITOL = AN INDICATOR FOR THE TYPE OF ERROR CONTROL. SEE
C DESCRIPTION BELOW UNDER ATOL. USED ONLY FOR INPUT.

C RTOL = A RELATIVE ERROR TOLERANCE PARAMETER, EITHER A SCALAR OR
C AN ARRAY OF LENGTH NEQ. SEE DESCRIPTION BELOW UNDER ATOL.
C INPUT ONLY.

C ATOL = AN ABSOLUTE ERROR TOLERANCE PARAMETER, EITHER A SCALAR OR
C AN ARRAY OF LENGTH NEQ. INPUT ONLY.

C THE INPUT PARAMETERS ITOL, RTOL, AND ATOL DETERMINE THE
C ERROR CONTROL PERFORMED BY THE SOLVER. THE SOLVER WILL
C CONTROL THE VECTOR E = (E(I)) OF ESTIMATED LOCAL ERRORS
C IN Y, ACCORDING TO AN INEQUALITY OF THE FORM
C RMS-NORM OF (E(I)/EWT(I)) .LE. 1,
C WHERE EWT(I) = RTOL(I)*ABS(Y(I)) + ATOL(I),
C AND THE RMS-NORM (ROOT-MEAN-SQUARE NORM) HERE IS
C RMS-NORM(V) = SQRT(SUM V(I)**2 / NEQ). HERE EWT = (EWT(I))
C IS A VECTOR OF WEIGHTS WHICH MUST ALWAYS BE POSITIVE, AND
C THE VALUES OF RTOL AND ATOL SHOULD ALL BE NON-NEGATIVE.
C THE FOLLOWING TABLE GIVES THE TYPES (SCALAR/ARRAY) OF
C RTOL AND ATOL, AND THE CORRESPONDING FORM OF EWT(I).

continues

C ITOL RTOL ATOL EWT(I)

1	SCALAR	SCALAR	RTOL*ABS(Y(I)) + ATOL
2	SCALAR	ARRAY	RTOL*ABS(Y(I)) + ATOL(I)
3	ARRAY	SCALAR	RTOL(I)*ABS(Y(I)) + ATOL
4	ARRAY	ARRAY	RTOL(I)*ABS(Y(I)) + ATOL(I)

C WHEN EITHER OF THESE PARAMETERS IS A SCALAR, IT NEED NOT
C BE DIMENSIONED IN THE USER-S CALLING PROGRAM.

C IF NONE OF THE ABOVE CHOICES (WITH ITOL, RTOL, AND ATOL
C FIXED THROUGHOUT THE PROBLEM) IS SUITABLE, MORE GENERAL
C ERROR CONTROLS CAN BE OBTAINED BY SUBSTITUTING USER-
C SUPPLIED ROUTINES FOR THE SETTING OF EWT AND/OR FOR
C THE NORM CALCULATION. SEE PART IV BELOW.

C IF GLOBAL ERRORS ARE TO BE ESTIMATED BY MAKING A REPEATED
C RUN ON THE SAME PROBLEM WITH SMALLER TOLERANCES, THEN ALL
C COMPONENTS OF RTOL AND ATOL (I.E. OF EWT) SHOULD BE SCALED
C DOWN UNIFORMLY.

C ITASK = AN INDEX SPECIFYING THE TASK TO BE PERFORMED.
C INPUT ONLY. ITASK HAS THE FOLLOWING VALUES AND MEANINGS.

- 1 MEANS NORMAL COMPUTATION OF OUTPUT VALUES OF Y(T) AT
T = TOUT (BY OVERSHOOTING AND INTERPOLATING).
- 2 MEANS TAKE ONE STEP ONLY AND RETURN.
- 3 MEANS STOP AT THE FIRST INTERNAL MESH POINT AT OR
BEYOND T = TOUT AND RETURN.
- 4 MEANS NORMAL COMPUTATION OF OUTPUT VALUES OF Y(T) AT
T = TOUT BUT WITHOUT OVERSHOOTING T = TCRIT. TCRIT
MUST BE INPUT AS RWORK(1). TCRIT MAY BE EQUAL TO
OR BEYOND TOUT, BUT NOT BEHIND IT IN THE DIRECTION OF
INTEGRATION. THIS OPTION IS USEFUL IF THE PROBLEM
HAS A SINGULARITY AT OR BEYOND T = TCRIT.
- 5 MEANS TAKE ONE STEP, WITHOUT PASSING TCRIT, AND RETURN.
TCRIT MUST BE INPUT AS RWORK(1).

C NOTE.. IF ITASK = 4 OR 5 AND THE SOLVER REACHES TCRIT
(WITHIN ROUNDOFF), IT WILL RETURN T = TCRIT (EXACTLY) TO
INDICATE THIS (UNLESS ITASK = 4 AND TOUT COMES BEFORE TCRIT,
IN WHICH CASE ANSWERS AT T = TOUT ARE RETURNED FIRST).

C ISTATE = AN INDEX USED FOR INPUT AND OUTPUT TO SPECIFY THE
C THE STATE OF THE CALCULATION.

C ON INPUT, THE VALUES OF ISTATE ARE AS FOLLOWS.

- 1 MEANS THIS IS THE FIRST CALL FOR THE PROBLEM
(INITIALIZATIONS WILL BE DONE). SEE NOTE BELOW.
- 2 MEANS THIS IS NOT THE FIRST CALL, AND THE CALCULATION
IS TO CONTINUE NORMALLY, WITH NO CHANGE IN ANY INPUT
PARAMETERS EXCEPT POSSIBLY TOUT AND ITASK.
(IF ITOL, RTOL, AND/OR ATOL ARE CHANGED BETWEEN CALLS
WITH ISTATE = 2, THE NEW VALUES WILL BE USED BUT NOT
TESTED FOR LEGALITY.)
- 3 MEANS THIS IS NOT THE FIRST CALL, AND THE CALCULATION
IS TO CONTINUE NORMALLY, BUT WITH A CHANGE IN INPUT
PARAMETERS OTHER THAN TOUT AND ITASK. CHANGES ARE
ALLOWED IN NEQ, ITOL, RTOL, ATOL, IOPT, LRW, LIW, MF,
THE CONDITIONAL INPUTS IA AND JA, AND ANY OF THE
OPTIONAL INPUTS EXCEPT HO.
IN PARTICULAR, IF MITER = 1 OR 2, A CALL WITH ISTATE = 3
WILL CAUSE THE SPARSITY STRUCTURE OF THE PROBLEM TO BE
RECOMPUTED (OR REREAD FROM IA AND JA IF MOSS = 0).

C NOTE.. A PRELIMINARY CALL WITH TOUT = T IS NOT COUNTED
AS A FIRST CALL HERE, AS NO INITIALIZATION OR CHECKING OF

continues

C INPUT IS DONE. (SUCH A CALL IS SOMETIMES USEFUL FOR THE
C PURPOSE OF OUTPUTTING THE INITIAL CONDITIONS.) THUS
C THE FIRST CALL FOR WHICH TOUT .NE. T REQUIRES ISTATE = 1
C ON INPUT.

C ON OUTPUT, ISTATE HAS THE FOLLOWING VALUES AND MEANINGS.
C 1 MEANS NOTHING WAS DONE, AS TOUT WAS EQUAL TO T WITH
C ISTATE = 1 ON INPUT. (HOWEVER, AN INTERNAL COUNTER WAS
C SET TO DETECT AND PREVENT REPEATED CALLS OF THIS TYPE.)
C 2 MEANS THE INTEGRATION WAS PERFORMED SUCCESSFULLY.
C -1 MEANS AN EXCESSIVE AMOUNT OF WORK (MORE THAN MXSTEP
C STEPS) WAS DONE ON THIS CALL, BEFORE COMPLETING THE
C REQUESTED TASK, BUT THE INTEGRATION WAS OTHERWISE
C SUCCESSFUL AS FAR AS T. (MXSTEP IS AN OPTIONAL INPUT
C AND IS NORMALLY 500.) TO CONTINUE, THE USER MAY
C SIMPLY RESET ISTATE TO A VALUE .GT. 1 AND CALL AGAIN
C (THE EXCESS WORK STEP COUNTER WILL BE RESET TO 0).
C IN ADDITION, THE USER MAY INCREASE MXSTEP TO AVOID
C THIS ERROR RETURN (SEE BELOW ON OPTIONAL INPUTS).
C -2 MEANS TOO MUCH ACCURACY WAS REQUESTED FOR THE PRECISION
C OF THE MACHINE BEING USED. THIS WAS DETECTED BEFORE
C COMPLETING THE REQUESTED TASK, BUT THE INTEGRATION
C WAS SUCCESSFUL AS FAR AS T. TO CONTINUE, THE TOLERANCE
C PARAMETERS MUST BE RESET, AND ISTATE MUST BE SET
C TO 3. THE OPTIONAL OUTPUT TOLSF MAY BE USED FOR THIS
C PURPOSE. (NOTE.. IF THIS CONDITION IS DETECTED BEFORE
C TAKING ANY STEPS, THEN AN ILLEGAL INPUT RETURN
C (ISTATE = -3) OCCURS INSTEAD.)
C -3 MEANS ILLEGAL INPUT WAS DETECTED, BEFORE TAKING ANY
C INTEGRATION STEPS. SEE WRITTEN MESSAGE FOR DETAILS.
C NOTE.. IF THE SOLVER DETECTS AN INFINITE LOOP OF CALLS
C TO THE SOLVER WITH ILLEGAL INPUT, IT WILL CAUSE
C THE RUN TO STOP.
C -4 MEANS THERE WERE REPEATED ERROR TEST FAILURES ON
C ONE ATTEMPTED STEP, BEFORE COMPLETING THE REQUESTED
C TASK, BUT THE INTEGRATION WAS SUCCESSFUL AS FAR AS T.
C THE PROBLEM MAY HAVE A SINGULARITY, OR THE INPUT
C MAY BE INAPPROPRIATE.
C -5 MEANS THERE WERE REPEATED CONVERGENCE TEST FAILURES ON
C ONE ATTEMPTED STEP, BEFORE COMPLETING THE REQUESTED
C TASK, BUT THE INTEGRATION WAS SUCCESSFUL AS FAR AS T.
C THIS MAY BE CAUSED BY AN INACCURATE JACOBIAN MATRIX,
C IF ONE IS BEING USED.
C -6 MEANS EWT(I) BECAME ZERO FOR SOME I DURING THE
C INTEGRATION. PURE RELATIVE ERROR CONTROL (ATOL(I)=0.0)
C WAS REQUESTED ON A VARIABLE WHICH HAS NOW VANISHED.
C THE INTEGRATION WAS SUCCESSFUL AS FAR AS T.
C -7 MEANS A FATAL ERROR RETURN FLAG CAME FROM THE SPARSE
C SOLVER CDRV BY WAY OF PRJS OR SLSS (NUMERICAL
C FACTORIZATION OR BACKSOLVE). THIS SHOULD NOT HAPPEN.

C NOTE.. AN ERROR RETURN WITH ISTATE = -1, -4, OR -5 AND WITH
C MITER = 1 OR 2 MAY MEAN THAT THE SPARSITY STRUCTURE OF THE
C PROBLEM HAS CHANGED SIGNIFICANTLY SINCE IT WAS LAST
C DETERMINED (OR INPUT). IN THAT CASE, ONE CAN ATTEMPT TO
C COMPLETE THE INTEGRATION BY SETTING ISTATE = 3 ON THE NEXT
C CALL, SO THAT A NEW STRUCTURE DETERMINATION IS DONE.

C NOTE.. SINCE THE NORMAL OUTPUT VALUE OF ISTATE IS 2,
C IT DOES NOT NEED TO BE RESET FOR NORMAL CONTINUATION.
C ALSO, SINCE A NEGATIVE INPUT VALUE OF ISTATE WILL BE
C REGARDED AS ILLEGAL, A NEGATIVE OUTPUT VALUE REQUIRES
C THE USER TO CHANGE IT, AND POSSIBLY OTHER INPUTS, BEFORE
C CALLING THE SOLVER AGAIN.

continues

C IOPT = AN INTEGER FLAG TO SPECIFY WHETHER OR NOT ANY OPTIONAL
 C INPUTS ARE BEING USED ON THIS CALL. INPUT ONLY.
 C THE OPTIONAL INPUTS ARE LISTED SEPARATELY BELOW.
 C IOPT = 0 MEANS NO OPTIONAL INPUTS ARE BEING USED.
 C DEFAULT VALUES WILL BE USED IN ALL CASES.
 C IOPT = 1 MEANS ONE OR MORE OPTIONAL INPUTS ARE BEING USED.
 C

C RWORK = A WORK ARRAY USED FOR A MIXTURE OF REAL (SINGLE PRECISION)
 C AND INTEGER WORK SPACE.
 C THE LENGTH OF RWORK (IN REAL WORDS) MUST BE AT LEAST
 C $20 + NYH * (\text{MAXORD} + 1) + 3 * \text{NEQ} + \text{LWM}$ WHERE
 C NYH = THE INITIAL VALUE OF NEQ,
 C $\text{MAXORD} = 12$ (IF $\text{METH} = 1$) OR 5 (IF $\text{METH} = 2$) (UNLESS A
 C SMALLER VALUE IS GIVEN AS AN OPTIONAL INPUT),
 C $\text{LWM} = 0$ IF $\text{MITER} = 0$,
 C $\text{LWM} = 2 * \text{NNZ} + 2 * \text{NEQ} + (\text{NNZ} + 9 * \text{NEQ}) / \text{LENRAT}$ IF $\text{MITER} = 1$,
 C $\text{LWM} = 2 * \text{NNZ} + 2 * \text{NEQ} + (\text{NNZ} + 10 * \text{NEQ}) / \text{LENRAT}$ IF $\text{MITER} = 2$,
 C $\text{LWM} = \text{NEQ} + 2$ IF $\text{MITER} = 3$.
 C IN THE ABOVE FORMULAS,
 C NNZ = NUMBER OF NONZERO ELEMENTS IN THE JACOBIAN MATRIX.
 C LENRAT = THE REAL TO INTEGER WORDLENGTH RATIO (USUALLY 1 IN
 C SINGLE PRECISION AND 2 IN DOUBLE PRECISION).
 C (SEE THE MF DESCRIPTION FOR METH AND MITER.)
 C THUS IF MAXORD HAS ITS DEFAULT VALUE AND NEQ IS CONSTANT,
 C THE MINIMUM LENGTH OF RWORK IS..
 C $20 + 16 * \text{NEQ}$ FOR MF = 10,
 C $20 + 16 * \text{NEQ} + \text{LWM}$ FOR MF = 11, 111, 211, 12, 112, 212,
 C $22 + 17 * \text{NEQ}$ FOR MF = 13,
 C $20 + 9 * \text{NEQ}$ FOR MF = 20,
 C $20 + 9 * \text{NEQ} + \text{LWM}$ FOR MF = 21, 121, 221, 22, 122, 222,
 C $22 + 10 * \text{NEQ}$ FOR MF = 23.
 C IF MITER = 1 OR 2, THE ABOVE FORMULA FOR LWM IS ONLY A
 C CRUDE LOWER BOUND. THE REQUIRED LENGTH OF RWORK CANNOT
 C BE READILY PREDICTED IN GENERAL, AS IT DEPENDS ON THE
 C SPARSITY STRUCTURE OF THE PROBLEM. SOME EXPERIMENTATION
 C MAY BE NECESSARY.
 C
 C THE FIRST 20 WORDS OF RWORK ARE RESERVED FOR CONDITIONAL
 C AND OPTIONAL INPUTS AND OPTIONAL OUTPUTS.
 C
 C THE FOLLOWING WORD IN RWORK IS A CONDITIONAL INPUT..
 C $\text{RWORK}(1) = \text{TCRIT}$ = CRITICAL VALUE OF T WHICH THE SOLVER
 C IS NOT TO OVERTHROW. REQUIRED IF ITASK IS
 C 4 OR 5, AND IGNORED OTHERWISE. (SEE ITASK.)
 C
 C LRW = THE LENGTH OF THE ARRAY RWORK, AS DECLARED BY THE USER.
 C (THIS WILL BE CHECKED BY THE SOLVER.)
 C
 C IWORK = AN INTEGER WORK ARRAY. THE LENGTH OF IWORK MUST BE AT LEAST
 C $31 + \text{NEQ} + \text{NNZ}$ IF MOSS = 0 AND MITER = 1 OR 2, OR
 C 30 OTHERWISE.
 C (NNZ IS THE NUMBER OF NONZERO ELEMENTS IN DF/DY.)
 C
 C IN LSODES, IWORK IS USED ONLY FOR CONDITIONAL AND
 C OPTIONAL INPUTS AND OPTIONAL OUTPUTS.
 C
 C THE FOLLOWING TWO BLOCKS OF WORDS IN IWORK ARE CONDITIONAL
 C INPUTS, REQUIRED IF MOSS = 0 AND MITER = 1 OR 2, BUT NOT
 C OTHERWISE (SEE THE DESCRIPTION OF MF FOR MOSS).
 C $\text{IWORK}(30 + J) = \text{IA}(J)$ ($J = 1, \dots, \text{NEQ} + 1$)
 C $\text{IWORK}(31 + \text{NEQ} + K) = \text{JA}(K)$ ($K = 1, \dots, \text{NNZ}$)
 C THE TWO ARRAYS IA AND JA DESCRIBE THE SPARSITY STRUCTURE
 C TO BE ASSUMED FOR THE JACOBIAN MATRIX. JA CONTAINS THE ROW

continues

C INDICES WHERE NONZERO ELEMENTS OCCUR, READING IN COLUMNWISE
 C ORDER, AND IA CONTAINS THE STARTING LOCATIONS IN JA OF THE
 C DESCRIPTIONS OF COLUMNS 1,...,NEQ, IN THAT ORDER, WITH
 C IA(1) = 1. THUS, FOR EACH COLUMN INDEX J = 1,...,NEQ, THE
 C VALUES OF THE ROW INDEX I IN COLUMN J WHERE A NONZERO
 C ELEMENT MAY OCCUR ARE GIVEN BY
 C I = JA(K), WHERE IA(J) .LE. K .LT. IA(J+1).
 C IF NNZ IS THE TOTAL NUMBER OF NONZERO LOCATIONS ASSUMED,
 C THEN THE LENGTH OF THE JA ARRAY IS NNZ, AND IA(NEQ+1) MUST
 C BE NNZ + 1. DUPLICATE ENTRIES ARE NOT ALLOWED.
 C
 C LIW = THE LENGTH OF THE ARRAY IWORK, AS DECLARED BY THE USER.
 C (THIS WILL BE CHECKED BY THE SOLVER.)
 C
 C NOTE.. THE WORK ARRAYS MUST NOT BE ALTERED BETWEEN CALLS TO LSODES
 C FOR THE SAME PROBLEM, EXCEPT POSSIBLY FOR THE CONDITIONAL AND
 C OPTIONAL INPUTS, AND EXCEPT FOR THE LAST 3*NEQ WORDS OF RWORK.
 C THE LATTER SPACE IS USED FOR INTERNAL SCRATCH SPACE, AND SO IS
 C AVAILABLE FOR USE BY THE USER OUTSIDE LSODES BETWEEN CALLS, IF
 C DESIRED (BUT NOT FOR USE BY F OR JAC).
 C
 C JAC = NAME OF USER-SUPPLIED ROUTINE (MITER = 1 OR MOSS = 1) TO
 C COMPUTE THE JACOBIAN MATRIX, DF/DY, AS A FUNCTION OF
 C THE SCALAR T AND THE VECTOR Y. IT IS TO HAVE THE FORM
 C SUBROUTINE JAC (NEQ, T, Y, J, IAN, JAN, PDJ)
 C DIMENSION Y(1), IAN(1), JAN(1), PDJ(1)
 C WHERE NEQ, T, Y, J, IAN, AND JAN ARE INPUT, AND THE ARRAY
 C PDJ, OF LENGTH NEQ, IS TO BE LOADED WITH COLUMN J OF
 C THE JACOBIAN ON OUTPUT. THUS DF(1)/DY(J) IS TO BE
 C LOADED INTO PDJ(1) FOR ALL RELEVANT VALUES OF I. HERE
 C T AND Y HAVE THE SAME MEANING AS IN SUBROUTINE F,
 C AND J IS A COLUMN INDEX (1 TO NEQ). IAN AND JAN ARE
 C UNDEFINED IN CALLS TO JAC FOR STRUCTURE DETERMINATION
 C (MOSS = 1). OTHERWISE, IAN AND JAN ARE STRUCTURE
 C DESCRIPTORS, AS DEFINED UNDER OPTIONAL OUTPUTS BELOW, AND
 C SO CAN BE USED TO DETERMINE THE RELEVANT ROW INDICES I,
 C IF DESIRED. (IN THE DIMENSION STATEMENT ABOVE, 1 IS A
 C DUMMY DIMENSION.. IT CAN BE REPLACED BY ANY VALUE.)
 C JAC NEED NOT PROVIDE DF/DY EXACTLY. A CRUDE
 C APPROXIMATION (POSSIBLY WITH GREATER SPARSITY) WILL DO.
 C IN ANY CASE, PDJ IS PRESET TO ZERO BY THE SOLVER,
 C SO THAT ONLY THE NONZERO ELEMENTS NEED BE LOADED BY JAC.
 C CALLS TO JAC ARE MADE WITH J = 1,...,NEQ, IN THAT ORDER, AND
 C EACH SUCH SET OF CALLS IS PRECEDED BY A CALL TO F WITH THE
 C SAME ARGUMENTS NEQ, T, AND Y. THUS TO GAIN SOME EFFICIENCY,
 C INTERMEDIATE QUANTITIES SHARED BY BOTH CALCULATIONS MAY BE
 C SAVED IN A USER COMMON BLOCK BY F AND NOT RECOMPUTED BY JAC,
 C IF DESIRED. JAC MUST NOT ALTER ITS INPUT ARGUMENTS.
 C JAC MUST BE DECLARED EXTERNAL IN THE CALLING PROGRAM.
 C SUBROUTINE JAC MAY ACCESS USER-DEFINED QUANTITIES IN
 C NEQ(2),... AND Y(NEQ(1)+1),... IF NEQ IS AN ARRAY
 C (DIMENSIONED IN JAC) AND Y HAS LENGTH EXCEEDING NEQ(1).
 C SEE THE DESCRIPTIONS OF NEQ AND Y ABOVE.
 C
 C MF = THE METHOD FLAG. USED ONLY FOR INPUT.
 C MF HAS THREE DECIMAL DIGITS-- MOSS, METH, MITER--
 C MF = 100*MOSS + 10*METH + MITER.
 C MOSS INDICATES THE METHOD TO BE USED TO OBTAIN THE SPARSITY
 C STRUCTURE OF THE JACOBIAN MATRIX IF MITER = 1 OR 2..
 C MOSS = 0 MEANS THE USER HAS SUPPLIED IA AND JA
 C (SEE DESCRIPTIONS UNDER IWORK ABOVE).
 C MOSS = 1 MEANS THE USER HAS SUPPLIED JAC (SEE BELOW)
 C AND THE STRUCTURE WILL BE OBTAINED FROM NEQ
 C INITIAL CALLS TO JAC.

continues

```

C      MOSS = 2 MEANS THE STRUCTURE WILL BE OBTAINED FROM NEQ+1
C      INITIAL CALLS TO F.
C      METH INDICATES THE BASIC LINEAR MULTISTEP METHOD..
C      METH = 1 MEANS THE IMPLICIT ADAMS METHOD.
C      METH = 2 MEANS THE METHOD BASED ON BACKWARD
C          DIFFERENTIATION FORMULAS (BDF-S).
C      MITER INDICATES THE CORRECTOR ITERATION METHOD..
C      MITER = 0 MEANS FUNCTIONAL ITERATION (NO JACOBIAN MATRIX
C          IS INVOLVED).
C      MITER = 1 MEANS CHORD ITERATION WITH A USER-SUPPLIED
C          SPARSE JACOBIAN, GIVEN BY SUBROUTINE JAC.
C      MITER = 2 MEANS CHORD ITERATION WITH AN INTERNALLY
C          GENERATED (DIFFERENCE QUOTIENT) SPARSE JACOBIAN
C          (USING NGP EXTRA CALLS TO F PER DF/DY VALUE,
C          WHERE NGP IS AN OPTIONAL OUTPUT DESCRIBED BELOW.).
C      MITER = 3 MEANS CHORD ITERATION WITH AN INTERNALLY
C          GENERATED DIAGONAL JACOBIAN APPROXIMATION.
C          (USING 1 EXTRA CALL TO F PER DF/DY EVALUATION).
C      IF MITER = 1 OR MOSS = 1, THE USER MUST SUPPLY A SUBROUTINE
C          JAC (THE NAME IS ARBITRARY) AS DESCRIBED ABOVE UNDER JAC.
C          OTHERWISE, A DUMMY ARGUMENT CAN BE USED.
C
C      THE STANDARD CHOICES FOR MF ARE..
C      MF = 10 FOR A NONSTIFF PROBLEM,
C      MF = 21 OR 22 FOR A STIFF PROBLEM WITH IA/JA SUPPLIED
C          (21 IF JAC IS SUPPLIED, 22 IF NOT),
C      MF = 121 FOR A STIFF PROBLEM WITH JAC SUPPLIED,
C          BUT NOT IA/JA,
C      MF = 222 FOR A STIFF PROBLEM WITH NEITHER IA/JA NOR
C          JAC SUPPLIED.
C      THE SPARSENESS STRUCTURE CAN BE CHANGED DURING THE
C          PROBLEM BY MAKING A CALL TO LSODES WITH ISTATE = 3.

```

indicates that the use of LSODES is somewhat more involved than LSODE, as expected, since the accommodation of a sparse matrix is more complicated than that for a full or banded matrix; however, we have found that LSODES is still straightforward to use, and very effective in the NUMOL integration of the ODEs (since the ODE Jacobian matrix is typically sparse, or more precisely, banded and sparse, as we shall see in Chapter 5).

So far, in using the NUMOL, we have encountered only *explicit ODEs*, i.e., each ODE has only one derivative, so that each derivative can be explicitly (directly) expressed mathematically, and can be programmed explicitly in subroutine DERV. However, there are formulations of the NUMOL that lead to *implicit ODEs*, for which more than one derivative appears in a given ODE. Implicit ODEs result, for example, when the spatial derivatives in the PDE(s) are approximated by finite elements or by the method of weighted residuals rather than by finite differences as we have done.

Implicit ODEs can have various mathematical forms. For example

$$\bar{A} \bar{d}\bar{y}/dt = \bar{f}(\bar{y}, t) \quad (4.60)$$

where \bar{A} is a coupling matrix that can give more than one derivative in a given ODE ($\bar{A} \bar{d}\bar{y}/dt$ denotes the usual matrix multiplication between the

$N \times N$ matrix \bar{A} and the N vector $d\bar{y}/dt$). If \bar{A} is the identity matrix, equation (4.60) reduces to equation (4.18) (a system of explicit ODEs).

Equation (4.60) is sometimes termed *linearly implicit* because of the coupling through the operation of matrix multiplication; i.e., the derivatives in any given ODE appear as a linear combination. However, the ODE integrator, LSODI (the “I” denotes implicit), designed for equation (4.60) permits the elements of \bar{A} to be functions of t and \bar{y} (the latter makes the ODEs nonlinear through the multiplying elements of \bar{A}).

The call to RKF45 in Program 1.3d was again replaced with a call to LSODI, as shown in Program 4.4. The output from Program 4.4 is again essentially identical to Table 4.4.

The arguments of LSODI are explained in the user information listed in Table 4.8, taken from the LSODI user’s manual (a set of comments at the beginning of LSODI), so only a few details will be discussed. The principal difference between the calls to subroutines LSODE and LSODI is a third external subroutine for LSODI, e.g., ADDA, which defines the elements of \bar{A} in equation (4.60); in the case of the NUMOL solution of equation (1.21), \bar{A} is the 51×51 identity matrix. Otherwise, the coding in Programs 4.2, 4.3, and 4.4 is similar [in Program 4.4, RES is the subroutine defining the implicit ODEs of equation (4.60), and is analogous to FCN in the calls to LSODE and LSODES in Programs 4.2 and 4.3].

Finally, we should note that if any row of \bar{A} in equation (4.60) contains only zeros, the equation corresponding to that row does not have any derivatives; for example, if we consider row m of \bar{A} to contain only zeros, then the corresponding equation is $f_m(\bar{y}, t) = 0$, i.e., a nonlinear algebraic equation. Of course, for this case, \bar{A} is singular (has a zero determinant), but LSODI can accommodate this case, so LSODI is capable of solving systems of Differential and Algebraic Equations (DAEs) provided they can be expressed through the general format of equation (4.60).

Program 4.4 Call to Subroutine LSODI from Program 1.3d

```
C...
C...  SET THE PARAMETERS FOR SUBROUTINE LSODI
      ITOL=1
      RTOL=ERROR
      ATOL=ERROR
      LRW=7500
      LIW=470
      IOPT=0
      ITASK=1
      ISTATE=1
      MF=22
      TOUT=TV+TP
C...
C...  CALL SUBROUTINE LSODI TO COVER ONE PRINT INTERVAL
4      CALL LSODI(RES,ADDA,JAC,NEQN,YV,YDOT,TV,TOUT,ITOL,RTOL,ATOL,
1                  ITASK,ISTATE,IOPT,RWORK,LRW,IWORK,LIW,MF)
```

Table 4.8 User Information for Subroutine LSODI

C SUMMARY OF USAGE.

C COMMUNICATION BETWEEN THE USER AND THE LSODI PACKAGE, FOR NORMAL SITUATIONS, IS SUMMARIZED HERE. THIS SUMMARY DESCRIBES ONLY A SUBSET OF THE FULL SET OF OPTIONS AVAILABLE. SEE THE FULL DESCRIPTION FOR DETAILS, INCLUDING OPTIONAL COMMUNICATION, NONSTANDARD OPTIONS, AND INSTRUCTIONS FOR SPECIAL SITUATIONS. SEE ALSO THE EXAMPLE PROBLEM (WITH PROGRAM AND OUTPUT) FOLLOWING THIS SUMMARY.

C A. FIRST, PROVIDE A SUBROUTINE OF THE FORM..

```
SUBROUTINE RES (NEQ, T, Y, S, R, IRES)
DIMENSION Y(NEQ), S(NEQ), R(NEQ)
```

C WHICH COMPUTES THE RESIDUAL FUNCTION

```
R = G(T,Y) - A(T,Y) * S,
```

C AS A FUNCTION OF T AND THE VECTORS Y AND S. (S IS AN INTERNALLY GENERATED APPROXIMATION TO DY/DT.) THE ARRAYS Y AND S ARE INPUTS TO THE RES ROUTINE AND SHOULD NOT BE ALTERED. THE RESIDUAL VECTOR C IS TO BE STORED IN THE ARRAY R. THE ARGUMENT IRES SHOULD BE IGNORED FOR CASUAL USE OF LSODI. (FOR USES OF IRES, SEE THE PARAGRAPH ON RES IN THE FULL DESCRIPTION BELOW.)

C B. NEXT, DECIDE WHETHER FULL OR BANDED FORM IS MORE ECONOMICAL FOR THE STORAGE OF MATRICES. LSODI MUST DEAL INTERNALLY WITH THE MATRICES A AND DR/DY, WHERE R IS THE RESIDUAL FUNCTION DEFINED ABOVE. LSODI GENERATES A LINEAR COMBINATION OF THESE TWO MATRICES, AND THIS IS TREATED IN EITHER FULL OR BANDED FORM.

C THE MATRIX STRUCTURE IS COMMUNICATED BY A METHOD FLAG MF, WHICH IS 21 OR 22 FOR THE FULL CASE, AND 24 OR 25 IN THE BAND CASE.

C IN THE BANDED CASE, LSODI REQUIRES TWO HALF-BANDWIDTH PARAMETERS ML AND MU. THESE ARE, RESPECTIVELY, THE WIDTHS OF THE LOWER AND UPPER PARTS OF THE BAND, EXCLUDING THE MAIN DIAGONAL. C THUS THE BAND CONSISTS OF THE LOCATIONS (I,J) WITH C I-ML .LE. J .LE. I+MU, AND THE FULL BANDWIDTH IS ML+MU+1.

C NOTE THAT THE BAND MUST ACCOMMODATE THE NONZERO ELEMENTS OF C A(T,Y), DG/DY, AND D(A*S)/DY (S FIXED). ALTERNATIVELY, ONE CAN C DEFINE A BAND THAT ENCLOSES ONLY THE ELEMENTS THAT ARE RELATIVELY C LARGE IN MAGNITUDE, AND GAIN SOME ECONOMY IN STORAGE AND POSSIBLY C ALSO EFFICIENCY, ALTHOUGH THE APPROPRIATE THRESHOLD FOR RETAINING MATRIX ELEMENTS IS HIGHLY PROBLEM-DEPENDENT.

C C. YOU MUST ALSO PROVIDE A SUBROUTINE OF THE FORM..

```
SUBROUTINE ADDA (NEQ, T, Y, ML, MU, P, NROWP)
DIMENSION Y(NEQ), P(NROWP,NEQ)
```

C WHICH ADDS THE MATRIX A = A(T,Y) TO THE CONTENTS OF THE ARRAY P. C T AND THE Y ARRAY ARE INPUT AND SHOULD NOT BE ALTERED.

C IN THE FULL MATRIX CASE, THIS ROUTINE SHOULD ADD ELEMENTS OF C TO P IN THE USUAL ORDER. I.E., ADD A(I,J) TO P(I,J). (IGNORE THE C ML AND MU ARGUMENTS IN THIS CASE.)

C IN THE BAND MATRIX CASE, THIS ROUTINE SHOULD ADD ELEMENT A(I,J) C TO P(I-J+MU+1,J). I.E., ADD THE DIAGONAL LINES OF A TO THE ROWS OF C P FROM THE TOP DOWN (THE TOP LINE OF A ADDED TO THE FIRST ROW OF P).

C D. FOR THE SAKE OF EFFICIENCY, YOU ARE ENCOURAGED TO SUPPLY THE C JACOBIAN MATRIX DR/DY IN CLOSED FORM, WHERE R = G(T,Y) - A(T,Y)*S C (S = A FIXED VECTOR) AS ABOVE. IF DR/DY IS BEING SUPPLIED, C USE MF = 21 OR 24, AND PROVIDE A SUBROUTINE OF THE FORM..

```
SUBROUTINE JAC (NEQ, T, Y, S, ML, MU, P, NROWP)
DIMENSION Y(NEQ), S(NEQ), P(NROWP,NEQ)
```

C WHICH COMPUTES DR/DY AS A FUNCTION OF T, Y, AND S. HERE T, Y, AND C S ARE INPUTS, AND THE ROUTINE IS TO LOAD DR/DY INTO P AS FOLLOWS..

C IN THE FULL MATRIX CASE (MF = 21), LOAD P(I,J) WITH DR(I)/DY(J),

continues

```

C THE PARTIAL DERIVATIVE OF R(I) WITH RESPECT TO Y(J). (IGNORE THE
C ML AND MU ARGUMENTS IN THIS CASE.)
C IN THE BAND MATRIX CASE (MF = 24), LOAD P(I-J+MU+1,J) WITH
C DR(I)/DY(J), I.E. LOAD THE DIAGONAL LINES OF DR/DY INTO THE ROWS OF
C P FROM THE TOP DOWN.
C IN EITHER CASE, ONLY NONZERO ELEMENTS NEED BE LOADED, AND THE
C INDEXING OF P IS THE SAME AS IN THE ADDA ROUTINE.
C NOTE THAT IF A IS INDEPENDENT OF Y (OR THIS DEPENDENCE
C IS WEAK ENOUGH TO BE IGNORED) THEN JAC IS TO COMPUTE DG/DY.
C IF IT IS NOT FEASIBLE TO PROVIDE A JAC ROUTINE, USE
C MF = 22 OR 25, AND LSODI WILL COMPUTE AN APPROXIMATE JACOBIAN
C INTERNALLY BY DIFFERENCE QUOTIENTS.
C
C E. NEXT DECIDE WHETHER OR NOT TO PROVIDE THE INITIAL VALUE OF THE
C DERIVATIVE VECTOR DY/DT. IF THE INITIAL VALUE OF A(T,Y) IS
C NONSINGULAR (AND NOT TOO ILL-CONDITIONED), YOU MAY LET LSODI COMPUTE
C THIS VECTOR (ISTATE = 0). (LSODI WILL SOLVE THE SYSTEM A*S = G FOR
C S, WITH INITIAL VALUES OF A AND G.) IF A(T,Y) IS INITIALLY
C SINGULAR, THEN THE SYSTEM IS A DIFFERENTIAL-ALGEBRAIC SYSTEM, AND
C YOU MUST MAKE USE OF THE PARTICULAR FORM OF THE SYSTEM TO COMPUTE
C THE INITIAL VALUES OF Y AND DY/DT. IN THAT CASE, USE ISTATE = 1 AND
C LOAD THE INITIAL VALUE OF DY/DT INTO THE ARRAY YDOTI. THE INPUT
C ARRAY YDOTI AND THE INITIAL Y ARRAY MUST BE CONSISTENT WITH
C THE EQUATIONS A*DY/DT = G. THIS IMPLIES THAT THE INITIAL RESIDUAL
C R = G(T,Y) - A(T,Y)*YDOTI MUST BE APPROXIMATELY ZERO.
C
C F. WRITE A MAIN PROGRAM WHICH CALLS SUBROUTINE LSODI ONCE FOR
C EACH POINT AT WHICH ANSWERS ARE DESIRED. THIS SHOULD ALSO PROVIDE
C FOR POSSIBLE USE OF LOGICAL UNIT 6 FOR OUTPUT OF ERROR MESSAGES
C BY LSODI. ON THE FIRST CALL TO LSODI, SUPPLY ARGUMENTS AS FOLLOWS..
C RES   = NAME OF USER SUBROUTINE FOR RESIDUAL FUNCTION R.
C ADDA  = NAME OF USER SUBROUTINE FOR COMPUTING AND ADDING A(T,Y).
C JAC   = NAME OF USER SUBROUTINE FOR JACOBIAN MATRIX DR/DY
C        (MF = 21 OR 24). IF NOT USED, PASS A DUMMY NAME.
C NOTE.. THE NAMES FOR THE RES AND ADDA ROUTINES AND (IF USED) THE
C JAC ROUTINE MUST BE DECLARED EXTERNAL IN THE CALLING PROGRAM.
C NEQ   = NUMBER OF SCALAR EQUATIONS IN THE SYSTEM.
C Y     = ARRAY OF INITIAL VALUES, OF LENGTH NEQ.
C YDOTI = ARRAY OF LENGTH NEQ (CONTAINING INITIAL DY/DT IF ISTATE = 1).
C T     = THE INITIAL VALUE OF THE INDEPENDENT VARIABLE.
C TOUT  = FIRST POINT WHERE OUTPUT IS DESIRED (.NE. T).
C ITOL  = 1 OR 2 ACCORDING AS ATOL (BELOW) IS A SCALAR OR ARRAY.
C RTOL  = RELATIVE TOLERANCE PARAMETER (SCALAR).
C ATOL  = ABSOLUTE TOLERANCE PARAMETER (SCALAR OR ARRAY).
C        THE ESTIMATED LOCAL ERROR IN Y(I) WILL BE CONTROLLED SO AS
C        TO BE ROUGHLY LESS (IN MAGNITUDE) THAN
C          EWT(I) = RTOL*ABS(Y(I)) + ATOL    IF ITOL = 1, OR
C          EWT(I) = RTOL*ABS(Y(I)) + ATOL(I)  IF ITOL = 2.
C        THUS THE LOCAL ERROR TEST PASSES IF, IN EACH COMPONENT,
C        EITHER THE ABSOLUTE ERROR IS LESS THAN ATOL (OR ATOL(I)),
C        OR THE RELATIVE ERROR IS LESS THAN RTOL.
C        USE RTOL = 0.0 FOR PURE ABSOLUTE ERROR CONTROL, AND
C        USE ATOL = 0.0 (OR ATOL(I) = 0.0) FOR PURE RELATIVE ERROR
C        CONTROL. CAUTION.. ACTUAL (GLOBAL) ERRORS MAY EXCEED THESE
C        LOCAL TOLERANCES, SO CHOOSE THEM CONSERVATIVELY.
C ITASK  = 1 FOR NORMAL COMPUTATION OF OUTPUT VALUES OF Y AT T = TOUT.
C ISTATE = INTEGER FLAG (INPUT AND OUTPUT). SET ISTATE = 1 IF THE
C        INITIAL DY/DT IS SUPPLIED, AND 0 OTHERWISE.
C IOPT   = 0 TO INDICATE NO OPTIONAL INPUTS USED.
C RWORK  = REAL WORK ARRAY OF LENGTH AT LEAST..
C           22 + 9*NEQ + NEQ**2      FOR MF = 21 OR 22,
C           22 + 10*NEQ + (2*ML + MU)*NEQ  FOR MF = 24 OR 25.

```

continues

```

C LRW      = DECLARED LENGTH OF RWORK (IN USER-S DIMENSION).
C IWORK    = INTEGER WORK ARRAY OF LENGTH AT LEAST 20 + NEQ.
C          IF MF = 24 OR 25, INPUT IN IWORK(1),IWORK(2) THE LOWER
C          AND UPPER HALF-BANDWIDTHS ML,MU.
C LIW      = DECLARED LENGTH OF IWORK (IN USER-S DIMENSION).
C MF       = METHOD FLAG. STANDARD VALUES ARE..
C          21 FOR A USER-SUPPLIED FULL JACOBIAN.
C          22 FOR AN INTERNALLY GENERATED FULL JACOBIAN.
C          24 FOR A USER-SUPPLIED BANDED JACOBIAN.
C          25 FOR AN INTERNALLY GENERATED BANDED JACOBIAN.
C          FOR OTHER CHOICES OF MF, SEE THE PARAGRAPH ON MF IN
C          THE FULL DESCRIPTION BELOW.
C NOTE THAT THE MAIN PROGRAM MUST DECLARE ARRAYS Y, YDOTI, RWORK, IWORK,
C AND POSSIBLY ATOL.
C
C G. THE OUTPUT FROM THE FIRST CALL (OR ANY CALL) IS..
C     Y = ARRAY OF COMPUTED VALUES OF Y(T) VECTOR.
C     T = CORRESPONDING VALUE OF INDEPENDENT VARIABLE (NORMALLY TOUT).
C ISTATE = 2 IF LSODI WAS SUCCESSFUL, NEGATIVE OTHERWISE.
C          -1 MEANS EXCESS WORK DONE ON THIS CALL (CHECK ALL INPUTS).
C          -2 MEANS EXCESS ACCURACY REQUESTED (TOLERANCES TOO SMALL).
C          -3 MEANS ILLEGAL INPUT DETECTED (SEE PRINTED MESSAGE).
C          -4 MEANS REPEATED ERROR TEST FAILURES (CHECK ALL INPUTS).
C          -5 MEANS REPEATED CONVERGENCE FAILURES (PERHAPS BAD JACOBIAN
C          SUPPLIED OR WRONG CHOICE OF TOLERANCES).
C          -6 MEANS ERROR WEIGHT BECAME ZERO DURING PROBLEM. (SOLUTION
C          COMPONENT I VANISHED, AND ATOL OR ATOL(I) = 0.)
C          -7 CANNOT OCCUR IN CASUAL USE.
C          -8 MEANS LSODI WAS UNABLE TO COMPUTE THE INITIAL DY/DT.
C              IN CASUAL USE, THIS MEANS A(T,Y) IS INITIALLY SINGULAR.
C              SUPPLY YDOTI AND USE ISTATE = 1 ON THE FIRST CALL.
C
C IF LSODI RETURNS ISTATE = -1, -4, OR -5, THEN THE OUTPUT OF
C LSODI ALSO INCLUDES YDOTI = ARRAY CONTAINING RESIDUAL VECTOR
C R = G - A * DY/DT EVALUATED AT THE CURRENT T, Y, AND DY/DT.
C
C H. TO CONTINUE THE INTEGRATION AFTER A SUCCESSFUL RETURN, SIMPLY
C RESET TOUT AND CALL LSODI AGAIN. NO OTHER PARAMETERS NEED BE RESET.

```

DAEs are frequently encountered in scientific and engineering problems, for example, in the NUMOL when the PDE spatial derivatives are approximated by finite elements and the method of weighted residuals (rather than finite differences).

For the final ODE integrator to be considered in this chapter, we consider going one step beyond equation (4.60) to a more general DAE system. The integrator is DASSL, written by Linda R. Petzold (4.9) for this more general DAE problem.

4.8 DASSL

A general first-order DAE system can be written as

$$\bar{f}(\bar{y}, \bar{dy}/dt, t) = \bar{0} \quad (4.61)$$

where \bar{f} is a user-defined vector of functions. A problem unique to equa-

Program 4.5 Call to Subroutine DASSL from Program 1.3d

```

C...
C...  SET THE PARAMETERS FOR SUBROUTINE DASSL
      RTOL=ERROR
      ATOL=ERROR
      LRW=7000
      LIW=475
      DO 7 I=1,15
      INFO(I)=0
7     CONTINUE
      TOUT=TV+TP
C...
C...  CALL SUBROUTINE DASSL TO COVER ONE PRINT INTERVAL
4     CALL DASSL(RES,NEQN,TV,YV,YPRIME,TOUT,INFO,RTOL,ATOL,IDL,
1           RWORK,LRW,IWORK,LIW,RPAR,IPAR,JAC)

```

tion (4.61) that we did not encounter previously is the requirement to specify a consistent set of initial conditions; e.g., for an initial time $t = t_0$, what are the values of $\bar{y}(t_0)$ and $d\bar{y}(t_0)/dt$ that satisfy equation (4.61). We shall only mention this problem of specifying initial conditions; a detailed discussion is given by Brenan et al. (4.9).

The call to RKF45 in Program 1.3d was replaced with the call to DASSL listed in Program 4.5. The output from Program 4.5 is again essentially identical to Table 4.4.

A portion of the user's manual from the beginning of subroutine DASSL is given in Table 4.9. As expected, a user-supplied subroutine, RES, is required to define the vector of functions \bar{f} in equation (4.61). Also, DASSL is based on the BDFs for stiff DAEs, and therefore a user-supplied subroutine for the Jacobian matrix (4.21) can be provided, or an option can be selected for an internally generated finite difference Jacobian matrix.

Table 4.9 User Information for Subroutine DASSL

```

C *Arguments:
C
C   RES:EXT  This is a subroutine which you provide to define the
C             differential/algebraic system.
C
C   NEQ:IN   This is the number of equations to be solved.
C
C   T:INOUT  This is the current value of the independent variable.
C
C   Y(*) :INOUT This array contains the solution components at T.
C
C   YPRIME(*) :INOUT This array contains the derivatives of the solution
C                     components at T.
C
C   TOUT:IN   This is a point at which a solution is desired.

```

continues

```

C
C INFO(N):IN  The basic task of the code is to solve the system from T
C          to TOUT and return an answer at TOUT.  INFO is an integer
C          array which is used to communicate exactly how you want
C          this task to be carried out.  N must be greater than or
C          equal to 15.
C
C RTOL,ATOL:INOUT These quantities represent absolute and relative
C                    error tolerances which you provide to indicate how
C                    accurately you wish the solution to be computed.
C                    You may choose them to be both scalars or else
C                    both vectors.
C
C IDID:OUT This scalar quantity is an indicator reporting what the
C            code did.  You must monitor this integer variable to decide
C            what action to take next.
C
C RWORK:WORK A real work array of length LRW which provides the
C            code with needed storage space.
C
C LRW:IN The length of RWORK.
C
C IWORK:WORK An integer work array of length LIW which provides the
C            code with needed storage space.
C
C LIW:IN The length of IWORK.
C
C RPAR,IPAR:IN These are real and integer parameter arrays which
C                  you can use for communication between your calling
C                  program and the RES subroutine (and the JAC subroutine)
C
C JAC:EXT This is the name of a subroutine which you may choose to
C           provide for defining a matrix of partial derivatives
C           described below.
C
C
C *Description
C QUANTITIES WHICH MAY BE ALTERED BY THE CODE ARE
C     T,Y(*),YPRIME(*),INFO(1),RTOL,ATOL,IDID,RWORK(*) AND IWORK(*)
C
C Subroutine SDASSL uses the backward differentiation formulas of
C orders one through five to solve a system of the above form for Y and
C YPRIME.  Values for Y and YPRIME at the initial time must be given as
C input.  These values must be consistent, (that is, if T,Y,YPRIME are
C the given initial values, they must satisfy G(T,Y,YPRIME) = 0.).  The
C subroutine solves the system from T to TOUT.  It is easy to continue
C the solution to get results at additional TOUT.  This is the interval
C mode of operation.  Intermediate results can also be obtained easily
C by using the intermediate-output capability.
C
C -----INPUT-WHAT TO DO ON THE FIRST CALL TO SDASSL-----
C
C
C The first call of the code is defined to be the start of each new
C problem.  Read through the descriptions of all the following items,
C provide sufficient storage space for designated arrays, set
C appropriate variables for the initialization of the problem, and
C give information about how you want the problem to be solved.
C
C
C RES -- Provide a subroutine of the form
C           SUBROUTINE RES(T,Y,YPRIME,DELTA,IRES,RPAR,IPAR)

```

continues

C to define the system of differential/algebraic
C equations which is to be solved. For the given values
C of T,Y and YPRIME, the subroutine should
C return the residual of the differential/algebraic
C system
C DELTA = G(T,Y,YPRIME)
C (DELTA(*) is a vector of length NEQ which is output
C for RES.)
C
C Subroutine RES must not alter T,Y or YPRIME.
C You must declare the name RES in an external
C statement in your program that calls SDASSL.
C You must dimension Y,YPRIME and DELTA in RES.
C
C IRES is an integer flag which is always equal to
C zero on input. Subroutine RES should alter IRES
C only if it encounters an illegal value of Y or
C a stop condition. Set IRES = -1 if an input value
C is illegal, and SDASSL will try to solve the problem
C without getting IRES = -1. If IRES = -2, SDASSL
C will return control to the calling program
C with IDID = -11.
C
C RPAR and IPAR are real and integer parameter arrays which
C you can use for communication between your calling program
C and subroutine RES. They are not altered by SDASSL. If you
C do not need RPAR or IPAR, ignore these parameters by treating
C them as dummy arguments. If you do choose to use them,
C dimension them in your calling program and in RES as arrays
C of appropriate length.
C
C NEQ -- Set it to the number of differential equations.
C (NEQ .GE. 1)
C
C T -- Set it to the initial point of the integration.
C T must be defined as a variable.
C
C Y(*) -- Set this vector to the initial values of the NEQ solution
C components at the initial point. You must dimension Y of
C length at least NEQ in your calling program.
C
C YPRIME(*) -- Set this vector to the initial values of
C the NEQ first derivatives of the solution
C components at the initial point. You
C must dimension YPRIME at least NEQ
C in your calling program. If you do not
C know initial values of some of the solution
C components, see the explanation of INFO(11).
C
C TOUT - Set it to the first point at which a solution
C is desired. You can not take TOUT = T. Inte-
C gration either forward in T (TOUT .GT. T) or
C backward in T (TOUT .LT. T) is permitted.
C
C The code advances the solution from T to TOUT using
C step sizes which are automatically selected so as to
C achieve the desired accuracy. If you wish, the code
C will return with the solution and its derivative at
C intermediate steps (intermediate-output mode) so that
C you can monitor them, but you still must provide TOUT
C in accord with the basic aim of the code.

continues

C
C The first step taken by the code is a critical one
C because it must reflect how fast the solution changes near
C the initial point. The code automatically selects an
C initial step size which is practically always suitable for
C the problem. By using the fact that the code will not step
C past TOUT in the first step, you could, if necessary,
C restrict the length of the initial step size.
C
C For some problems it may not be permissible to integrate
C past a point TSTOP because a discontinuity occurs there
C or the solution or its derivative is not defined beyond
C TSTOP. When you have declared a TSTOP point (SEE INFO(4)
C and RWWORK(1)), you have told the code not to integrate
C past TSTOP. In this case any TOUT beyond TSTOP is invalid
C input.
C
C INFO(*) - Use the INFO array to give the code more details about
C how you want your problem solved. This array should be
C dimensioned of length 15, though SDASSL uses only
C the first eleven entries. You must respond to all of
C the following items which are arranged as questions. The
C simplest use of the code corresponds to answering all
C questions as yes, i.e. setting all entries of INFO to 0.
C
C INFO(1) - This parameter enables the code to initialize
C itself. You must set it to indicate the start of
C every new problem.
C
C ***** Is this the first call for this problem ...
C Yes - Set INFO(1) = 0
C No - Not applicable here.
C See below for continuation calls. *****
C
C INFO(2) - How much accuracy you want of your solution
C is specified by the error tolerances RTOL and ATOL.
C The simplest use is to take them both to be scalars.
C To obtain more flexibility, they can both be vectors.
C The code must be told your choice.
C
C ***** Are both error tolerances RTOL, ATOL scalars ...
C Yes - Set INFO(2) = 0
C and input scalars for both RTOL and ATOL
C No - Set INFO(2) = 1
C and input arrays for both RTOL and ATOL *****
C
C INFO(3) - The code integrates from T in the direction
C of TOUT by steps. If you wish, it will return the
C computed solution and derivative at the next
C intermediate step (the intermediate-output mode) or
C TOUT, whichever comes first. This is a good way to
C proceed if you want to see the behavior of the solution.
C If you must have solutions at a great many specific
C TOUT points, this code will compute them efficiently.
C
C ***** Do you want the solution only at
C TOUT (and not at the next intermediate step) ...
C Yes - Set INFO(3) = 0
C No - Set INFO(3) = 1 *****
C
C INFO(4) - To handle solutions at a great many specific

continues

C values TOUT efficiently, this code may integrate past
 C TOUT and interpolate to obtain the result at TOUT.
 C Sometimes it is not possible to integrate beyond some
 C point TSTOP because the equation changes there or it is
 C not defined past TSTOP. Then you must tell the code
 C not to go past.
 C

C **** Can the integration be carried out without any
 C restrictions on the independent variable T ...
 C Yes - Set INFO(4)=0
 C No - Set INFO(4)=1
 C and define the stopping point TSTOP by
 C setting RWORK(1)=TSTOP ****
 C

C INFO(5) - To solve differential/algebraic problems it is
 C necessary to use a matrix of partial derivatives of the
 C system of differential equations. If you do not
 provide a subroutine to evaluate it analytically (see
 C description of the item JAC in the call list), it will
 C be approximated by numerical differencing in this code.
 C Although it is less trouble for you to have the code
 C compute partial derivatives by numerical differencing,
 the solution will be more reliable if you provide the
 C derivatives via JAC. Sometimes numerical differencing
 C is cheaper than evaluating derivatives in JAC and
 sometimes it is not - this depends on your problem.
 C

C **** Do you want the code to evaluate the partial
 C derivatives automatically by numerical differences ...
 C Yes - Set INFO(5)=0
 C No - Set INFO(5)=1
 C and provide subroutine JAC for evaluating the
 C matrix of partial derivatives ****
 C

C INFO(6) - SDASSL will perform much better if the matrix of
 C partial derivatives, DG/DY + CJ*DGYPRIME,
 (here CJ is a scalar determined by SDASSL)
 C is banded and the code is told this. In this
 case, the storage needed will be greatly reduced,
 C numerical differencing will be performed much cheaper,
 and a number of important algorithms will execute much
 C faster. The differential equation is said to have
 half-bandwidths ML (lower) and MU (upper) if equation i
 C involves only unknowns Y(J) with
 I-ML .LE. J .LE. I+MU
 C for all I=1,2,...,NEQ. Thus, ML and MU are the widths
 C of the lower and upper parts of the band, respectively,
 with the main diagonal being excluded. If you do not
 indicate that the equation has a banded matrix of partial
 C derivatives, the code works with a full matrix of NEQ**2
 elements (stored in the conventional way). Computations
 with banded matrices cost less time and storage than with
 C full matrices if 2*ML+MU .LT. NEQ. If you tell the code
 that the matrix of partial derivatives has a banded
 structure and you want to provide subroutine JAC to
 compute the partial derivatives, then you must be careful
 to store the elements of the matrix in the special form
 indicated in the description of JAC.
 C

C **** Do you want to solve the problem using a full
 C (dense) matrix (and not a special banded
 structure) ...
 C Yes - Set INFO(6)=0

continues

```

C          No - Set INFO(6)=1
C                  and provide the lower (ML) and upper (MU)
C                  bandwidths by setting
C                  IWORK(1)=ML
C                  IWORK(2)=MU *****
C
C          INFO(7) -- You can specify a maximum (absolute value of)
C                  stepsize, so that the code
C                  will avoid passing over very
C                  large regions.
C
C          ***** Do you want the code to decide
C                  on its own maximum stepsize?
C          Yes - Set INFO(7)=0
C          No - Set INFO(7)=1
C                  and define HMAX by setting
C                  RWORK(2)=HMAX *****
C
C          INFO(8) -- Differential/algebraic problems
C                  may occasionally suffer from
C                  severe scaling difficulties on the
C                  first step. If you know a great deal
C                  about the scaling of your problem, you
C                  can help to alleviate this problem by
C                  specifying an initial stepsize H0.
C
C          ***** Do you want the code to define
C                  its own initial stepsize?
C          Yes - Set INFO(8)=0
C          No - Set INFO(8)=1
C                  and define H0 by setting
C                  RWORK(3)=H0 *****
C
C          INFO(9) -- If storage is a severe problem,
C                  you can save some locations by
C                  restricting the maximum order MAXORD.
C                  the default value is 5. for each
C                  order decrease below 5, the code
C                  requires NEQ fewer locations, however
C                  it is likely to be slower. In any
C                  case, you must have 1 .LE. MAXORD .LE. 5
C
C          ***** Do you want the maximum order to
C                  default to 5?
C          Yes - Set INFO(9)=0
C          No - Set INFO(9)=1
C                  and define MAXORD by setting
C                  IWORK(3)=MAXORD *****
C
C          INFO(10) --If you know that the solutions to your
C                  equations will always be nonnegative, it may
C                  help to set this parameter. However, it
C                  is probably best to try the code without
C                  using this option first, and only to use
C                  this option if that doesn't work very well.
C
C          ***** Do you want the code to solve the problem without
C                  invoking any special nonnegativity constraints?
C          Yes - Set INFO(10)=0
C          No - Set INFO(10)=1

```

continues

```

C
C      INFO(11) --SDASSL normally requires the initial T,
C      Y, and YPRIME to be consistent. That is,
C      you must have G(T,Y,YPRIME) = 0 at the initial
C      time. If you do not know the initial
C      derivative precisely, you can let SDASSL try
C      to compute it.
C
C      ***** Are the initial T, Y, YPRIME consistent?
C      Yes - Set INFO(11) = 0
C      No - Set INFO(11) = 1,
C              and set YPRIME to an initial approximation
C              to YPRIME. (If you have no idea what
C              YPRIME should be, set it to zero. Note
C              that the initial Y should be such
C              that there must exist a YPRIME so that
C              G(T,Y,YPRIME) = 0.)
C
C      RTOL, ATOL -- You must assign relative (RTOL) and absolute (ATOL)
C      error tolerances to tell the code how accurately you
C      want the solution to be computed. They must be defined
C      as variables because the code may change them. You
C      have two choices --
C          Both RTOL and ATOL are scalars. (INFO(2)=0)
C          Both RTOL and ATOL are vectors. (INFO(2)=1)
C      in either case all components must be non-negative.
C
C      The tolerances are used by the code in a local error
C      test at each step which requires roughly that
C          ABS(LOCAL ERROR) .LE. RTOL*ABS(Y)+ATOL
C      for each vector component.
C      (More specifically, a root-mean-square norm is used to
C      measure the size of vectors, and the error test uses the
C      magnitude of the solution at the beginning of the step.)
C
C      The true (global) error is the difference between the
C      true solution of the initial value problem and the
C      computed approximation. Practically all present day
C      codes, including this one, control the local error at
C      each step and do not even attempt to control the global
C      error directly.
C
C      Usually, but not always, the true accuracy of the
C      computed Y is comparable to the error tolerances. This
C      code will usually, but not always, deliver a more
C      accurate solution if you reduce the tolerances and
C      integrate again. By comparing two such solutions you
C      can get a fairly reliable idea of the true error in the
C      solution at the bigger tolerances.
C
C      Setting ATOL=0. results in a pure relative error test on
C      that component. Setting RTOL=0. results in a pure
C      absolute error test on that component. A mixed test
C      with non-zero RTOL and ATOL corresponds roughly to a
C      relative error test when the solution component is much
C      bigger than ATOL and to an absolute error test when the
C      solution component is smaller than the threshold ATOL.
C
C      The code will not attempt to compute a solution at an
C      accuracy unreasonable for the machine being used. It
C      will advise you if you ask for too much accuracy and
C      inform you as to the maximum accuracy it believes
C      possible.

```

continues

```

C
C RWORK(*) -- Dimension this real work array of length LRW in your
C calling program.
C
C LRW -- Set it to the declared length of the RWORK array.
C     You must have
C         LRW .GE. 40+(MAXORD+4)*NEQ+NEQ**2
C     for the full (dense) JACOBIAN case (when INFO(6)=0), or
C         LRW .GE. 40+(MAXORD+4)*NEQ+(2*ML+MU+1)*NEQ
C     for the banded user-defined JACOBIAN case
C     (when INFO(5)=1 and INFO(6)=1), or
C         LRW .GE. 40+(MAXORD+4)*NEQ+(2*ML+MU+1)*NEQ
C             +2*(NEQ/(ML+MU+1)+1)
C     for the banded finite-difference-generated JACOBIAN case
C     (when INFO(5)=0 and INFO(6)=1)
C
C IWORK(*) -- Dimension this integer work array of length LIW in
C your calling program.
C
C LIW -- Set it to the declared length of the IWORK array.
C     you must have LIW .GE. 20+NEQ
C
C RPAR, IPAR -- These are parameter arrays, of real and integer
C type, respectively. You can use them for communication
C between your program that calls SDASSL and the RES
C subroutine (and the JAC subroutine). They are not
C altered by SDASSL. If you do not need RPAR or IPAR,
C ignore these parameters by treating them as dummy
C arguments. If you do choose to use them, dimension
C them in your calling program and in RES (and in JAC)
C as arrays of appropriate length.
C
C JAC -- If you have set INFO(5)=0, you can ignore this parameter by
C treating it as a dummy argument. Otherwise, you must
C provide a subroutine of the form
C     JAC(T,Y,YPRIME,PD,CJ,RPAR,IPAR)
C to define the matrix of partial derivatives
C     PD=DG/DY+CJ*DGYPRIME
C     CJ is a scalar which is input to JAC.
C     For the given values of T,Y,YPRIME, the
C     subroutine must evaluate the non-zero partial
C     derivatives for each equation and each solution
C     component, and store these values in the
C     matrix PD. The elements of PD are set to zero
C     before each call to JAC so only non-zero elements
C     need to be defined.
C
C Subroutine JAC must not alter T,Y,(*),YPRIME(*),
C or CJ. You must declare the name JAC in an
C EXTERNAL STATEMENT in your program that calls
C SDASSL. You must dimension Y, YPRIME and PD
C in JAC.
C
C The way you must store the elements into the PD matrix
C depends on the structure of the matrix which you
C indicated by INFO(6).
C     *** INFO(6)=0 -- Full (dense) matrix ***
C     Give PD a first dimension of NEQ.
C     When you evaluate the (non-zero) partial derivative
C     of equation I with respect to variable J, you must
C     store it in PD according to
C     PD(I,J) = * DG(I)/DY(J)+CJ*DGYPRIME(I)/DGYPRIME(J)*
C     *** INFO(6)=1 -- Banded JACOBIAN with ML lower and MU

```

continues

```

C           upper diagonal bands (refer to INFO(6) description
C           of ML and MU) ***
C           Give PD a first dimension of 2*ML+MU+1.
C           when you evaluate the (non-zero) partial derivative
C           of equation I with respect to variable J, you must
C           store it in PD according to
C           IROW = I - J + ML + MU + 1
C           PD(IROW,J) = *DG(I)/DY(J)+CJ*DG(I)/DYPRIIME(J)*
C
C           RPAR and IPAR are real and integer parameter arrays
C           which you can use for communication between your calling
C           program and your JACOBIAN subroutine JAC. They are not
C           altered by SDASSL. If you do not need RPAR or IPAR,
C           ignore these parameters by treating them as dummy
C           arguments. If you do choose to use them, dimension
C           them in your calling program and in JAC as arrays of
C           appropriate length.
C
C
C
C           OPTIONAL REPLACEABLE NORM ROUTINE:
C           SDASSL uses a weighted norm SDANRM to measure the size
C           of vectors such as the estimated error in each step.
C           A FUNCTION subprogram
C           REAL FUNCTION SDANRM(NEQ,V,WT,RPAR,IPAR)
C           DIMENSION V(NEQ),WT(NEQ)
C           is used to define this norm. Here, V is the vector
C           whose norm is to be computed, and WT is a vector of
C           weights. A SDANRM routine has been included with SDASSL
C           which computes the weighted root-mean-square norm
C           given by
C           SDANRM=SQRT((1/NEQ)*SUM(V(I)/WT(I))**2)
C           this norm is suitable for most problems. In some
C           special cases, it may be more convenient and/or
C           efficient to define your own norm by writing a function
C           subprogram to be called instead of SDANRM. This should,
C           however, be attempted only after careful thought and
C           consideration.
C
C
C-----OUTPUT-AFTER ANY RETURN FROM SDASSL-----
C
C           The principal aim of the code is to return a computed solution at
C           TOUT, although it is also possible to obtain intermediate results
C           along the way. To find out whether the code achieved its goal or
C           if the integration process was interrupted before the task was
C           completed, you must check the IDID parameter.
C
C
C           T -- The solution was successfully advanced to the
C               output value of T.
C
C           Y(*) -- Contains the computed solution approximation at T.
C
C           YPRIME(*) -- Contains the computed derivative
C               approximation at T.
C
C           IDID -- Reports what the code did.
C
C           *** Task completed ***
C           Reported by positive values of IDID
C
C           IDID = 1 -- A step was successfully taken in the
C               intermediate-output mode. The code has not
C               yet reached TOUT.

```

continues

```

C
C      IDID = 2 -- The integration to TOUT was successfully
C                  completed (T=TOUT) by stepping exactly to TOUT.
C
C      IDID = 3 -- The integration to TOUT was successfully
C                  completed (T=TOUT) by stepping past TOUT.
C                  Y(*) is obtained by interpolation.
C                  YPRIME(*) is obtained by interpolation.
C
C      *** Task interrupted ***
C      Reported by negative values of IDID
C
C      IDID = -1 -- A large amount of work has been expended.
C                  (About 500 steps)
C
C      IDID = -2 -- The error tolerances are too stringent.
C
C      IDID = -3 -- The local error test cannot be satisfied
C                  because you specified a zero component in ATOL
C                  and the corresponding computed solution
C                  component is zero. Thus, a pure relative error
C                  test is impossible for this component.
C
C      IDID = -6 -- SDASSL had repeated error test
C                  failures on the last attempted step.
C
C      IDID = -7 -- The corrector could not converge.
C
C      IDID = -8 -- The matrix of partial derivatives
C                  is singular.
C
C      IDID = -9 -- The corrector could not converge.
C                  There were repeated error test failures
C                  in this step.
C
C      IDID = -10 -- The corrector could not converge
C                  because IRES was equal to minus one.
C
C      IDID = -11 -- IRES equal to -2 was encountered
C                  and control is being returned to the
C                  calling program.
C
C      IDID = -12 -- SDASSL failed to compute the initial
C                  YPRIME.
C
C      IDID = -13,...,-32 -- Not applicable for this code
C
C      *** Task terminated ***
C      Reported by the value of IDID=-33
C
C      IDID = -33 -- The code has encountered trouble from which
C                  it cannot recover. A message is printed explaining
C                  the trouble and control is returned to the calling
C                  program. For example, this occurs when invalid input
C                  is detected.
C
C      RTOL, ATOL -- These quantities remain unchanged except when
C                  IDID = -2. In this case, the error tolerances have been
C                  increased by the code to values which are estimated to
C                  be appropriate for continuing the integration. However,
C                  the reported solution at T was obtained using the input
C                  values of RTOL and ATOL.

```

continues

```

C
C RWORK, IWORK -- Contain information which is usually of no interest
C           to the user but necessary for subsequent calls. However,
C           you may find use for
C
C RWORK(3) -- Which contains the step size H to be attempted
C           on the next step.
C
C RWORK(4) -- Which contains the current value of the
C           independent variable, i.e., the farthest point
C           integration has reached. This will be different
C           from T only when interpolation has been
C           performed (IDID=3).
C
C RWORK(7) -- Which contains the stepsize used on the last
C           successful step.
C
C IWORK(7) -- Which contains the order of the method to
C           be attempted on the next step.
C
C IWORK(8) -- Which contains the order of the method used
C           on the last step.
C
C IWORK(11) -- Which contains the number of steps taken so
C           far.
C
C IWORK(12) -- Which contains the number of calls to RES
C           so far.
C
C IWORK(13) -- Which contains the number of evaluations of
C           the matrix of partial derivatives needed so
C           far.
C
C IWORK(14) -- Which contains the total number of error
C           test failures so far.
C
C IWORK(15) -- Which contains the total number of conver-
C           gence test failures so far (includes singular
C           iteration matrix failures).
C
C
C INPUT -- What to do to continue the integration
C           (calls after the first)          **
C
C This code is organized so that subsequent calls to continue the
C integration involve little (if any) additional effort on your
C part. You must monitor the IDID parameter in order to determine
C what to do next.
C
C Recalling that the principal task of the code is to integrate
C from T to TOUT (the interval mode), usually all you will need
C to do is specify a new TOUT upon reaching the current TOUT.
C
C Do not alter any quantity not specifically permitted below,
C in particular do not alter NEQ,T,Y(*),YPRIME(*),RWORK(*),IWORK(*)
C or the differential equation in subroutine RES. Any such
C alteration constitutes a new problem and must be treated as such,
C i.e., you must start afresh.
C
C You cannot change from vector to scalar error control or vice
C versa (INFO(2)), but you can change the size of the entries of

```

continues

C RTOL, ATOL. Increasing a tolerance makes the equation easier
C to integrate. Decreasing a tolerance will make the equation
C harder to integrate and should generally be avoided.
C

C You can switch from the intermediate-output mode to the interval
C mode (INFO(3)) or vice versa at any time.
C

C If it has been necessary to prevent the integration from going
C past a point TSTOP (INFO(4), RWORK(1)), keep in mind that the
C code will not integrate to any TOUT beyond the currently
C specified TSTOP. Once TSTOP has been reached you must change
C the value of TSTOP or set INFO(4)=0. You may change INFO(4)
C or TSTOP at any time but you must supply the value of TSTOP in
C RWORK(1) whenever you set INFO(4)=1.
C

C Do not change INFO(5), INFO(6), IWORK(1), or IWORK(2)
C unless you are going to restart the code.
C

C *** Following a completed task ***
C If
C IDID = 1, call the code again to continue the integration
C another step in the direction of TOUT.
C

C IDID = 2 or 3, define a new TOUT and call the code again.
C TOUT must be different from T. You cannot change
C the direction of integration without restarting.
C

C *** Following an interrupted task ***
C To show the code that you realize the task was
C interrupted and that you want to continue, you
C must take appropriate action and set INFO(1) = 1
C If:
C

C IDID = -1, The code has taken about 500 steps. If you want
C to continue, set INFO(1) = 1 and call the code
C again. An additional 500 steps will be allowed.
C

C IDID = -2, The error tolerances RTOL, ATOL have been
C increased to values the code estimates appropriate
C for continuing. You may want to change them
C yourself. If you are sure you want to continue
C with relaxed error tolerances, set INFO(1)=1 and call
C the code again.
C

C IDID = -3, A solution component is zero and you set the
C corresponding component of ATOL to zero. If you
C are sure you want to continue, you must first
C alter the error criterion to use positive values
C for those components of ATOL corresponding to zero
C solution components, then set INFO(1)=1 and call
C the code again.
C

C IDID = -4,-5 --- Cannot occur with this code.
C

C IDID = -6, Repeated error test failures occurred on the last
C attempted step in SDASSL. A singularity in the
C solution may be present. If you are absolutely
C certain you want to continue, you should restart
C the integration. (Provide initial values of Y and
C YPRIME which are consistent)

continues

```

C
C      IDID = -7, Repeated convergence test failures occurred on the
C          last attempted step in SDASSL. An inaccurate or
C          ill-conditioned JACOBIAN may be the problem. If
C          you are absolutely certain you want to continue,
C          you should restart the integration.
C
C      IDID = -8, The matrix of partial derivatives is singular. Some
C          of your equations may be redundant. SDASSL cannot
C          solve the problem as stated. It is possible that
C          the redundant equations could be removed, and then
C          SDASSL could solve the problem. It is also possible
C          that a solution to your problem either does not
C          exist or is not unique.
C
C      IDID = -9, SDASSL had multiple convergence test failures, preceded
C          by multiple error test failures, on the last attempted
C          step. It is possible that your problem is ill-posed,
C          and cannot be solved using this code. Or, there may be
C          a discontinuity or a singularity in the solution. If
C          you are absolutely certain you want to continue, you
C          should restart the integration.
C
C      IDID =-10, SDASSL had multiple convergence test failures because
C          IRES was equal to minus one. If you are absolutely
C          certain you want to continue, you should restart the
C          integration.
C
C      IDID =-11, IRES=-2 was encountered, and control is being returned
C          to the calling program.
C
C      IDID =-12, SDASSL failed to compute the initial YPRIME. This could
C          happen because the initial approximation to YPRIME
C          was not very good, or if a YPRIME consistent with the
C          initial Y does not exist. The problem could also be
C          caused by an inaccurate or singular iteration matrix.
C
C      IDID =-13,...,-32 --- Cannot occur with this code.
C
C          *** Following a terminated task ***
C      IDID= -33, You cannot continue the solution of this problem. An
C          attempt to do so will result in your run being term-
C          inated.
C

```

The development of integrators for ODEs and DAEs continues to be an active area of research, and new results are appearing regularly; see, for example, Brenan, et al (4.9) and Byrne and Hindmarsh (4.6). The more notable developments include the implementation of the BDF without requiring the storage of matrices, the so-called matrix-free methods developed by Brown and Hindmarsh (4.10, 4.11, 4.12), and the variable-order code, VODE, developed by Byrne and Hindmarsh (4.13). Because of the limitations of space in this book, we shall not discuss these developments. However, they offer the promise of greater generality and efficiency in the solution of the ODEs and DAEs resulting from the NUMOL solution of PDEs.

In this chapter, we have briefly surveyed the use of several quality ODE and DAE solvers, with particular application to the NUMOL. We have considered the stability of the numerical integration of general sets of differential equations, and the use of implicit methods, particularly BDF methods, to enhance the stability of the numerical integration. In the next chapter, we again consider stability, but with particular emphasis on the differential equations that result from the NUMOL approximation of PDEs. The purpose of this analysis is to point out that the formulation of some NUMOL approximations can lead to significant stability problems, and therefore, some care is required in selecting the approximations of the PDE spatial derivatives.

References

- (4.1) Kahaner, D., C. Moler, and S. Nash, *Numerical Methods and Software*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
- (4.2) Lapidus, L., and J. H. Seinfeld, *Numerical Solution of Ordinary Differential Equations*, Academic Press, New York, 1971.
- (4.3) Ortega, J. M., and W. G. Poole, Jr., *An Introduction to Numerical Methods for Differential Equations*, Pitman Publishing, Marshfield, MA, 1981.
- (4.4) Shampine, L. F., and M. K. Gordon, *Computer Solution of Ordinary Differential Equations. The Initial Value Problem*, Freeman, San Francisco, 1975.
- (4.5) Gear, C. W., *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.
- (4.6) Byrne, G. D., and A. C. Hindmarsh, "Stiff ODE Solvers: A Review of Current and Coming Attractions," *J. Comput. Phys.*, 70, 1–62 (1987).
- (4.7) Hindmarsh, A. C., "ODE Solvers for Time-Dependent PDE Software," UCRL-89311 Preprint, Lawrence Livermore National Laboratory, Livermore, CA, June, 1983; see also *PDE Software: Modules, Interfaces and Systems*, B. Enquist and T. Smedsaas, eds., Elsevier Science Publishers, B. V. North-Holland, Amsterdam 1984, pp. 325–341.
- (4.8) Hindmarsh, A. C., "ODEPACK, A Systematized Collection of ODE Solvers," in *Scientific Computing*, R. S. Stepleman et al., eds., North-Holland, Amsterdam 1983, pp. 55–64.
- (4.9) Brenan, K. E., S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, North-Holland, New York, 1989.
- (4.10) Brown, Peter N., and Alan C. Hindmarsh, "Matrix-free Methods for Stiff Systems of ODEs," *Siam J. Numer. Anal.*, 23, (3), 610–638, (June 1986).
- (4.11) Brown, Peter N., and Alan C. Hindmarsh, "Reduced Storage Matrix Methods in Stiff ODE Systems," *J. Appl. Math. Comput.*, 31, 40–91 (1989).
- (4.12) Hindmarsh, Alan, and Peter N. Brown, "Reduced-storage Techniques in the Numerical Method of Lines," in *Proceedings of the Sixth IMACS Symposium on Computer Methods for PDEs*, R. Vichnevetsky and R. Stepleman, eds., Lehigh University, Bethlehem, PA, June 23–26, 1987.
- (4.13) Brown, Peter N., George D. Byrne and Alan C. Hindmarsh, "VODE, A Variable-coefficient ODE Solver," *SIAM J. Sci. Stat. Comput.*, 10, 1038–1051 (1989).

Problems

- (4.1) Verify that equations (4.10) and (4.11) are solutions to equations (4.1) and (4.2) [keep in mind equation (4.9) for each eigenvalue, λ_1 and λ_2].
- (4.2) Verify that equations (4.29) and (4.30) are solutions to equations (4.23) and (4.24) [keep in mind equation (4.28) for each eigenvalue, β_1 and β_2].
- (4.3) Obtain the *specific solution* from general solution (4.38) and (4.39) (i.e., compute the constants C_{11} to C_{22}) for the initial conditions $y_1(0) = 0$, $y_2(0) = 2$. Plot the solution as $y_1(t)$ and $y_2(t)$ vs. $\log_{10}t$ for $t = 10^{-9}, 10^{-8}, \dots, 10^0, 10^1$.
- (4.4) Use subroutine SDRV2 to solve equations (4.1) and (4.2) for $a_{11} = a_{22} = -a$, $a_{12} = a_{21} = b$, $a = 500000.5$, $b = 499999.5$, $y_1(0) = 0$, $y_2(0) = 2$. Print the numerical solution (from SDRV2), the exact solution (from Problem 4.3) and the difference between these solutions vs. t for $t = 10^{-9}, 10^{-8}, \dots, 10^0, 10^1$. Also, at a final value of t , print WORK(1) to WORK(3) and IWORK(1) to IWORK(5) (with labels suggested by the following excerpt from the SDRV2 documentation):

The first three elements of WORK and the first five elements of IWORK will contain the following statistical data:

AVGH	The average step size used
HUSED	The last step size used (successfully)
AVGORD	The average order used
IMXERR	The index of the element of the solution vector that contributed most to the last error test
NQUSED	The order last used (successfully)
NSTEP	The number of steps taken since the last initialization
NFE	The number of evaluations of the right-hand side
NJE	The number of evaluations of the Jacobian matrix

Note that NFE is the number of derivative evaluations (calls to DERV). What conclusions can you come to regarding the relative efficiency of the implicit (BDF) and explicit methods (recall again that we estimated in Section (4.3) this problem would require about 5,000,000 steps when using the explicit Euler method, and that the maximum step size would be about 2/1,000,000).

- (4.5) Problem 4.3 using LSODE. Print the computational statistics using the code of Program 4.2b.

- (4.6) Problem 4.3 using LSODES. Print the computational statistics (refer to the user's manual in LSODES to see which elements of the work arrays contain the statistics).
- (4.7) Problem 4.3 using LSODI. Print the computational statistics (refer to the user's manual in LSODI to see which elements of the work arrays contain the statistics).
- (4.8) Problem 4.3 using DASSL. Print the computational statistics (refer to the user's manual in DASSL to see which elements of the work arrays contain the statistics).

5

Stability of Numerical Method of Lines Approximations

In Chapter 3, we considered the first principal step in developing a NUMOL solution to PDEs, the replacement of the spatial (boundary value) partial derivatives in the PDEs with algebraic approximations, in this case finite differences. This leads to a set of initial values ODEs that then can be integrated by the methods discussed in Chapter 4. Also, in discussing the numerical integration of ODEs in Chapter 4, we considered the stability properties of various integration algorithms, e.g., explicit vs. implicit methods. We concluded that the performance of ODE integration algorithms is dependent on the product $\lambda \Delta t$, where λ is a representative eigenvalue of the ODE system and Δt is the integration step size. For example, we concluded for the explicit Euler method, the numerical solution will be stable if and only if the condition

$$|\lambda \Delta t| \leq 2 \tag{5.1}$$

is satisfied for all of the ODE eigenvalues [see equations (4.36) and (4.37)]. We might therefore logically inquire about the nature of the ODE eigenvalues for various NUMOL approximations; in this chapter, we briefly analyze the eigenvalue properties, and thus, stability properties, of some basic NUMOL approximations.

5.1 Approximations of the Advection Equation

The NUMOL approximation of the advection equation (3.73) (renumbered here)

$$T_t + vT_x = 0 \quad (5.2)$$

first requires that we replace the spatial derivative, T_x with an algebraic approximation. If we choose a centered difference approximation

$$T_x(x, t) = \frac{T(x + \Delta x, t) - T(x - \Delta x, t)}{2\Delta x} + O(\Delta x^2) \quad (5.3)$$

then substitution of equation (5.3) in equation (5.2) gives the ODEs that must be integrated numerically at the spatial grid points, $i = 2, \dots, N - 1$:

$$dT_i/dt = -v(T_{i+1} - T_{i-1})/(2\Delta x) \quad (5.4)$$

We can now inquire about the eigenvalues of the N ODEs of equation (5.4), much like we did in the case of equations (4.1) and (4.2). Thus, we assume a trial solution and substitute it into equation (5.4). However, the trial solution must take into account the variation of $T(x, t)$ with both x and t (or i and t). Therefore, we assume a *product solution* [also called a *separated solution*] because x and t appear in separate functions, $\phi(x)$ and $\psi(t)$, respectively]:

$$T(x, t) = C\psi(t)\phi(x) \quad (5.5)$$

Further, in accordance with a method proposed by von Neumann [as described, for example, by Strang (5.1)], we assume the x dependency, $\phi(x)$, to be of the form

$$\phi(x) = e^{j k x} \quad (j = \sqrt{-1}) \quad (5.6)$$

where k is a *Fourier number* [k can be considered the summation index in a Fourier series representation of the x dependency of the solution for $T(x, t)$]. Substitution of equations (5.5) and (5.6) in equation (5.4) gives

$$C(d\psi/dt)e^{jkx_i} = -vC\psi(e^{jk(x_i+\Delta x)} - e^{jk(x_i-\Delta x)})/(2\Delta x)$$

or

$$\begin{aligned} d\psi/dt &= -v\psi(e^{jk\Delta x} - e^{-jk\Delta x})/(2\Delta x) \\ &= -v(j/\Delta x) \left(\frac{(e^{jk\Delta x} - e^{-jk\Delta x})}{2j} \right) \psi \\ &= -v(j/\Delta x) \sin(k \Delta x) \psi \end{aligned} \quad (5.7)$$

Equation (5.7) written in terms of an eigenvalue λ becomes

$$d\psi/dt = \lambda\psi \quad (5.8)$$

where

$$\lambda = -v(j/\Delta x) \sin(k \Delta x) \quad (5.9)$$

Note that λ is pure imaginary.

If we now consider integrating equation (5.8) numerically by the explicit Euler method, we can investigate the stability of this numerical integration by applying the explicit Euler method stability criterion, equation (5.1), to equation (5.8). Equation (5.1) can be written in the alternative form

$$|1 + \lambda \Delta t| \leq 1 \quad (5.10)$$

or for the RHS of equation (5.9)

$$|1 - v(j/\Delta x) \sin(k \Delta x) \Delta t| = |1 - j(v \Delta t/\Delta x) \sin(k \Delta x)| \quad (5.11)$$

Then, the square of the absolute value in equation (5.11) is

$$|1 + \lambda \Delta t|^2 = (1 + (v \Delta t/\Delta x)^2 \sin^2(k \Delta x))$$

From equation (5.10), we also require

$$|1 + \lambda \Delta t|^2 = (1 + (v \Delta t/\Delta x)^2 \sin^2(k \Delta x)) \leq 1 \quad (5.12)$$

for stability of the explicit Euler integration of equation (5.8).

We now come to the crucial question, whether we can choose the integration step size Δt so that equation (5.12) is satisfied. The answer is “no.” (Why?) Thus, we come to the conclusion that the NUMOL approximation of equation (5.4) is *unconditionally unstable*; i.e., we cannot choose a value of Δt for which the explicit Euler integration of equation (5.4) is stable.

You might naturally question whether we didn’t earlier find a contraction (or counterexample) to this conclusion. In fact, equation (5.3) is used in subroutine DSS002, and we did the t integration of several ODE problems produced from DSS002 by explicit integration without having the integration become unstable (see, for example, Program 3.5). There are two differences in the previous examples, however, which could account for the fact that we were able to produce a stable numerical integration of the ODEs:

- (1) The ODE integration was done by a fourth order Runge–Kutta method (in subroutine RKF45) rather than by the explicit Euler method. However, this really does not explain the apparent stability

of the ODE integration, since the algorithms in RKF45 do not have significantly better stability properties than the explicit Euler method [recall from equation (4.40) that the stability interval for the fourth order Runge–Kutta method is 2.785 rather than 2 for the explicit Euler method].

- (2) The ODE integration was done by a variable-step method (Δt was changed automatically by the integration algorithm to achieve a prescribed accuracy, which probably also produced a stable solution, at least to the final value of t where the calculations were terminated); however, the preceding analysis is for the case of a constant value of Δt .

In any case, the solution produced by equation (5.3) in DSS002 was highly oscillatory (recall Figure 3.2) [which is consistent with the pure imaginary eigenvalues of equation (5.9)] and therefore not very accurate. We could therefore come to the conclusion, based on the numerical evidence in Chapter 3 (e.g., Figures 3.2 and 3.3), and the preceding stability analysis, that *centered approximations should not be used to approximate spatial derivatives in first-order hyperbolic (convective) PDEs*.

Additionally, since the eigenvalues resulting from centered approximations of convective PDEs are strongly imaginary [again, as suggested by equation (5.9)], the BDF methods described in Chapter 4 will not be very effective in the ODE integration (recall that for BDF methods of order 3 and above, the stability is poor near the imaginary axis, as discussed in Section 4.5 and illustrated in Figures 4.2a and 4.2b). Thus, even if implicit ODE integration is used, e.g., BDF methods of order 3 and higher, the t integration may proceed very slowly or become unstable if a method for approximating the spatial derivatives is used that produces eigenvalues near the imaginary axis. We have, in fact, observed this poor performance of BDF integration.

You could therefore logically ask how we should approximate convective PDEs. An answer has already been suggested in Chapter 3: *Use an upwind approximation* (refer to Section 3.5, and in particular, subroutine DSS012 in Program 3.6). For example, if we use equation (3.79) (renumbered here)

$$T_x(x, t) = \frac{T(x, t) - T(x - \Delta x, t)}{\Delta x} + O(\Delta x) \quad (5.13)$$

we clearly lose accuracy [$O(\Delta x)$ for equation (5.13) and $O(\Delta x^2)$ for equation (5.3)], but we hope to gain improved stability.

If we again follow the preceding analysis, but use equation (5.13) in equation (5.2), we obtain

$$dT_i/dt = -v(T_i - T_{i-1})/\Delta x \quad (5.14)$$

Now, the stability of an explicit Euler integration of equation (5.14) can be investigated. Substitution of the same trial solution, equation (5.5), in equation (5.14) gives

$$C(d\psi/dt)e^{jkx_i} = -vC\psi(e^{jkx_i} - e^{jk(x_i - \Delta x)})/\Delta x$$

or

$$d\psi/dt = -(v/\Delta x)(1 - e^{-jk\Delta x})\psi \quad (5.15)$$

If equation (5.15) is expressed in terms of an eigenvalue, λ ,

$$d\psi/dt = \lambda\psi \quad (5.8)$$

$$\lambda = -(v/\Delta x)(1 - e^{-jk\Delta x}) \quad (5.16)$$

We can again look for a conclusion regarding the integration step size Δt by applying the stability criterion for the explicit Euler method, equation (5.10), to the eigenvalue of equation (5.16):

$$|1 - (v/\Delta x)(1 - e^{-jk\Delta x})\Delta t| = |1 - \alpha(1 - \cos(k \Delta x) + j \sin(k \Delta x))| \leq 1 \quad (5.17)$$

where $\alpha = v \Delta t / \Delta t$. In terms of the square of the absolute value in equation (5.17), we obtain

$$\begin{aligned} |1 + \lambda \Delta t|^2 &= (1 - \alpha(1 - \cos(k \Delta x) \\ &\quad + j \sin(k \Delta x)))^2 \\ &= 1 - 2\alpha(1 - \alpha)(1 - \cos(k \Delta x)) \end{aligned}$$

Therefore, for a stable numerical integration,

$$1 - 2\alpha(1 - \alpha)(1 - \cos(k \Delta x)) \leq 1 \quad (5.18)$$

which requires

$$\alpha \leq 1 \quad (5.19)$$

or

$$|v| \Delta t / \Delta x \leq 1 \quad (5.20)$$

(the absolute value of v is required for flow in the positive or negative directions; we also take $\Delta t > 0$ and $\Delta x > 0$ so that $\alpha > 0$).

Equation (5.20), known as the *Courant–Friedrichs–Lowy (CFL) condition*, places an upper limit on the time integration step Δt , above which the explicit Euler integration of equation (5.14) will be unstable. The CFL condition has an important implication for the NUMOL. As we attempt to improve the accuracy of an NUMOL solution by using a finer

spatial grid (and thereby improve the accuracy of the computed spatial derivatives in the PDEs), Δx is reduced; then, according to equation (5.20), Δt must be decreased proportionately to maintain stability in the ODE numerical integration (if the explicit Euler or some other explicit method is used). In other words, *using a finer spatial grid makes the ODEs stiffer*; we have observed this phenomenon in Section 1.6 of Chapter 1 in the form of longer computer times (more calls to subroutine DERV) when the number of spatial grid points was increased from 11 to 51. This conclusion also follows from equations (5.8) and (5.16); equation (5.16) represents a *spectrum of eigenvalues, λ_k , $k = 1, 2, \dots$, and this spectrum is broader with decreasing Δx* (the ODEs are stiffer with decreasing Δx).

Of course, we have already observed that equation (5.14) is a rather inaccurate approximation of equation (5.2) (see Figure 3.4). If we attempt to improve the accuracy by using a finer spatial grid, and at the same time attempt to use a larger integration step than allowed by equation (5.20) to reduce the computation, we must use implicit integration of the ODEs [such as equation (5.14)]. Again, though the eigenvalues given by equation (5.16) are complex, and we must be careful not to go out of the stability region of whatever implicit integrator we use (e.g., BDF methods).

Finally, we should keep in mind that the preceding stability analysis requires the PDE to be linear. In using the methods of stability analysis presented in this chapter, we can only hope that what we conclude for linear PDEs will also apply, at least qualitatively, to nonlinear PDEs as well.

5.2 Approximation of the Heat-Conduction Equation

We obviously could attempt to analyze the stability of many NUMOL approximations applied to many different PDEs (consider, for example, the five-point approximations in subroutine DSS004 applied to the several PDEs discussed in previous chapters). However, in order to keep the discussion in this chapter to reasonable length, we will develop just one more stability analysis for a linear PDE, and then conclude the chapter with an example of the NUMOL applied to a system of nonlinear PDEs, to which we will apply a linearized stability analysis.

The linear PDE will again be Fourier's second law (or the heat-conduction equation), equation (1.4) [equation (1.10) in subscript notation, which is renumbered here]:

$$T_t = \alpha T_{xx} \quad (5.21)$$

If we apply a three-point centered approximation,

$$T_{xx}(x, t) = \frac{T(x + \Delta x, t) - 2T(x, t) + T(x - \Delta x, t)}{\Delta x^2} + O(\Delta x^2) \quad (5.22)$$

to equation (5.21), the resulting NUMOL ODEs are

$$dT_i/dt = \alpha(T_{i+1} - 2T_i + T_{i-1})/\Delta x^2 \quad (5.23)$$

Again, substitution of the trial solution, equation (5.5), in equation (5.23) gives

$$C(d\psi/dt)e^{jkx_i} = \alpha C\psi(e^{jk(x_i+\Delta x)} - 2e^{jkx_i} + e^{jk(x_i-\Delta x)})/\Delta x^2$$

or the NUMOL ODEs

$$d\psi/dt = \alpha\psi(e^{jk\Delta x} - 2 + e^{-jk\Delta x})/\Delta x^2$$

Some straightforward rearrangement gives

$$\begin{aligned} d\psi/dt &= \alpha\psi(e^{jk(\Delta x/2)} - e^{jk(-\Delta x/2)})^2/\Delta x^2 \\ &= -(4\alpha/\Delta x^2) \frac{(e^{jk\Delta x/2} - e^{-jk\Delta x/2})^2}{(2j)^2}\psi \\ &= -\alpha(2/\Delta x)^2 \sin^2(k \Delta x/2)\psi \end{aligned} \quad (5.24)$$

Equation (5.24) can be written as an ODE with an eigenvalue

$$d\psi/dt = \lambda\psi \quad (5.25)$$

$$\lambda = -\alpha(2/\Delta x)^2 \sin^2(k \Delta x/2) \quad (5.26)$$

If we now apply the stability criterion of the explicit Euler method, equations (4.37) or (5.1) (renumbered here)

$$|\lambda \Delta t| \leq 2 \quad (5.27)$$

we immediately come to the conclusion

$$\alpha \Delta t/\Delta x^2 \leq 1/2 \quad (5.28)$$

which again puts an upper limit on the integration step size, Δt , above which the explicit Euler integration of equation (5.23) will be unstable. Note again that equation (5.28) indicates what will happen to the stability of the explicit ODE integration when we use a finer grid (smaller Δx); now, however, the eigenvalues [from equation (5.26)] are proportional (or separate) as $1/\Delta x^2$, so that the stiffness for equation (5.21) is in a sense more severe than for equation (5.2). Recall again that we observed increasing stiffness with increasing numbers of spatial grid points N for equation (5.21) in Section 1.6; this is not surprising since the spread in the eigen-

values is proportional to N^2 [i.e., proportional to $1/\Delta x^2$ from equation (5.28)].

This completes all of the examples of von Neumann stability analysis of linear PDEs that we shall consider. Basically, we have observed that stiffness of the NUMOL ODEs increases with increasing numbers of spatial grid points, and that we may need to use an implicit ODE integrator because of this stiffness. Also, we must be careful in approximating the problem PDEs as a system of ODEs to avoid violation of the stability constraints of explicit and *implicit* methods (for the latter, recall the example of the higher-order BDF methods, which have poor stability properties along the imaginary axis).

All of the preceding discussion was for linear ODEs. To conclude this chapter, we shall apply these ideas to a nonlinear PDE problem, which also illustrates the versatility and flexibility of the NUMOL in the solution of realistic PDE models.

5.3 A System of Nonlinear PDEs with _____ Mapping and Eigenvalue Analysis of _____ the ODE Jacobian Matrix _____

The physical system to be analyzed is illustrated in Figure 5.1. The system shown in Figure 5.1 was selected because

- (1) It is a physical system that is easily understood by readers with any background; i.e., its two basic components are air and water, and the physical phenomenon described by the PDE model is well known (humidification).
- (2) The mathematical model for the system is a system of three nonlinear, one-dimensional, initial-value PDEs, which are well suited for NUMOL solution.
- (3) The resulting system of NUMOL ODEs is modest in size, yet has interesting properties as reflected in the map and eigenvalues of its Jacobian matrix.

However, before we get into these mathematical details, we first discuss the derivation of the model PDEs. As we have observed in Section 1.2, the usual procedure for deriving a model of a system with a significant spatial dimension, a *distributed system*, is to write one or more equations for an incremental section of the system, and then take the limit as the dimension(s) of the section approaches zero. The equations are typically based on fundamental principles of physics and chemistry, e.g., conserva-

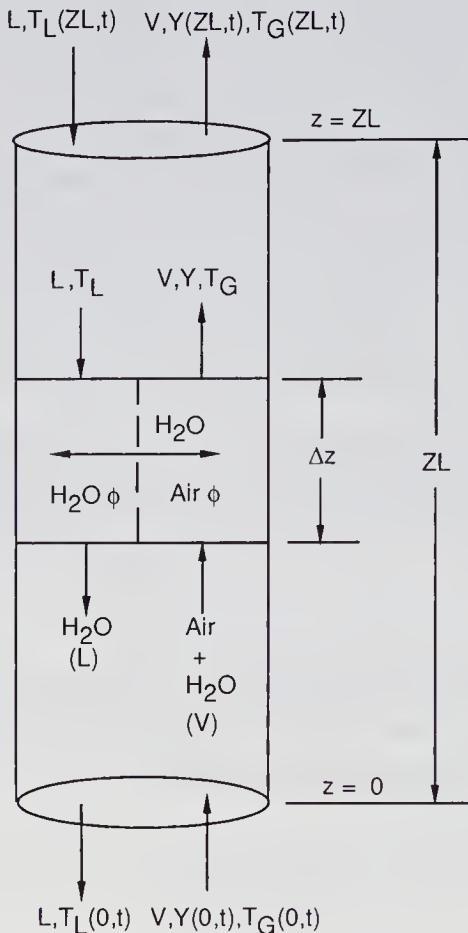


Figure 5.1 Humidification column.

tion of mass, energy, momentum with associated thermodynamics, and chemical kinetics.

In the present case, the humidification column of Figure 5.1 is packed with a porous medium that permits the flow of water down through the column by gravity and the flow of air up as a result of an imposed pressure at the bottom of the column (e.g., from a centrifugal blower). The packing in the column promotes contact between the water and air, and therefore enhances the exchange of mass and energy between the two streams (we will neglect the exchange of momentum, although this is manifest as a drop of the air pressure along the column).

To fully analyze the operation of the column, we require the calculation of the following dependent variables:

- (1) The air humidity $y(z, t)$ as a function of position along the column z and time t . We define the humidity as the ratio of the moles of water to the moles of dry air.

- (2) The air temperature $TG(z, t)$.
- (3) The water temperature, $TL(z, t)$.

The humidity y is computed from a mass balance for the air written on a section of the column of length Δz , followed by $\Delta z \rightarrow 0$. The final result is

$$y_t = -(V/(GS))y_z + (k_y a_v/G)(y_s - y) \quad (5.29)$$

Again, a subscript with respect to an independent variable (z or t) denotes a partial derivative with respect to that variable. Thus the combination of the t derivative on the LHS of equation (5.29) and the z derivative in the first RHS term is again the advection group discussed in Section 3.5, representing accumulation of water in the air and convection of water in the air within the differential volume of the column, respectively. Note that the air velocity $V/(GS)$ is positive since the air flows up from the bottom of the column (in the positive z direction).

The second RHS of equation (5.29) represents the mass transfer of water to or from the air due to the humidity difference between the air and water ($y_s - y$). When $(y_s - y)$ is negative (the air humidity exceeds the saturation humidity), condensation of water from the air takes place; if $(y_s - y)$ is positive, evaporation of water into the air (humidification) take places.

Equation (5.29) is derived in detail in the subsequent coding (the comments at the beginning of subroutine INITIAL), so the details will not be given here. Also, all of the variables are defined in these comments. We offer the documentation in INITIAL as an example of what should be provided for prospective users of a NUMOL code so that they may achieve a full understanding of the PDE problem and how it is coded (i.e., the author of a NUMOL code can probably never provide too many comments).

The temperature of the air is computed by writing an energy balance for the air in the differential section of the column (again, the details are given in subroutine INITIAL). The final result is

$$\begin{aligned} EV_t &= -(V/(GS))EP_z + (ha_v/G)(TL - TG) \\ &\quad + (k_y a_v/G)(y_s - y)(C_{vv}TG + \Delta H_{\text{vap}}) \end{aligned} \quad (5.30)$$

The LHS and first RHS terms of equation (5.30) are again the advection group for the air [the air velocity $V/(GS)$ is positive since the air enters the bottom of the column and flows in the positive z direction]. However, there is one difference in this advection group we did not observe previously; the dependent variable in the time derivative is the air internal energy EV , while the dependent variable in the spatial derivative is the air enthalpy EP .

The second RHS term of equation (5.30) is the heat-transfer rate between the water and air due to the temperature difference ($TL - TG$). The third RHS term of equation (5.30) represents the gain or loss of energy by the air due to mass transfer of water to the air (evaporation or humidification) or to the water (condensation); this transfer occurs as a result of differences between the air humidity y and the saturation humidity y_s , as in equation (5.29). Note also that when equation (5.30) is integrated in time, the computed dependent variable is the air internal energy, EV . This must then be converted to the corresponding air temperature TG for use in the heat-transfer term of equations (5.30).

Finally, the water temperature $TL(z, t)$ is computed by writing an energy balance on the water in the differential section of the column. The final result is (the details are given in subroutine INITIAL)

$$\begin{aligned} TL_t = & (L/(HS))TL_z - (ha_v/(C_LH))(TL - TG) \\ & - (k_y a_v/(C_LH))(y_s - y)(C_{vv}TG + \Delta H_{vap}) \end{aligned} \quad (5.31)$$

The advection group for the liquid temperature is apparent. (What is the sign of the velocity?) The second RHS term of equation (5.31) is the heat-transfer rate between the water and the air. The third RHS term of equation (5.31) represents the gain or loss of energy by the water due to mass transfer of water from the air to the water (condensation) or to the air from the water (evaporation or humidification); this transfer occurs as a result of differences between the air humidity y and the saturation humidity, y_s , as in equations (5.29) and (5.30). Note that the second and third RHS terms of equation (5.31) are the same as in equation (5.30), but opposite in sign. (Why?)

Equations (5.29), (5.30), and (5.31) are the PDE model for the humidification column; since they are derived in detail in the comments in subroutine INITIAL, their derivation will not be discussed further here. Note that three initial conditions and three boundary conditions are required to complete the model. (Why?) The initial conditions [for the derivatives in t in equations (5.29), (5.30), and (5.31)] are

$$y(z, 0) = 0.01 \text{ gm moles water/gm mole dry air} \quad (5.32)$$

$$\begin{aligned} EV(z, 0) = & C_{va}TG(z, 0) + y(z, 0)(C_{vv}TG(z, 0) + \Delta H_{vap}) \\ = & (5.3)(43.33) + 0.01((5.3)(43.33) \\ & + 9443.6) \text{ cal/gm mole air} \end{aligned} \quad (5.33)$$

$$TL(z, 0) = 43.33^\circ\text{C} \quad (5.34)$$

The boundary conditions [for the derivatives in z in equations (5.29),

(5.30), and (5.31)] are

$$y(0, t) = 0.01 \text{ gm moles water/gm mole dry air} \quad (5.35)$$

$$\begin{aligned} EP(0, t) &= C_{pa}TG(0, t) + y(0, t)(C_{pv}TG(0, t) + \Delta H_{vap}) \\ &= (5.3)(43.33) + 0.01((5.3)(43.33) \\ &\quad + 9443.6) \text{ cal/gm mole air} \end{aligned} \quad (5.36)$$

$$TL(z_l, t) = 43.33^\circ\text{C} \quad (5.37)$$

These boundary conditions reflect the entering air humidity $y(0, t)$, the entering air enthalpy $EP(0, t)$, and the entering liquid temperature $TL(z_l, t)$ (z_l is the column length).

Subroutine INITAL for equations (5.29) to (5.37) is listed in Program 5.1a. Several features of subroutine INITAL should be noted:

Program 5.1a Subroutine INITAL for Equations (5.29) to (5.42)

SUBROUTINE INITAL

```

C... DYNAMICS OF A HUMIDIFICATION COLUMN
C... THE HUMIDIFICATION COLUMN IS DIAGRAMMED IN FIGURE 5.1. THE
C... MODEL VARIABLES AND PARAMETERS ARE
C...
C...     TG      GAS TEMPERATURE (UNITS OF THE MODEL VARIABLES ARE
C...             DEFINED BELOW WITH THE NUMERICAL VALUES OF THE MODEL
C...             PARAMETERS AND AUXILIARY CONDITIONS)
C...
C...     EV      INTERNAL ENERGY OF THE GAS STREAM
C...             = (CVA*TG + Y*(CVV*TG + DHVAP))
C...
C...     EP      ENTHALPY OF THE GAS STREAM
C...             = (CPA*TG + Y*(CPV*TG + DHVAP))
C...
C...     TL      LIQUID TEMPERATURE
C...
C...     Y       MOLE RATIO OF H2O IN THE GAS
C...
C...     YS      MOLE RATIO OF H2O IN THE GAS WHICH WOULD BE IN
C...             EQUILIBRIUM WITH THE LIQUID AT TEMPERATURE TL
C...
C...     T       TIME
C...
C...     Z       AXIAL POSITION ALONG THE COLUMN
C...
C...     ZL      LENGTH OF THE COLUMN
C...
C...     V       DRY AIR MOLAR FLOW RATE
C...
C...     G       DRY AIR MOLAR HOLDUP
C...
C...     S       COLUMN CROSS SECTIONAL AREA

```

continues

C... KY MASS TRANSFER COEFFICIENT
C... AV HEAT AND MASS TRANSFER AREAS PER UNIT VOLUME OF
C... COLUMN
C... H HEAT TRANSFER COEFFICIENT
C... CVV SPECIFIC HEAT OF WATER VAPOR AT CONSTANT VOLUME
C... CPV SPECIFIC HEAT OF WATER VAPOR AT CONSTANT PRESSURE
C... CVA SPECIFIC HEAT OF DRY AIR AT CONSTANT VOLUME
C... CPA SPECIFIC HEAT OF DRY AIR AT CONSTANT PRESSURE
C... CL SPECIFIC HEAT OF WATER
C... L LIQUID (H₂O) MOLAR FLOW RATE
C... H LIQUID (H₂O) MOLAR HOLDUP
C... DHVAP HEAT OF VAPORIZATION OF WATER
C...
C... THE DEPENDENT VARIABLES OF PARTICULAR INTEREST ARE Y(Z,T), TG(Z,T)
C... TL(Z,T). IN ORDER TO COMPUTE THESE VARIABLES, BASIC CONSERVATION
C... PRINCIPLES ARE APPLIED TO THE AIR AND WATER STREAMS WITHIN THE
C... COLUMN OVER THE INCREMENTAL SECTION OF LENGTH DZ (SEE THE DIAGRAM)
C... AND THEN THE LIMIT OF DZ → 0 GIVES THE MODEL DIFFERENTIAL EQUA-
C... TIONS. SINCE THE MODEL IS DYNAMIC, TIME, T, AND AXIAL POSITION,
C... Z, APPEAR IN THE EQUATIONS AS INDEPENDENT VARIABLES AND THEREFORE
C... PARTIAL DIFFERENTIAL EQUATIONS (PDES) MUST BE USED.
C...
C... THE PDES ARE DERIVED AS FOLLOWS. A H₂O BALANCE ON THE GAS PHASE
C... GIVES
C...
$$\frac{(Y \cdot G \cdot S \cdot DZ)}{T} = \frac{V \cdot Y}{Z} - \frac{V \cdot Y}{Z + DZ} + KY \cdot AV \cdot (YS - Y) \cdot S \cdot DZ \quad (1A)$$

C... WHERE THE SUBSCRIPT T ON THE LHS DENOTES A PARTIAL DERIVATIVE WITH
C... RESPECT TO TIME, T.
C...
C... THE PHYSICAL INTERPRETATION AND UNITS OF EACH OF THE TERMS IN
C... EQUATION (1A) ARE INDICATED BELOW
C...
C...
$$\frac{(Y \cdot G \cdot S \cdot DZ)}{T}$$

C...
C... = (MOLS H₂O/MOL DRY AIR) * (MOLS DRY AIR/CM^{**3}) * (CM^{**2}) *
C... CM * (1/HR) = MOLS H₂O/HR ACCUMULATING IN THE GAS PHASE
C... WITHIN THE INCREMENTAL SECTION OF LENGTH DZ (SEE THE
C... DIAGRAM BELOW)
C...
C...
$$\frac{V \cdot Y}{Z}$$

C...
C... = (MOLS DRY AIR/HR) * (MOLS H₂O/MOL DRY AIR) = MOLS H₂O/HR
C... FLOWING INTO THE INCREMENTAL SECTION AT Z
C...
C...
$$\frac{V \cdot Y}{Z + DZ}$$

C...
C... = (MOLS DRY AIR/HR) * (MOLS H₂O/MOL DRY AIR) = MOLS H₂O/HR
C... FLOWING OUT OF THE INCREMENTAL SECTION AT Z+DZ

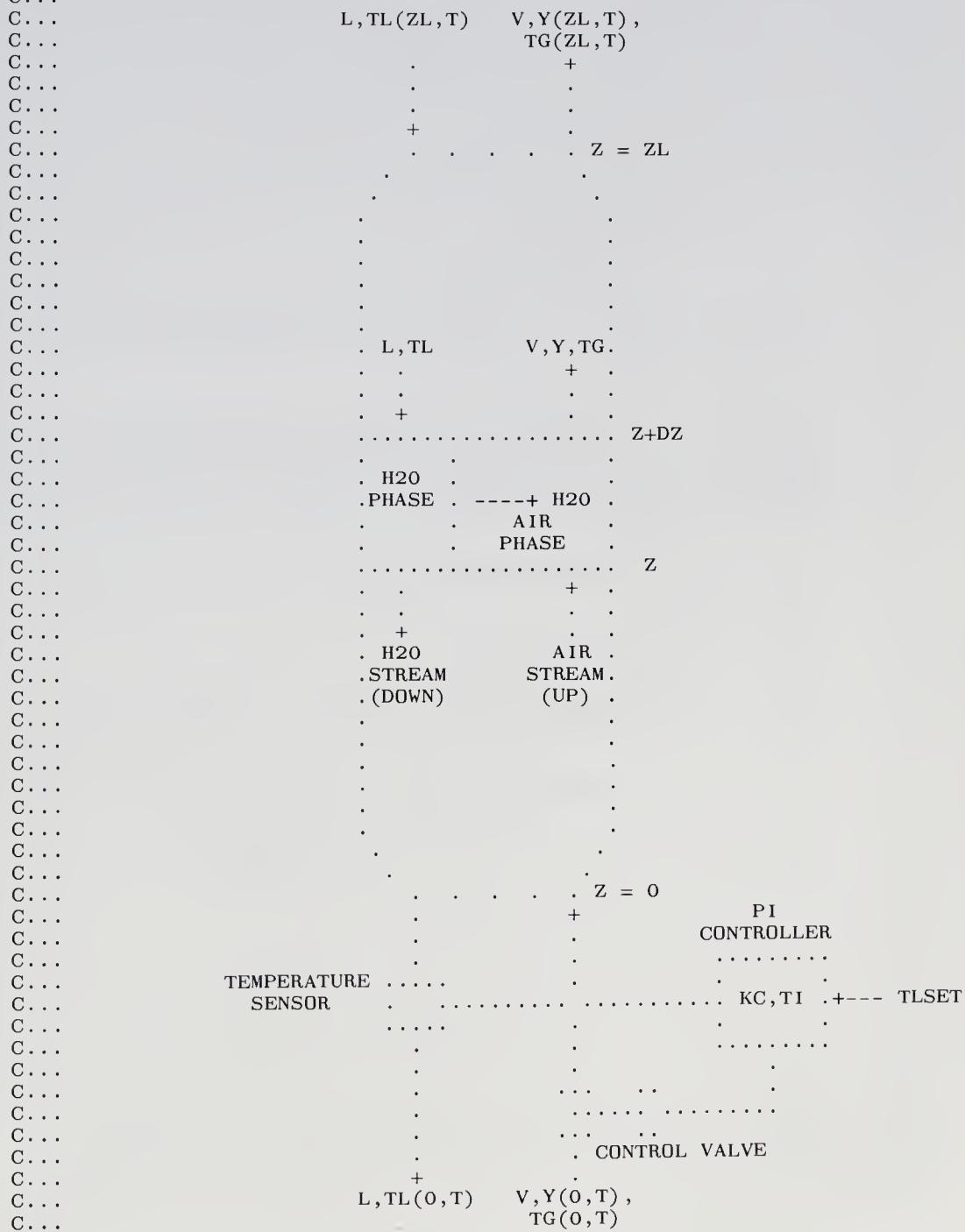
continues

C...
C... KY*AV*(YS - Y)*S*DZ
C...
C... = (MOLS DRY AIR/HR-CM**2)*(CM**2/CM**3)*(MOLS H2O/MOL DRY
C... AIR)*(CM**2)*(CM) = MOLS H2O/HR TRANSFERRED TO OR FROM
C... THE AIR STREAM IN THE INCREMENTAL SECTION (THE DIRECTION
C... OF TRANSFER DEPENDS ON THE SIGN OF (YS - Y))
C...
C... IN THE LIMIT AS DZ ---+ 0, EQUATION (1A) BECOMES
C...
C... $\frac{Y_T}{T} = -\left(\frac{V}{G*S}\right)*Y_Z + \left(\frac{KY*AV}{G}\right)*(YS - Y)$ (1)
C...
C... AN ENERGY BALANCE ON THE GAS PHASE GIVES
C...
C... $\left(\frac{(CVA*TG + Y*(CVV*TG + DHVAP))*G*S*DZ}{T}\right) =$
C...
C... $\left(\frac{(CPA*TG + Y*(CPV*TG + DHVAP))*V}{Z}\right) -$
C...
C... $\left(\frac{(CPA*TG + Y*(CPV*TG + DHVAP))*V}{Z+DZ}\right) -$
C...
C... $H*AV*(TL - TG)*S*DZ + KY*AV*(YS - Y)*S*DZ*(CVV*TG + DHVAP)$
C...
C... THE PHYSICAL INTERPRETATION AND UNITS OF EACH OF THE TERMS IN
C... EQUATION (2A) ARE INDICATED BELOW
C...
C... $\left(\frac{(CVA*TG + Y*(CVV*TG + DHVAP))*G*S*DZ}{T}\right)$
C...
C... = (CAL/MOL DRY AIR-C)*(C) + (MOLS H2O/MOL DRY AIR)*
C... (CAL/MOL H2O-C)*(C) + (CAL/MOL H2O)*(MOLS DRY AIR/
C... CM**3)*(CM**2)*CM*(1/HR) = CAL/HR ACCUMULATING IN THE
C... GAS PHASE WITHIN THE INCREMENTAL SECTION
C...
C... $\left(\frac{(CPA*TG + Y*(CPV*TG + DHVAP))*V}{Z}\right)$
C...
C... = (CAL/MOL DRY AIR-C)*(C) + (MOLS H2O/MOL DRY AIR)*
C... (CAL/MOL H2O-C)*(C) + (CAL/MOL H2O)*(MOLS DRY AIR/HR)
C... = CAL/HR FLOWING INTO THE INCREMENTAL SECTION AT Z
C...
C... $\left(\frac{(CPA*TG + Y*(CPV*TG + DHVAP))*V}{Z+DZ}\right)$
C...
C... = (CAL/MOL DRY AIR-C)*(C) + (MOLS H2O/MOL DRY AIR)*
C... (CAL/MOL H2O-C)*(C) + (CAL/MOL H2O)*(MOLS DRY AIR/HR)
C... = CAL/HR FLOWING OUT OF THE INCREMENTAL SECTION AT Z+DZ
C...
C... $H*AV*(TL - TG)*S*DZ$
C...
C... = (CAL/HR-CM**2-C)*(CM**2/CM**3)*(C)*(CM**2)*CM = CAL/HR
C... TRANSFERRED BETWEEN THE H2O AND GAS PHASES DUE TO TEMP-
C... ERATURE DIFFERENCES (THE DIRECTION OF TRANSFER DEPENDS
C... ON THE SIGN OF (TL - TG))
C...
C... KY*AV*(YS - Y)*S*DZ*(CVV*TG + DHVAP)
C...
C... = (MOLS DRY AIR/HR-CM**2)*(CM**2/CM**3)*(MOLS H2O/MOL DRY
C... AIR)*(CM**2)*(CM)*((CAL/MOL H2O-C)*C + (CAL/MOL H2O)) =
C... CAL/HR TRANSFERRED BETWEEN THE H2O AND GAS PHASES DUE TO
C... DIFFERENCES IN HUMIDITY

continues

C... IN THE LIMIT AS DZ ---+ 0, EQUATION (2A) BECOMES
C... C... EV = -(V/(G*S))*EP + (H*AV/G)*(TL - TG)
C... T Z (2)
C... C... + (KY*AV/G)*(YS - Y)*(CVV*TG + DHVAP)
C... C... NOTE THE ENERGY GROUPS, EV (ENERGY AT CONSTANT VOLUME) AND EP
C... (ENERGY AT CONSTANT PRESSURE) HAVE BEEN USED TO SIMPLIFY THE
C... TERMS IN EQUATION (2). EV AND EP ARE DEFINED IN THE TABLE OF
C... NOMENCLATURE GIVEN EARLIER.
C... C... AN ENERGY BALANCE ON THE LIQUID GIVES
C... C... (CL*TL*H*S*DZ) = (CL*TL*L) - (CL*TL*L)
C... T Z+DZ Z (3A)
C... C... -H*AV*(TL - TG)*S*DZ - KY(AV*(YS - Y)*S*DZ*(CVV*TG + HDVAP)
C... C... THE PHYSICAL INTERPRETATION AND UNITS OF EACH OF THE TERMS IN
C... EQUATION (3A) ARE GIVEN BELOW
C... C... (CL*TL*H*S*DZ)
C... T
C... C... (CAL/MOL H2O-C)*(C)*(MOLS H2O/CM**3)*(CM**2)*CM*(1/HR)
C... = CAL/HR ACCUMULATING IN THE H2O PHASE WITHIN THE
C... INCREMENTAL SECTION OF LENGTH DZ
C... C... (CL*TL*L)
C... Z+DZ
C... C... = (CAL/MOL H2O-C)*(C)*(MOLS H2O/HR) = CAL/HR FLOWING INTO
C... THE INCREMENTAL SECTION AT Z+DZ
C... C... (CL*TL*L)
C... Z
C... C... = (CAL/MOL H2O-C)*(C)*(MOLS H2O/HR) = CAL/HR FLOWING OUT OF
C... THE INCREMENTAL SECTION AT Z
C... C... H*AV*(TL - TG)*S*DZ
C... C... = (CAL/HR-CM**2-C)*(CM**2/CM**3)*(C)*(CM**2)*CM = CAL/HR
C... TRANSFERRED BETWEEN THE H2O AND GAS PHASES DUE TO TEMP-
C... ERATURE DIFFERENCES (THE DIRECTION OF TRANSFER DEPENDS
C... ON THE SIGN OF (TL - TG))
C... C... KY*AV*(YS - Y)*S*DZ*(CVV*TG + DHVAP)
C... C... = (MOLS DRY AIR/HR-CM**2)*(CM**2/CM**3)*(MOLS H2O/MOL DRY
C... AIR)*(CM**2)*(CM)*((CAL/MOL H2O-C)*C + CAL/MOL H2O) =
C... CAL/HR TRANSFERRED BETWEEN THE H2O AND GAS PHASES DUE TO
C... DIFFERENCES IN HUMIDITY
C... C... IN THE LIMIT AS DZ ---+ 0, EQUATION (3A) BECOMES
C... C... TL = (L/(H*S))*TL - (H*AV/(CL*H))*(TL - TG)
C... T Z (3)
C... C... - (KY*AV/(CL*H))*(YS - Y)*(CVV*TG + DHVAP)
C... C... AN EXTENSION OF THE PRECEDING SYSTEM IS ALSO SIMULATED IN THE
continues

C... FOLLOWING CODING. SPECIFICALLY, THE EXITING WATER TEMPERATURE
C... $TL(0,T)$, IS SENSED, COMPARED WITH A SET POINT VALUE, $TLSET$,
C... AND THE DIFFERENCE (ERROR), $E = TL(0,T) - TLSET$, IS USED TO
C... ADJUST THE ENTERING AIR FLOW RATE, V , TO MOVE $TL(0,T)$ TO $TLSET$.
C... THE ADJUSTMENT OF V IS ACHIEVED WITH A PROPORTIONAL-INTEGRAL (PI)
C... CONTROLLER. THE FOLLOWING DIAGRAM ILLUSTRATES THE CONTROLLER
C... AT THE BOTTOM OF THE HUMIDIFICATION COLUMN



continues

C...
C... THE PURPOSE OF THE SIMULATION IS ESSENTIALLY TO DETERMINE THE
C... VALUES OF THE CONTROLLER GAIN, KC, AND INTEGRAL TIME, TI, SO
C... THAT THE EXITING LIQUID TEMPERATURE, TL(0,T), IS CONTROLLED CLOSE
C... TO THE SET POINT, TLSET.
C...
C... THE CONTROLLER EQUATIONS ARE
C...
C... V = CV*SQRT(DP)*X
C...
C... X = XSS + KC*(E + (1/TI) INT(E)DT)
C...
C... E = TL(0,T) - TLSET
C...
C... X CONTROL VALVE STEM POSITION
C...
C... XSS X AT STEADY STATE
C...
C... KC CONTROLLER GAIN
C...
C... TI CONTROLLER INTEGRAL TIME
C...
C... CVDP PRODUCT OF THE CONTROL VALVE CONSTANT AND SQUARE
C... ROOT OF THE PRESSURE DROP ACROSS THE VALVE
C...
C... TLSET CONTROLLER SET POINT
C...
C... THE MODEL PARAMETERS AND AUXILIARY CONDITIONS IN ENGLISH UNITS ARE
C... LISTED BELOW, FOLLOWED BY EQUIVALENT VALUES IN METRIC UNITS. THE
C... PROGRAM CAN BE EXECUTED WITH EITHER SET BY SELECTING THE VALUE OF
C... THE VARIABLE METRIC IN THE PROGRAM (METRIC = 0 FOR ENGLISH UNITS,
C... METRIC = 1 FOR METRIC UNITS)
C...
C... *****
C... ENGLISH UNITS
C...
C... H*AV = 50 BTU/(HR-FT**3-F)
C...
C... KY*AV = 4.1 LB MOLS DRY AIR/(HR-FT**3)
C...
C... ZL = 8 FT, CL = 18 BTU/(LB MOL-F), DHVAP = 17000 BTU/(LB MOL)
C...
C... CPV = 7.3 BTU/(LB MOL-F), CPA = 7.3 BTU/(LB MOL-F)
C...
C... CVV = CPV - R = 7.3 - 1.987 = 5.3 BTU/(LB MOL-F)
C...
C... CVA = CPA - R = 7.3 - 1.987 = 5.3 BTU/(LB MOL-F)
C...
C... L = 50 LB MOLS/HR, S = 1 FT**2
C...
C... H = 0.5 LB MOLS/FT**3, G = 0.01 LB MOLS/FT**3
C...
C... P = 10**((7.96681 - 3002.4/(378.4 + TL)) MM HG (TL IS IN F)
C...
C... YS = P/(760 - P) LB MOLS H2O/LB MOL DRY AIR
C...
C... TG(0,T) = 110 F, TL(ZL,T) = 110 F, Y(0,T) = 0.01 LB MOLS H2O/
C... LB MOL DRY AIR
C... TG(Z,0) = TL(Z,0) = 110 F, Y(Z,0) = 0.01
C...
C... KC = 0.02 (1/F), TI = 0.1 HR, CV*(DP**0.5) = 50 LB MOLS/HR

continues

```

C...
C... XSS = 0.5, TLSET = 90 F
C...
C... ****
C... METRIC UNITS
C...
C... H*AV = 0.8015 CAL/(HR-CM**3-C)
C...
C... KY*AV = 0.06573 GM MOLS DRY AIR/(HR-CM**3)
C...
C... ZL = 243.8 CM, CL = 18 CAL/(GM MOL-C), DHVAP = 9443.6 CAL/GM MOL
C...
C... CPV = 7.3 CAL/(GM MOL-C), CPA = 7.3 CAL/(GM MOL-C)
C...
C... CVV = CPV - R = 7.3 - 1.987 = 5.3 CAL/(GM MOL-C)
C...
C... CVA = CPA - R = 7.3 - 1.987 = 5.3 CAL/(GM MOL-C)
C...
C... L = 22700 GM MOLS/HR, S = 929.03 CM**2
C...
C... H = 0.008015 GM MOLS/CM**3, G = 0.0001603 GM MOLS/CM**3
C...
C... P = 10**((7.96681 - 3002.4/(378.4 + 1.8*TL + 32)) MM HG
C... (TL IS IN C)
C...
C... YS = P/(760 - P) GM MOLS H2O/GM MOL DRY AIR
C...
C... TG(0,T) = 43.33 C, TL(ZL,T) = 43.33 C, Y(0,T) = 0.01 GM MOL H2O/
C... GM MOL DRY AIR)
C...
C... TG(Z,0) = TL(Z,0) = 43.33 C, Y(Z,0) = 0.01
C...
C... KC = 0.036 (1/C), TI = 0.1 HR, CV*(DP**0.5) = 22700 GM MOLS/HR
C...
C... XSS = 0.5, TLSET = 32.22 C
C...
C... ****
C...
C... THE CONVERSION FACTORS USED TO GO FROM ONE SET OF UNITS TO THE
C... OTHER ARE
C...
C... 1 BTU = 252.2 CAL
C...
C... 1 BTU/LB MOL-F = 1 CAL/GM MOL-C
C...
C... 1 LB = 454 GM
C...
C... 1 LB MOL = 454 GM MOL
C...
C... 1 F = 1/1.8 C
C...
C... 1 FT = 30.48 CM
C...
C... 1 FT**3 = 28316.8 CM**3
C...
C... TF = 1.8*TC + 32
C...
C... THE EXITING GAS COMPOSITION AND TEMPERATURE, Y(ZL,T) AND TG(ZL,T),
C... THE EXITING LIQUID TEMPERATURE, TL(0,T), AND THE GAS FLOW RATE,
C... V, ARE OF PARTICULAR INTEREST IN THIS SIMULATION.

```

continues

```

C...
C... COMMON AREA TO ESTABLISH LINKAGE WITH OTHER SUBROUTINES
COMMON/T/      T,  NSTOP,  NORUN
1      /Y/  Y(11),  EV(11),  TL(11),      EI
2      /F/  YT(11),EVT(11),TLT(11),      E
3      /S/  YZ(11),EPZ(11),TLZ(11)
4      /C/      V,          L,          G,          H,          HLA,          KYAV,
5          CL,          CPV,          CPA,          CVV,          CVA,          ZL,
6          DHVAP,          S,          P1,          P2,          P3,          P4,
7          P5,          P6,          P7,  TG(11),  EP(11),  YS(11),
8          KC,          TI,  CVDP,          XSS,  TLSET,          X,
9          IP,          N,  METRIC
REAL          L,          KYAV,          KC
C...
C... SELECT ENGLISH (METRIC = 0) OR METRIC (METRIC = 1) UNITS
IF(NORUN.EQ.1)METRIC=0
IF(NORUN.EQ.2)METRIC=1
C...
C... SET THE MODEL PARAMETERS IN THE UNITS SELECTED
C...
C... ENGLISH UNITS
IF(METRIC.EQ.0)THEN
N=11
L=50.
G=0.01
H=0.5
HLA=50.
KYAV=4.1
CL=18.
CPV=7.3
CPA=7.3
CVV=5.3
CVA=5.3
ZL=8.
DHVAP=17000.
S=1.
C...
C... SET THE CONTROLLER PARAMETERS
KC=0.02
TI=0.1
CVDP=50.
XSS=0.5
TLSET=90.
C...
C... METRIC UNITS
ELSE
N=11
L=22700.
G=0.0001603
H=0.008015
HLA=0.8015
KYAV=0.06573
CL=18.
CPV=7.3
CPA=7.3
CVV=5.3
CVA=5.3
ZL=243.8
DHVAP=9443.6
S=929.03

```

continues

```

C...
C... SET THE CONTROLLER PARAMETERS
KC=0.036
TI=0.1
CVDP=22700.
XSS=0.5
TLSET=32.22
END IF
C...
C... COMPUTE SOME CONSTANTS FOR USE IN OTHER SUBROUTINES
P2=KYAV/G
P3=HLAV/G
P4=L/(H*S)
P5=HLAV/(CL*H)
P6=KYAV/(CL*H)
C...
C... INITIAL CONDITIONS
DO 1 I=1,N
Y(I)=0.01
IF(METRIC.EQ.0)THEN
TL(I)=110.
TG(I)=110.
ELSE
TL(I)=43.33
TG(I)=43.33
END IF
EV(I)=CVA*TG(I)+Y(I)*(CVV*TG(I)+DHVAP)
1 CONTINUE
EI=0.
C...
C... INITIALIZE THE MODEL DERIVATIVES
CALL DERV
IP=0
RETURN
END

```

The COMMON area

```

COMMON/T/      T,    NSTOP,    NORUN
1      /Y/    Y(11),  EV(11),  TL(11),      EI
2      /F/    YT(11), EVT(11), TLT(11),     E
3      /S/    YZ(11), EPZ(11), TLZ(11)
4      /C/      V,      L,      G,      H,      HLA V,    KYAV,
5          CL,    CPV,    CPA,    CVV,    CVA,    ZL,
6          DHVAP,   S,      P1,    P2,      P3,      P4.
7          P5,    P6,      P7,    TG(11),  EP(11),  YS(11),
8          KC,    TI,    CVDP,    XSS,    TLSET,      X,
9          IP,    N,    METRIC

```

consists of four sections. COMMON/T/, as usual, contains the time t , an integer variable for stopping a run, NSTOP, and a run counter, NORUN.

COMMON/Y/ contains the dependent variables of equation (5.29), $y(z, t)$, equation (5.30), $EV(z, t)$, and equation (5.31), $TL(z, t)$, respectively, each defined on an 11-point grid in z . COMMON/Y/ also contains a dependent variable, EI , which is included to model the action of an automatic controller for the column of Figure 5.1; this controller is depicted in the diagram of the column in subroutine INITAL. Essentially, the controller adjusts the air flow rate into the bottom of the volume V to maintain the exiting liquid temperature $TL(0, t)$ at a prescribed value $TL_{\text{set}} = 32.22^\circ\text{C}$ (the *set point*). First, the controller computes an error e , as the difference between the actual exiting liquid temperature and the set point

$$e = TL(0, t) - TL_{\text{set}} \quad (5.38)$$

This error is then used in a *controller equation* or *control law* to adjust the position x of a valve, which regulates the entering air flow rate V

$$x = x_{ss} + K_c \left(e + (1/T_i) \int_0^t e dt \right) \quad (5.39)$$

where x_{ss} is the *steady-state value position*, K_c is the *controller gain*, and T_i is the *controller integral time*. A principal objective in simulating the column might be the determination of K_c and T_i to achieve accurate, stable control of the outlet liquid temperature [to maintain $TL(0, t)$ close to TL_{set}]. The valve position x from equation (5.39) then sets the entering gas flow rate V through the valve equation

$$V = C_v \sqrt{\Delta p} x \quad (5.40)$$

where C_v is a constant for the valve and Δp is the pressure drop across the valve (assumed constant). The product $C_v \sqrt{\Delta p}$ is represented by the variable CVDP in the code in subroutines INITAL and DERV. Also, the valve position is constrained in the code (in subroutine DERV) to the limits $0 \leq x \leq 1$ corresponding to the valve fully closed ($x = 0$) to fully open ($x = 1$); V from equation (5.40) is then used in the advection groups of equations (5.29) and (5.30).

To implement equations (5.38) to (5.40), we must first compute the integral in equation (5.39): this can easily be done if the integral

$$e_i = \int_0^t e dt$$

is written as the equivalent ODE

$$de_i/dt = e, \quad e_i(0) = 0 \quad (5.41)(5.42)$$

Then, equation (5.41) can be integrated as any other ODE, subject to initial condition (5.42); thus COMMON/Y/ contains $e_i (= EI)$ and COMMON/F/ contains $e (= E)$.

Continuing the discussion of subroutine INITAL, after the COMMON area, an integer variable, METRIC, is set to 0 if the code is to be executed with English units or 1 if it is to be executed with metric units; the computed solution is the same in each case (except for the required conversion from one set of units to the other). Then, the model parameters are set for METRIC = 0 or METRIC = 1. Next, a set of constants is calculated (P2 to P6), which are used in other subroutines; calculation of constants in INITAL is done for efficiency since INITAL is called only once each run.

Finally, the initial conditions, equations (5.32) to (5.34) and (5.42), are set in DO loop 1 and the following statement (for EI). Note that COMMON/Y/ contains 34 dependent variables, so that 34 initial conditions must be set (DO loop 1 with N = 11 and the statement for EI). Note also that the gas temperature is set [i.e., TG(I) = 110 or 43.33], from which the gas internal energy is calculated [EV(I)] [the dependent variable required by equation (5.30)]. Now everything is defined to start the solution, and in fact, a call to DERV is included at the end of INITAL to confirm that all of the code in DERV will execute starting from these 34 initial conditions. In other words, if there is anything wrong with the coding of the model equations in DERV, this first call to DERV will reveal the errors.

Subroutine DERV is listed in Program 5.1b.

Program 5.1b Subroutine DERV for Equations (5.29) to (5.42)

```

SUBROUTINE DERV
COMMON/T/      T,   NSTOP,   NORUN
1      /Y/   Y(11), EV(11), TL(11),      EI
2      /F/   YT(11), EVT(11), TLT(11),     E
3      /S/   YZ(11), EPZ(11), TLZ(11)
4      /C/    V,     L,     G,      H,     HLA,   KYAV,
5          CL,   CPV,   CPA,   CVV,   CVA,   ZL,
6          DHVAP, S,     P1,     P2,     P3,     P4,
7          P5,     P6,     P7,   TG(11), EP(11), YS(11),
8          KC,     TI,   CVDP,   XSS,   TLSET,     X,
9          IP,     N,   METRIC
REAL        L,   KYAV,     KC
C...
C...  CONTROLLER EQUATIONS
E=TL(1)-TLSET
X=XSS+KC*(E+(1./TI)*EI)
IF(X.LT.0.)X=0.
IF(X.GT.1.)X=1.
V=CVDP*X
P1=V/(G*S)

```

continues

```

C...
C... MODEL ALGEBRA
DO 1 I=1,N
TG(I)=(EV(I)-Y(I)*DHVAP)/(CVA+Y(I)*CVV)
EP(I)=CPA*TG(I)+Y(I)*(CPV*TG(I)+DHVAP)
IF(METRIC.EQ.0)THEN
P=10.**(7.96681-3002.4/(378.4+TL(I)))
ELSE
P=10.**(7.96681-3002.4/(378.4+1.8*TL(I)+32.))
END IF
YS(I)=P/(760.-P)
1 CONTINUE
C...
C... BOUNDARY CONDITIONS
Y(1)=0.01
IF(METRIC.EQ.0)THEN
TL(N)=110.
TG(1)=110.
ELSE
TL(N)=43.33
TG(1)=43.33
END IF
EP(1)=CPA*TG(1)+Y(1)*(CPV*TG(1)+DHVAP)
C...
C... CONSTRAINTS FOR THE DEPENDENT VARIABLES AND ASSOCIATED
C... VARIABLES
DO 3 I=1,N
IF( Y(I).LT. 0.) Y(I)=0.
IF(YS(I).LT. 0.)YS(I)=0.
3 CONTINUE
C...
C... SPATIAL DERIVATIVES
CALL DSS020(0.,ZL,N, Y, YZ, 1.)
CALL DSS020(0.,ZL,N,EP,EPZ, 1.)
CALL DSS020(0.,ZL,N,TL,TLZ,-1.)
C...
C... PARTIAL DIFFERENTIAL EQUATIONS
DO 2 I=1,N
YT(I)=-P1*YZ(I)+P2*(YS(I)-Y(I))
P7=CVV*TG(I)+DHVAP
EVT(I)=-P1*EPZ(I)+P3*(TL(I)-TG(I))+P2*(YS(I)-Y(I))*P7
TLT(I)= P4*TLZ(I)-P5*(TL(I)-TG(I))-P6*(YS(I)-Y(I))*P7
2 CONTINUE
C...
C... ZERO THE TIME DERIVATIVES TO MAINTAIN THE BOUNDARY CONDITIONS
YT(1)=0.
EVT(1)=0.
TLT(N)=0.
RETURN
END

```

DERV again consists of two major sections: algebra and ODEs. The algebra includes the controller equations, (5.38) to (5.40). Note that when the error, e ($=E$), is calculated according to equation (5.38), it is also included as the last element of COMMON/F/, while the integral of e , e_i ($=EI$) is returned as the last element of COMMON/Y/ through the numerical integration of the 34th ODE [equation (5.41)]. Thus, EI is available for use in the controller equation, equation (5.39) (as are all of the dependent variables in COMMON/Y/).

Next some model algebra is performed to obtain variables that are later used in the ODEs. In particular, the air temperature, $TG(I)$, is computed from the air internal energy, $EV(I)$, and humidity, $Y(I)$. [Why are $EV(I)$ and $Y(I)$ available for the calculation of $TG(I)$?] Then, with $TG(I)$ available, the air enthalpy, $EP(I)$ is calculated so that it will be available for use in equation (5.30). Also, the vapor pressure of water P is calculated from the water temperature $TL(I)$ (why is $TL(I)$ available for the calculation of P ?), and finally, the saturation humidity $YS(I)$ is calculated from the water vapor pressure P and the total pressure, 760 mm Hg; $YS(I)$ is then available for use in equations (5.29), (5.30), and (5.31).

Next, the boundary conditions, equations (5.35) to (5.37), are set in DERV. Note that the air enthalpy at $z = 0$, $EP(1)$, is required for boundary condition (5.36). Similarly, the entering humidity at $z = 0$, $Y(1)$, and entering water temperature at $z = z_l$, $TL(N)$, are required by boundary conditions (5.35) and (5.37).

Since some of the model variables might assume unrealistic values, constraints to prevent this are applied next; in this case, the air humidity and saturation humidity are constrained to nonnegative values. Then, the three spatial derivatives in equations (5.29), (5.30), and (5.31) are computed by the five-point biased upwind approximations in subroutine DSS020.

Finally, all of the intermediate variables for the PDEs have been computed [the RHS variables of equations (5.29), (5.30) and (5.31)]. DO loop 2 therefore calculates the time derivatives of the PDEs over the N ($= 11$) spatial grid points. This gives a total of 33 time derivatives. Since E has already been calculated as one of the controller equations [equation (5.38)], all 34 ODE time derivatives have been evaluated when DO loop 2 is completed. Because of boundary conditions (5.35), (5.36), and (5.37), three of the t derivatives at the boundaries are zeroed to prevent any change in the boundary values of the corresponding dependent variables [$Y(1)$, $EV(1)$, and $TL(N)$].

This essentially completes the coding of the model, equations (5.29) to (5.42). Note in particular that even though the model is relatively complex, the NUMOL coding in subroutine DERV is relatively short and straightforward. In fact, the coding in subroutine PRINT, which only displays the solution in numerical and graphical form, is considerably longer than the coding in DERV; see Program 5.1c.

The following points should be noted about subroutine PRINT. The COMMON area is the same as in subroutines INITAL and DERV. Then, two sets of arrays are dimensioned (via DIMENSION statements) for plotting of the numerical solution and calculation of the ODE temporal eigenvalues. The first dependent variable in COMMON/Y/, Y, is EQUIV-

Program 5.1c Subroutine PRINT for Equations (5.29) to (5.42)

```

SUBROUTINE PRINT(NI ,NO)
COMMON/T/      T,   NSTOP,   NORUN
1      /Y/   Y(11), EV(11), TL(11),      EI
2      /F/   YT(11),EVT(11),TLT(11),      E
3      /S/   YZ(11),EPZ(11),TLZ(11)
4      /C/    V,     L,     G,     H,   HLA V,   KYAV,
5          CL,   CPV,   CPA,   CVV,   CVA,   ZL,
6          DHVAP,   S,     P1,     P2,   P3,   P4,
7          P5,     P6,     P7,   TG(11), EP(11), YS(11),
8          KC,     TI,   CVDP,   XSS,   TLSET,   X,
9          IP,     N,   METRIC
REAL      L,   KYAV,   KC
C...
C... DIMENSION THE ARRAYS FOR THE PLOTTED SOLUTION, INDIVIDUAL TERMS
C... IN THE PDES
DIMENSION      TLP(101),        TGP(101),        YP(101),
1          VP(101),        XP(101),        ERP(101),
2          TP(101),        TEP(11),        TGLP(2,11)
C...
C... DIMENSION THE ARRAYS FOR SUBROUTINE JMAP AND EIGEN
DIMENSION      A(34,34),        WR(34),        WI(34),
1          Z(34,34),        RW(34),        IW(34),
2          SV(34),        SVOLD(34),
3          F(34),        FOLD(34)
C...
C... EQUIVALENCE THE STATE VARIABLE VECTOR, S(34), TO THE DEPENDENT
C... VARIABLE VECTORS IN COMMON/Y/
EQUIVALENCE (Y(1),SV(1)), (YT(1),F(1))
C...
C... PRINT A DETAILED SOLUTION (THE INDIVIDUAL TERMS IN THE PDES AND
C... THE DEPENDENT VARIABLES) FOR IP = 1, 51, 101 CALLS TO SUBROUTINE
C... PRINT. FIRST, UPDATE ALL OF THE MODEL CALCULATIONS BY A CALL
C... TO SUBROUTINE DERV (THE INITIAL-VALUE INTEGRATION THROUGH COMMON
C... /Y/ AND /F/ IS DONE CORRECTLY WITHOUT THIS UPDATE, BUT SOME OF THE
C... MODEL VARIABLES MAY BE ONE INTEGRATION TIME STEP BEHIND IN TIME,
C... T, WHEN THEY ARE PRINTED OUT IN SUBROUTINE PRINT IF THE FOLLOWING
C... CALL TO SUBROUTINE DERV IS NOT EXECUTED)
IP=IP+1
IF((IP.NE.1).AND.(IP.NE.51).AND.(IP.NE.101))GO TO 100
CALL DERV
C...
C... **** GAS-PHASE MATERIAL BALANCE ****
C... **** (1)
C...
C...      -(V/(G*S))*Y
C...      Z
DO 1 I=1,N
TEP(I)=-P1*YZ(I)
CONTINUE
1      N2=1
WRITE(NO,2)T,(TEP(I),I=1,N,N2)
2      FORMAT(1H1,5H T = ,F7.3,//,
1          25H -(V/(G*S))*YZ           ,5E11.3,/,14X,6E11.3,//)
C...
C... **** (2)
C...
C...      (KY*AV/G)*(YS - Y)
DO 3 I=1,N
TEP(I)=P3*(YS(I)-Y(I))
CONTINUE
3      WRITE(NO,4)(TEP(I),I=1,N,N2)

```

continues

```

4      FORMAT(25H (KA*AV/G)*(YS - Y) ,5E11.3,,14X,6E11.3,,/) C...
C... ****
C... Y
C... T
C... WRITE(NO,5)(YT(I),I=1,N,N2)
5      FORMAT(25H YT ,5E11.3,,14X,6E11.3,,/) C...
C... ****
C... Y
C... WRITE(NO,6)(Y(I),I=1,N,N2)
6      FORMAT(25H Y ,5E11.3,,14X,6E11.3,,/) C...
C... ****
C... EQUILIBRIUM RELATIONSHIP
C... ****
C... YS
C... WRITE(NO,7)(YS(I),I=1,N,N2)
7      FORMAT(25H YS ,5E11.3,,14X,6E11.3,,/) C...
C... ****
C... GAS-PHASE ENERGY BALANCE
C... ****
C... -(V/(G*S))*EP = -(V/(G*S))*(CPA*TG + Y*(CPV*TG + DHVAP)) Z
C... DO 8 I=1,N
C... TEP(I)=-P1*EPZ(I)
8      CONTINUE
C... WRITE(NO,9)(TEP(I),I=1,N,N2)
9      FORMAT(25H -(V/(G*S))*EPZ ,5E11.3,,14X,6E11.3,,/) C...
C... ****
C... (H*AV/G)*(TL - TG)
C... DO 10 I=1,N
C... TEP(I)=P3*(TL(I)-TG(I))
10     CONTINUE
C... WRITE(NO,11)(TEP(I),I=1,N,N2)
11    FORMAT(25H -(H*AV/G)*(TL - TG) ,5E11.3,,14X,6E11.3,,/) C...
C... ****
C... (KY*AV/G)*(YS - Y)*(CVV*TG + DHVAP)
C... DO 12 I=1,N
C... TEP(I)=-P2*(YS(I)-Y(I))*(CVV*TG(I)+DHVAP)
12     CONTINUE
C... WRITE(NO,13)(TEP(I),I=1,N,N2)
13    FORMAT(25H (KY*AV/G)*(YS - Y)*(...) ,5E11.3,,14X,6E11.3,,/) C...
C... ****
C... EV = (CVA*TG + Y*(CVV*TG + DHVAP)) T
C... WRITE(NO,14)(EVT(I),I=1,N,N2)
14    FORMAT(25H EVT ,5E11.3,,14X,6E11.3,,/) C...
C... ****
C... TG
C... WRITE(NO,15)(TG(I),I=1,N,N2)
15    FORMAT(25H TG ,5E11.3,,14X,6E11.3,,/)


```

continues

```

C...
C... *****
C... LIQUID-PHASE ENERGY BALANCE
C... *****
C... -(L/(H*S))*TL
C... Z
C... DO 16 I=1,N
C... TEP(I)=P4*TLZ(I)
16 CONTINUE
C... WRITE(NO,17)(TEP(I),I=1,N,N2)
17 FORMAT(25H -(L/(H*S))*TLZ ,5E11.3,/,14X,6E11.3,//)
C...
C... *****
C... -(H*AV/(CL*H))*(TL - TG)
DO 18 I=1,N
TEP(I)=-P5*(TL(I)-TG(I))
18 CONTINUE
C... WRITE(NO,19)(TEP(I),I=1,N,N2)
19 FORMAT(25H -(H*AV/(CL*H))*(TL-TG) ,5E11.3,/,14X,6E11.3,//)
C...
C... *****
C... -(KY*AV/(CL*H))*(YS - Y)*(CVV*TG + DHVAP)
DO 20 I=1,N
TEP(I)=-P6*(YS(I)-Y(I))*(CVV*TG(I)+DHVAP)
20 CONTINUE
C... WRITE(NO,21)(TEP(I),I=1,N,N2)
21 FORMAT(25H -(KY*AV/(CL*H))*(YS-Y) ,5E11.3,/,14X,6E11.3,//)
C...
C... *****
C... TL
C... T
C... WRITE(NO,22)(TLT(I),I=1,N,N2)
22 FORMAT(25H TLT ,5E11.3,/,14X,6E11.3,//)
C...
C... *****
C... TL
C... WRITE(NO,23)(TL(I),I=1,N,N2)
23 FORMAT(25H TL ,5E11.3,/,14X,6E11.3,//)
C...
C... *****
C... MAP THE JACOBIAN MATRIX OF THE 34 ODES DEFINED IN COMMON/Y/ AND
C... /F/
C... NODE=34
CALL JMAP(NODE,A,SV,SVOLD,F,FOLD)
C...
C... COMPUTE THE TEMPORAL EIGENVALUES OF THE 34 ODES DEFINED IN COMMON
C... /Y/ AND /F/
CALL EIGEN(NODE,A,WR,WI,Z,RW,IW)
C...
C... PLOT TG AND TL VS Z
DO 24 I=1,N
TGLP(1,I)=TG(I)
TGLP(2,I)=TL(I)
TEP(I)=ZL*FLOAT(I-1)/FLOAT(N-1)
CONTINUE
C... CALL TPLOTS(2,N,TEP,TGLP)
C... WRITE(NO,25)T
24

```

continues

```

25   FORMAT(1H ,/,36H 1 - TG(Z,T) , 2 - TL(Z,T) VS Z, T = ,F7.3)
C...
C... STORE THE NUMERICAL SOLUTION FOR PLOTTING (VS T)
100  TLP(IP)=TL(1)
      TGP(IP)=TG(N)
      YP(IP)= Y(N)
      VP(IP)= V
      XP(IP)= X
      ERP(IP)= E
      TP(IP)= T
C...
C... PLOT THE SOLUTION
      IF(IP.LT.101)RETURN
      CALL TPLOTS(1,IP,TP,TLP)
      WRITE(NO,101)
101  FORMAT(1H ,/,30H TL(0,T) VS T ) )
      CALL TPLOTS(1,IP,TP,TGP)
      WRITE(NO,102)
102  FORMAT(1H ,/,30H TG(ZL,T) VS T ) )
      CALL TPLOTS(1,IP,TP, YP)
      WRITE(NO,103)
103  FORMAT(1H ,/,30H Y(ZL,T) VS T ) )
      CALL TPLOTS(1,IP,TP, VP)
      WRITE(NO,104)
104  FORMAT(1H ,/,30H V(T) VS T ) )
      CALL TPLOTS(1,IP,TP, XP)
      WRITE(NO,105)
105  FORMAT(1H ,/,30H X(T) VS T ) )
      CALL TPLOTS(1,IP,TP,ERP)
      WRITE(NO,106)
106  FORMAT(1H ,/,30H E(T) VS T ) )
      RETURN
      END

```

ALENCEd to the array SV for the calculation of the eigenvalues; the same is true for the first derivative vector in COMMON/F/, YT, which is EQUIVALENCEd to the array F. [As we might expect from the discussion of ODE eigenvalues in Chapter 4, the ODE dependent variable vector in COMMON/Y/ and the derivative vector in COMMON/F/ will be required for the calculation of the ODE eigenvalues; they are accessed via single vectors, SV and F, through the use of the EQUIVALENCE statements rather than using a series of vectors (arrays) such as Y, EV, TL, EI and YT, EVT, TLT, E.]

Next, we have a rather long series of WRITE statements that can produce a substantial quantity of printed numerical output. In order to limit the output to a reasonable volume, these WRITE statements are executed only for the first, 51st, and 101st calls to PRINT (note the use of the counter IP). The idea in executing the WRITE statements is to print every RHS term in the PDEs, equations (5.29) to (5.31), so that we can observe the contribution of each term, e.g., in equation (5.29), we can observe the relative contributions of the convection of water in the air $-(V/(GS)y_z$ and the mass transfer of water $(k_y a/G)(y_s - y)$; additionally, we can observe each variable, e.g., the derivative y_t and the humidity y .

This detailed output can then give us a *picture of what the PDEs are doing in terms of all of their variables and terms*. This comprehensive picture is invaluable in understanding the behavior and performance of the model, particularly when we are first executing the code, and especially if we encounter difficulties in getting the code to run and give reasonable output; for example, we might observe that a PDE RHS term appears to be out of line in the sense that its magnitude far exceeds that of the other RHS terms, and we may therefore check that it is coded correctly. Also, looking at the magnitudes of the temporal derivatives in COMMON/F/ (YT, EVT, TLT, and E) gives an indication of whether the PDE RHS terms sum to reasonable values; if the temporal derivatives are unrealistic, the model will not have realistic dynamic behavior. We recommend that detailed, comprehensive output such as in subroutine PRINT be used when a new NUMOL code of a complex physical system is developed; if the numerical output is too voluminous to comprehend clearly, then the variables, PDE RHS terms and spatial and temporal derivatives can be plotted to observe their behavior.

Next, the Jacobian matrix of the 34 ODEs is mapped by a call to subroutine JMAP (the map is subsequently presented and described). JMAP returns a numerical approximation of the Jacobian matrix [in array A(34, 34)], which is an input in the call to subroutine EIGEN. The 34 eigenvalues of A are computed and printed in subroutine EIGEN; the eigenvalues are also returned from EIGEN as the real and imaginary parts in arrays WR and WI, respectively. The eigenvalues are subsequently presented and discussed.

Finally, some point plotting of the numerical solution is produced by a series of calls to subroutine TPLOTS. DO loop 24 and the following call to TPLOTS produces spatial profiles of the air and water temperatures [$TG(z, t)$ and $TL(z, t)$ vs. z at a series of values of t]. Then in a series of six calls to TPLOTS, $TL(0, t)$, $TG(z_l, t)$, $y(z_l, t)$, $V(t)$, $x(t)$, and $e(t)$ are plotted against t [$V(t)$, $x(t)$, and $e(t)$ are given by the controller equations (5.38) to (5.40)].

The data for the model are listed in Program 5.1d; two runs are programmed to execute the model with English and metric units.

Program 5.1d Data for Equations (5.29) to (5.42)

```

PI CONTROL OF A HUMIDIFICATION COLUMN - ENGLISH UNITS
0.          0.5          0.005
 34 1000    1      1 REL 0.001
PI CONTROL OF A HUMIDIFICATION COLUMN - METRIC UNITS
0.          0.5          0.005
 34 1000    1      1 REL 0.001
END OF RUNS

```

Note in particular that the data specify:

- (1) 101 calls to subroutine PRINT to provide enough points for the plots (through the calls to subroutine TPLOTS).
- (2) 34 ODEs, as discussed previously (again, this agrees with the number of elements in COMMON/Y/ and /F/).

Only excerpts from the output for the second set of data are listed in Table 5.1 at $t = 0$ and 0.5

Table 5.1 Numerical Output from Programs 5.1a to 5.1d

RUN NO.	-	2	PI CONTROL OF A HUMIDIFICATION COLUMN - METRIC UNITS			
INITIAL T	-	0.000E+00				
FINAL T	-	0.500E+00				
PRINT T	-	0.500E-02				
NUMBER OF DIFFERENTIAL EQUATIONS	-	34				
MAXIMUM INTEGRATION ERROR	-	0.100E-02				
T	=	0.000				
$-(V/(G*S))*YZ$		-0.873E-06 0.873E-06 0.873E-06 -0.262E-05 -0.262E-05				
		-0.262E-05 -0.262E-05 -0.262E-05 -0.262E-05 0.000E+00				
$(KA*AV/G)*(YS - Y)$	0.425E+03 0.425E+03	0.425E+03 0.425E+03 0.425E+03 0.425E+03	0.425E+03 0.425E+03 0.425E+03 0.425E+03	0.425E+03 0.425E+03 0.425E+03 0.425E+03	0.425E+03 0.425E+03 0.425E+03 0.425E+03	
YT	0.000E+00 0.349E+02	0.349E+02 0.349E+02	0.349E+02 0.349E+02	0.349E+02 0.349E+02	0.349E+02 0.349E+02	
Y	0.100E-01 0.100E-01	0.100E-01 0.100E-01	0.100E-01 0.100E-01	0.100E-01 0.100E-01	0.100E-01 0.100E-01	
YS	0.950E-01 0.950E-01	0.950E-01 0.950E-01	0.950E-01 0.950E-01	0.950E-01 0.950E-01	0.950E-01 0.950E-01	
$-(V/(G*S))*EPZ$	0.114E+00 0.114E+00	0.429E-01 0.114E+00	-0.429E-01 0.114E+00	0.114E+00 0.114E+00	0.114E+00 0.000E+00	
	
TLT	-0.375E+03 -0.375E+03	-0.375E+03 -0.375E+03	-0.375E+03 -0.375E+03	-0.375E+03 -0.375E+03	-0.375E+03 -0.375E+03	
	-0.375E+03 -0.375E+03	-0.375E+03 -0.375E+03	-0.375E+03 -0.375E+03	-0.375E+03 -0.375E+03	-0.375E+03 -0.375E+03	
TL	0.433E+02 0.433E+02	0.433E+02 0.433E+02	0.433E+02 0.433E+02	0.433E+02 0.433E+02	0.433E+02 0.433E+02	

continues

DEPENDENT VARIABLE COLUMN INDEX J (FOR YJ) IS PRINTED HORIZONTALLY
 DERIVATIVE ROW INDEX I (FOR DYI/DT = FI(Y1,Y2,...,YJ,...,YN) IS PRINTED VERTICALLY

JACOBIAN MATRIX ELEMENT IN THE MAP WITH INDICES I,J IS FOR PFI/PYJ WHERE P DENOTES A PARTIAL DERIVATIVE

	1111111111222222222233333		
	1234567890123456789012345678901234		
1			
2	8887	5	
3	8787	5	
4	8888	5	
5	78888	5	
6	78888	5	
7	78888	5	
8	78888	5	
9	78888	5	
10	78888	5	
11	88999	6	
12			
13	9999	8987	9
14	9999	8787	9
15	9999	8988	9
16	99999	78988	9
17	99999	78988	9
18	99999	78988	9
19	99999	78988	9
20	99999	78988	9
21	99999	78988	9
22	99999	89999	9
23		77776	
24	8	4	67766
25	8	4	67766
26	8	4	67766
27	8	4	67766
28	8	4	67766
29	8	4	67766
30	8	4	6776
31	8	4	6666
32	8	4	6676
33			
34		4	

I	REAL	IMAG
1	0.000	0.000
2	-2568.185	8983.811
3	-2568.185	-8983.811
4	-6946.026	8571.773
5	-6946.026	-8571.773
6	-1621.667	6522.824
7	-1621.667	-6522.824
8	-7427.623	6059.220
9	-7427.623	-6059.220
10	-4800.233	6222.272
11	-4800.233	-6222.272
12	-5150.282	4398.196
13	-5150.282	-4398.196

continues

14	-7035.274	3471.002
15	-7035.274	-3471.002
16	-7970.468	1641.967
17	-7970.468	-1641.967
18	-4866.332	2516.847
19	-4866.332	-2516.847
20	-5544.144	1191.733
21	-5544.144	-1191.733
22	-56.981	143.727
23	-56.981	-143.727
24	-131.102	146.656
25	-131.102	-146.656
26	-179.769	90.691
27	-179.769	-90.691
28	-92.898	98.245
29	-92.898	-98.245
30	-46.814	0.000
31	-198.603	0.000
32	0.000	0.000
33	0.000	0.000
34	0.000	0.000

T = 0.500

- (V/(G*S))*YZ	-0.158E+02	-0.149E+02	-0.142E+02	-0.137E+02	-0.135E+02	
	-0.134E+02	-0.135E+02	-0.138E+02	-0.143E+02	-0.152E+02	-0.164E+02

(KA*AV/G)*(YS - Y)	0.193E+03	0.182E+03	0.173E+03	0.168E+03	0.164E+03	
	0.163E+03	0.164E+03	0.168E+03	0.175E+03	0.185E+03	0.200E+03

YT	0.000E+00	0.134E-02	0.242E-02	0.346E-02	0.445E-02	
	0.528E-02	0.588E-02	0.633E-02	0.695E-02	0.736E-02	0.799E-02

Y	0.100E-01	0.149E-01	0.195E-01	0.239E-01	0.282E-01	
	0.324E-01	0.367E-01	0.410E-01	0.454E-01	0.501E-01	0.551E-01

YS	0.486E-01	0.512E-01	0.542E-01	0.574E-01	0.610E-01	
	0.650E-01	0.695E-01	0.746E-01	0.804E-01	0.871E-01	0.950E-01

- (V/(G*S))*EPZ	-0.957E+05	-0.102E+06	-0.108E+06	-0.113E+06	-0.119E+06	
	-0.125E+06	-0.132E+06	-0.140E+06	-0.150E+06	-0.163E+06	-0.179E+06

.

.

TLT	0.616E+00	0.706E+00	0.750E+00	0.764E+00	0.769E+00	
	0.764E+00	0.722E+00	0.612E+00	0.432E+00	0.217E+00	0.000E+00

TL	0.318E+02	0.327E+02	0.336E+02	0.346E+02	0.357E+02	
	0.368E+02	0.379E+02	0.391E+02	0.404E+02	0.418E+02	0.433E+02

continues

DEPENDENT VARIABLE COLUMN INDEX J (FOR YJ) IS PRINTED HORIZONTALLY
 DERIVATIVE ROW INDEX I (FOR DYI/DT = FI(Y1,Y2,...,YJ,...,YN) IS PRINTED VERTICALLY

JACOBIAN MATRIX ELEMENT IN THE MAP WITH INDICES I,J IS FOR PFI/PYJ
 WHERE P DENOTES A PARTIAL DERIVATIVE

	111111111122222222233333			
	1234567890123456789012345678901234			
1				
2	8887		55	6
3	8787		5 5	6
4	8887		4 5	5
5	78887		4 5	5
6	78887		4 5	5
7	78887		4 5	5
8	78887		4 5	5
9	78887		5 5	6
10	78887		5 5	6
11	78898		5	66
12				
13	9999	8887	89	9
14	9999	8787	8 9	9
15	9999	8888	8 9	9
16	99999	78888	8 9	9
17	99999	78888	8 9	9
18	99999	78888	8 9	9
19	99999	78888	9 9	9
20	99999	78888	9 9	9
21	99999	788889	9 9	9
22	99999	889989		99
23			77776	
24	8	5	67766	
25	8	5	67766	
26	8	5	67766	
27	8	5	67766	
28	8	5	67766	
29	8	4	67766	
30	8	4	6776	
31	8	4	6666	
32	8	4	6676	
33				
34			4	

I	REAL	IMAG
1	-4286.629	4815.080
2	-4286.629	-4815.080
3	-1849.207	5045.251
4	-1849.207	-5045.251
5	-1089.750	3658.147
6	-1089.750	-3658.147
7	-4556.249	3406.655
8	-4556.249	-3406.655
9	-2873.500	3490.155
10	-2873.500	-3490.155
11	-4858.477	920.015
12	-4858.477	-920.015
13	-3070.328	2466.465
14	-3070.328	-2466.465
15	-4342.380	1953.128

continues

16	-4342.380	-1953.128
17	-3291.221	669.164
18	-3291.221	-669.164
19	-2911.376	1413.926
20	-2911.376	-1413.926
21	-222.718	68.679
22	-222.718	-68.679
23	-150.789	156.172
24	-150.789	-156.172
25	-74.700	151.225
26	-74.700	-151.225
27	-51.984	131.998
28	-51.984	-131.998
29	-43.668	31.864
30	-43.668	-31.864
31	-1.569	0.000
32	0.000	0.000
33	0.000	0.000
34	0.000	0.000

The following points should be noted about the output:

The Jacobian matrix maps are significantly different for $t = 0$ and 0.5 , which is due to the nonlinearity of the 34 ODEs (recall from Section 4.2 that nonlinear ODEs have a nonconstant Jacobian matrix). When we first observed the map at $t = 0.5$, we noted the "N"s that appeared in rows 2 to 22 and joked that this clearly resulted from the nonlinearity of the 34 ODEs.

Each Jacobian matrix map has an integer index across the top indicating the number of a dependent variable in the 34 ODE system. Thus, 1 corresponds to the first dependent variable in COMMON/Y/, Y(1), while 34 corresponds to EI. Each map also has an integer index down the left side indicating the number of an ODE; 1 corresponds to the first derivative in COMMON/F/, YT(1), and 34 corresponds to E. Therefore, from the map, we can determine which dependent variables appear in the RHS of any ODE.

For example, ODE 1 (the first row of the map) at either $t = 0$ to 0.5 has no entries. In other words, this row corresponds to the ODE

$$dy_1/dt = c \quad (\text{a constant})$$

or in terms of the Fortran notation of the PDE model

$$\text{YT}(1) = 0$$

corresponding to boundary condition (5.35), $\text{Y}(1) = 0.01$.

ODE 2 (the second row of the map) at $t = 0.5$ has the form

$$dy_2/dt = f_2(y_2, y_3, y_4, y_5, y_{23}, y_{24}, y_{34})$$

or in terms of the Fortran notation of the PDE model

$$\text{YT}(2) = f_2(\text{Y}(2), \text{Y}(3), \text{Y}(4), \text{Y}(5), \text{TL}(1), \text{TL}(2), \text{EI})$$

The dependency on $Y(2)$, $Y(3)$, $Y(4)$, and $Y(5)$ results from the five-point biased upwind approximation of the spatial derivative y_z in equation (5.29) [Note the band of five diagonal elements in ODEs 2 to 11, ODEs 13 to 22, and ODEs 23 to 32 resulting from the five-point approximation of the spatial derivatives y_z , EP_z and TL_z in equations (5.29), (5.30), and (5.31), respectively]. The dependency on $TL(1)$ and EI results from the air flow rate V set by the controller according to equations (5.38) to (5.40); in fact, V appears in ODEs 2 to 11 and 13 to 22 via PDEs (5.29) and (5.30). Finally, the dependency on $TL(2)$ results from the calculation of the saturation humidity, $YS(2)$.

ODE 12 has the form

$$dy_{12}/dt = c \quad (\text{a constant})$$

or in terms of the Fortran notation of the PDE model

$$EVT(1) = 0$$

corresponding to boundary condition (5.36), $EP(1) = \text{entering internal energy (a constant)}$.

ODE 33 has the form

$$dy_{33}/dt = c \quad (\text{a constant})$$

or in terms of the Fortran notation of the PDE model

$$TLT(N) = 0$$

corresponding to boundary condition (5.37), $TL(N) = 43.33$.

ODE 34 has the form

$$dy_{34}/dt = f_{34}(y_{23})$$

or in terms of the Fortran notation of the PDE model

$$e = f_{34}(TL(0, t))$$

corresponding to the controller ODE [equation (5.41)].

Thus the Jacobian matrix map gives a detailed picture of the structure of the ODE system. In addition to printing the map, subroutine JMAP, also computes a numerical approximation of the ODE Jacobian matrix; i.e., it approximates by finite differences the numerical values of the elements of the Jacobian matrix at the current values of the dependent variables [$Y(1)$ to EI in this case]. However, because of the space required to print all of the numerical values of these elements, they are instead represented by shorter integer numbers that represent their magnitudes. Thus, for example, an element that is printed as a 7 is a factor of 10 greater than an element printed as a 6, a factor of 10^2 greater than an element

printed as a 5, etc.; in other words, the integers are indicators of order of magnitude (powers of 10). However, the numerical Jacobian is retained in an array that is available as an output from subroutine JMAP (array A in the preceding call to JMAP). This array can then be sent to an eigenvalue routine to calculate the eigenvalues of the numerical Jacobian. This is done within subroutine EIGEN by calling a series of subroutines from the EISPACK Library (e.g., array A containing the numerical Jacobian is an input in the preceding call to EIGEN). EIGEN then prints the computed eigenvalues and provides them as outputs through its arguments (arrays WR and WI in the preceding call).

The 34 eigenvalues all have negative real parts at $t = 0$ and 0.5. This suggests that the simulation is stable, at least at these times. However, since this model is quite nonlinear, the eigenvalues of the linearized ODEs do not ensure stability (the linearized ODEs which produce the eigenvalues only approximate the nonlinear ODEs). At least three eigenvalues are zero (both real and imaginary parts), corresponding to the three boundary conditions for which the temporal derivatives are set to zero [$YT(1) = EVT(1) = TLT(N) = 0$ in subroutine DERV]. The smallest nonzero eigenvalue at $t = 0$ is $-46.814 + j0.000$ ($j = \sqrt{-1}$), which indicates that the problem time scale is approximately 0.2 ($e^{-46.772(0.2)} \doteq e^{-10}$ indicates that the exponential for this eigenvalue has decayed to insignificance, and, of course, the exponentials for all of the other eigenvalues will decay faster). At $t = 0.5$, the smallest nonzero eigenvalue is $-1.569 + j0.000$, indicating an approximate time scale of 7. The NUMOL solution was terminated at $t = 0.5$, which is consistent with these eigenvalues since this type of time-scale analysis is, at best, only approximate. Also, do not conclude that a particular eigenvalue is associated with a particular ODE. The eigenvalues are defined by the total system of 34 ODEs as reflected through the Jacobian matrix.

The numerical solution has been abbreviated in the preceding output to the point that it is not possible to gain an overall perspective of its properties (also, the point plots produced by the calls to TPLOTS are not reproduced here because of space limitations). However, the exiting liquid temperature, $TL(0, t)$, does closely approach the set point $TLSET = 32.22^\circ\text{C}$ at $t = 0.5$, which is interesting since this is *below* both the entering liquid temperature [$TL(N) = 43.33$] and the entering air temperature [$TG(1) = 43.33$]. You might naturally ask how this could happen. Remember, though, that humidification (in this case, evaporation) takes place and, in fact, the humidity of the air increases significantly from the entering value of $Y(1) = 0.01$ before it leaves the column, i.e., $Y(N) > 0.01$. Thus, the humidification column performs the function of a water-cooling tower, much like in a nuclear or coal-fired power plant, and the vapor

commonly visible from a cooling tower is a manifestation of the high exiting-air humidity.

This example of the humidification column has been discussed in detail to illustrate how a PDE model is formulated and coded within the NUMOL framework. This application also illustrates the Jacobian mapping and eigenvalue analysis of the approximating ODEs. We have performed similar analyses for a variety of physical and chemical systems; a list of documented NUMOL applications available from the author is given in Appendix C.

In the concluding chapter, we consider the NUMOL solution of some two- and three-dimensional PDEs, and the application of the NUMOL using an adaptive grid to achieve better spatial resolution of the solution.

References

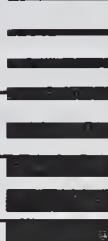
- (5.1) Stang, Gilbert, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, MA, 1986.
- (5.2) Richtmyer, Robert D., and K. W. Morton, *Difference Methods for Initial-value Problems*, Wiley-Interscience, New York, 1967.

Problems

- (5.1) Verify that equation (5.19) follows from equation (5.18) (this must be done for all possible values of the Fourier number k).
- (5.2) Judging from equation (5.16), would you expect that using a finer spatial grid (smaller Δx) to improve accuracy would move the eigenvalues of equation (5.14) closer to the imaginary axis, and therefore possibly cause a problem with stability when using the BDF methods of order greater than 3?
- (5.3) What limits are placed on the integration step size Δt by the eigenvalues of
 - (a) Equation (5.9)?
 - (b) Equation (5.16)?
 - (c) Equation (5.26)?
 when using the *implicit* Euler method (recall the stability criterion of the implicit Euler method discussed in Section 4.4).
- (5.4) Derive the three-point approximation of equation (5.23) (use the methods of Section 3.2).

- (5.5) Why is the initial condition of equation (5.41) zero [i.e., equation (5.42)]?
- (5.6) Identify some sources of nonlinearity in the humidification column model, equations (5.29) to (5.42).
- (5.7) Study the code in subroutine JMAP to determine how an ODE Jacobian matrix is computed and mapped, particularly for a nonlinear system such as the 34 ODEs in the NUMOL approximation of equations (5.29) to (5.42).
- (5.8) Prove that if an ODE has a row of zeros in the map of the Jacobian matrix, it produces a zero eigenvalue for the Jacobian matrix (this was demonstrated by three zero eigenvalues for the Jacobian matrix of the 34 ODEs representing the humidification column of Figure 5.1).
- (5.9) Study the plotted output of Programs 5.1a to 5.1d to gain some appreciation of how the controller defined by equations (5.38) to (5.40) performs. Would the exiting liquid temperature $TL(0, t)(= TL(1))$ eventually reach the set point TL_{set} ($= TLSET = 32.22$) if enough time passes?
- (5.10) Execute Programs 5.1a to 5.1d without the integral term in equation (5.39) [this can be done relatively easily by using equation (5.39) as it is stated, but using a large value of the integral time T_i to effectively drop out the integral term]. Does the controller perform as expected?

6



Additional Applications: Multidimensional PDEs and Adaptive Grids

In this final chapter, we consider by example some additional applications of the NUMOL, particularly the solution of multidimensional PDEs, and the use of adaptive grids to resolve sharp spatial variations in PDE solutions. Before proceeding to these topics, however, we consider the classification of PDEs based primarily on the ideas introduced in previous chapters.

6.1 Classification of PDEs

The examples discussed in the previous chapters suggest that the NUMOL can be applied to a general set of PDEs of the form

$$\begin{aligned} u_{1t} &= f_1(\bar{x}, t, u_1, u_2, \dots, u_n, u_{1\bar{x}}, u_{2\bar{x}}, \dots, \\ &\quad u_{n\bar{x}}, u_{1\bar{x}\bar{x}}, u_{2\bar{x}\bar{x}}, \dots, u_{n\bar{x}\bar{x}}, \dots) \\ u_{2t} &= f_2(\bar{x}, t, u_1, u_2, \dots, u_n, u_{1\bar{x}}, u_{2\bar{x}}, \dots, \\ &\quad u_{n\bar{x}}, u_{1\bar{x}\bar{x}}, u_{2\bar{x}\bar{x}}, \dots, u_{n\bar{x}\bar{x}}, \dots) \\ &\quad \cdot \qquad \qquad \qquad \cdot \\ &\quad \cdot \qquad \qquad \qquad \cdot \\ &\quad \cdot \qquad \qquad \qquad \cdot \\ u_{nt} &= f_n(\bar{x}, t, u_1, u_2, \dots, u_n, u_{1\bar{x}}, u_{2\bar{x}}, \dots, \\ &\quad u_{n\bar{x}}, u_{1\bar{x}\bar{x}}, u_{2\bar{x}\bar{x}}, \dots, u_{n\bar{x}\bar{x}}, \dots) \end{aligned} \tag{6.1}$$

where

u_1, u_2, \dots, u_n	vector of dependent variables of length n to be computed by the NUMOL
t	initial-value independent variable, typically time
f_1, f_2, \dots, f_n	vector of RHS functions defined for a particular PDE problem
\bar{x}	vector of boundary-value (spatial) independent variables, e.g., $[x, y, z]^T$ for Cartesian coordinates, $[r, \theta, z]^T$ for cylindrical coordinates, $[r, \theta, \phi]^T$ for spherical coordinates

As usual, a subscript with respect to t or \bar{x} indicates a partial derivative with respect to t or \bar{x} .

Equation (6.1) also requires an initial-condition vector

$$u_1(\bar{x}, t_0) = g_1(\bar{x}), u_2(\bar{x}, t_0) = g_2(\bar{x}), \dots, u_n(\bar{x}, t_0) = g_n(\bar{x}) \quad (6.2)$$

and a vector of boundary conditions

$$\begin{aligned} h_1(\bar{x}_b, t, u_1(\bar{x}_b, t), u_2(\bar{x}_b, t), \dots, u_n(\bar{x}_b, t), u_{1\bar{x}}(\bar{x}_b, t), \\ u_{2\bar{x}}(\bar{x}_b, t), \dots, u_{n\bar{x}}(\bar{x}_b, t). \dots) = 0 \\ h_2(\bar{x}_b, t, u_1(\bar{x}_b, t), u_2(\bar{x}_b, t), \dots, u_n(\bar{x}_b, t), u_{1\bar{x}}(\bar{x}_b, t), \\ u_{2\bar{x}}(\bar{x}_b, t), \dots, u_{n\bar{x}}(\bar{x}_b, t). \dots) = 0 \end{aligned} \quad (6.3)$$

where

t_0	initial value of t
g_1, g_2, \dots, g_n	vector of initial condition functions
h_1, h_2, \dots	vector of boundary condition functions
\bar{x}_b	boundary values of \bar{x}

The length of the boundary-condition vector $[h_1, h_2, \dots]^T$ cannot be stated generally for equation (6.1) since it will depend on the number and order of the spatial derivatives in equation (6.1). Also, \bar{x}_b , which generally denotes the boundary values of \bar{x} , cannot be stated more explicitly since it will depend on the number of boundary-value independent variables in equations (6.1) and (6.2) (typically, one, two, or three for each PDE).

Equations (6.1), (6.2), and (6.3) can be stated in a more concise vector form as

$$\bar{u}_t = \bar{f}(\bar{x}, t, \bar{u}, \bar{u}_{\bar{x}}, \bar{u}_{\bar{x}\bar{x}}, \dots) \quad (6.4)$$

$$\bar{u}(\bar{x}, t_0) = \bar{g}(\bar{x}) \quad (6.5)$$

$$\bar{h}(\bar{x}_b, t, \bar{u}(\bar{x}_b, t), \bar{u}_{\bar{x}}(\bar{x}_b, t), \dots) = \bar{0} \quad (6.6)$$

With this statement of a general system of PDEs, we can now consider a geometric classification of PDEs that can be useful in preparing an NUMOL code for a specific problem.

(1) *Elliptic PDEs.* If all of the derivatives with respect to t in equation (6.4) are identically zero, then we have a problem that is entirely of the boundary-value type in \bar{x}_b :

$$0 = \bar{f}(\bar{x}, \bar{u}, \bar{u}_{\bar{x}}, \bar{u}_{\bar{x}\bar{x}}, \dots) \quad (6.7)$$

$$\bar{h}(\bar{x}_b, \bar{u}(\bar{x}_b), \bar{u}_{\bar{x}}(\bar{x}_b), \dots) = \bar{0} \quad (6.8)$$

We classify this problem as *elliptic*; examples include Laplace's, Poisson's, and Helmholtz's equations discussed in Section 2.7. The procedure for solving elliptic problems within the NUMOL is to add a vector of time derivatives to equation (6.7), and include an assumed initial condition vector, then numerically integrate the resulting system of ODEs in t to equilibrium where the time derivatives are essentially zero, the *method of false transients*. This approach was demonstrated in Section 2.7. As a special case, if \bar{x}_b is one-dimensional (has one component), then this procedure gives a solution to a system of *boundary-value ODEs*.

(2) *Parabolic PDEs.* If each PDE in equation (6.4) has only a first-order derivative in t and a second-order derivative in \bar{x}

$$\bar{u}_t = \bar{f}(\bar{x}, t, \bar{u}, \bar{u}_{\bar{x}\bar{x}}) \quad (6.9)$$

$$\bar{u}(\bar{x}, t_0) = \bar{g}(\bar{x}) \quad (6.10)$$

$$\bar{h}(\bar{x}_b, t, \bar{u}(\bar{x}_b, t), \bar{u}_{\bar{x}}(\bar{x}_b, t)) = \bar{0} \quad (6.11)$$

we classify this problem as *parabolic*; examples include Fourier's second law in Cartesian and cylindrical coordinates discussed in Sections 1.2 and 2.4. The procedure for solving parabolic problems within the NUMOL is straightforward; centered approximations (e.g., the five-point approximations in DSS004) should be used for the spatial derivative $\bar{u}_{\bar{x}\bar{x}}$ since this type of problem does not have a dominant direction.

(3) *First-Order Hyperbolic PDE's.* If each PDE in equation (6.4) has only a first-order derivative in t and a first-order derivative in \bar{x} , then

$$\bar{u}_t = \bar{f}(\bar{x}, t, \bar{u}, \bar{u}_{\bar{x}}) \quad (6.12)$$

$$\bar{u}(\bar{x}, t_0) = \bar{g}(\bar{x}) \quad (6.13)$$

$$\bar{h}(\bar{x}_b, t, \bar{u}(\bar{x}_b, t)) = \bar{0} \quad (6.14)$$

We classify this problem as *first-order hyperbolic*; examples include the advection equation discussed in Section 3.5 and the system of three PDEs discussed in Section 5.3. The procedure for solving first-order hyperbolic problems within the NUMOL is, in principle, straightforward; upwind approximations (e.g., the five-point biased upwind approximations in DSS020) should be used for the spatial derivative \bar{u}_x since this type of problem has a dominant direction. In practice, numerical difficulties can occur because first-order hyperbolic PDEs can propagate discontinuities as discussed in Section 3.5.

(4) *Second-Order Hyperbolic PDEs.* If each PDE in equation (6.4) has only a first-order derivative in t and a second-order derivative in \bar{x} , but the derivatives in t are related so that when two PDEs are combined, a second-order derivative in t results:

$$u_{1t} = u_2$$

$$u_{2t} = f_2(\bar{x}, t, u_1, u_2, \dots, u_n, u_{1\bar{xx}}, u_{2\bar{xx}}, \dots, u_{n\bar{xx}}) \quad (6.15)$$

$$u_1(\bar{x}, t_0) = g_1(\bar{x}), \quad u_2(\bar{x}, t_0) = g_2(\bar{x}), \dots \quad (6.16)$$

$$h_1(\bar{x}_b, t), u_1(\bar{x}_b, t), u_2(\bar{x}_b, t), \dots, u_n(\bar{x}_b, t), u_{1x}(\bar{x}_b, t),$$

$$u_{2\bar{x}}(\bar{x}_b, t), \dots, u_{n\bar{x}}(\bar{x}_b, t), \dots) = 0$$

$$h_2(\bar{x}_b, t), u_1(\bar{x}_b, t), u_2(\bar{x}_b, t), \dots, u_n(\bar{x}_b, t), u_{1x}(\bar{x}_b, t),$$

$$u_{2\bar{x}}(\bar{x}_b, t), \dots, u_{n\bar{x}}(\bar{x}_b, t), \dots) = 0 \quad (6.17)$$

we classify this problem as *second-order hyperbolic*; the wave equation discussed in Section 2.6 is an example. The procedure for solving second-order hyperbolic problems within the NUMOL is, in principle, straightforward; centered approximations (e.g., the five point approximations in DSS004) can be used for the spatial derivative $\bar{u}_{\bar{x}\bar{x}}$. In practice, numerical difficulties can occur because second-order hyperbolic PDEs can propagate discontinuities if the initial conditions are not smooth, or if the initial and boundary conditions are not consistent.

(5) *Hyperbolic Parabolic PDEs.* If each PDE in equation (6.4) has only a first-order derivative in t and first- and second-order derivatives in \bar{x} , we classify this problem as *hyperbolic parabolic*; Burgers' equation, which will be discussed subsequently, is an example. The procedure for solving hyperbolic parabolic PDEs, which are also termed *convective-diffusion equations*, is, in principle, straightforward. Upwind approximations should be used for the first-order derivatives in \bar{x} (the five-point biased upwind approximations in DSS020), and centered approximations should be used for the second-order derivatives in \bar{x} (the five-point centered approximations in DSS004). In practice, hyperbolic parabolic PDEs can propagate discontinuities, as we shall observe in discussing Burgers' equation.

The preceding geometric classification of PDEs is admittedly rather loosely defined and also departs from the traditional geometric classification based on a single linear PDE. However, we have found that the preceding classification is useful in selecting particular procedures and approximations for use in a NUMOL code. Of course, many other combinations of PDE structures and properties can be considered, and a spectrum of approximations can also be considered for each. Thus, the NUMOL is an open-ended approach to the numerical integration of a broad spectrum of PDEs, and its utility is really limited only by the experience and imagination of the analyst.

We now proceed to the NUMOL solution of Burgers' equation, a classical hyperbolic parabolic PDE that is widely used as a standard test problem for PDE codes.

6.2 Burgers' Equation in One Dimension

Burgers' equation is ideally suited as a test problem for the following reasons:

- (1) It is nonlinear, yet has a known exact solution that is easily evaluated. All three types of boundary conditions—Dirichlet, Neumann, and the third type—can be used.
- (2) A parameter can be selected to change the equation from predominantly hyperbolic (a relatively difficult problem as discussed previously) to predominantly parabolic (a relatively easy problem).

In one dimension, Burgers' equation is

$$u_t = -uu_x + \mu u_{xx} \quad (6.18)$$

Note the nonlinear hyperbolic (convective) term $-uu_x$ and the linear

parabolic (diffusion) term μu_{xx} . For small μ , equation (6.18) is predominantly hyperbolic, while for large μ , it is predominantly parabolic.

Equation (6.18) has a solution [Byrne and Hindmarsh (6.1)]

$$u(x, t) = \frac{1}{1 + e^{x/(2\mu) - t/(4\mu)}} \quad (6.19)$$

We can then use any initial and boundary conditions for equation (6.18) that are consistent with equation (6.19).

Subroutines INITAL, DERV, PRINT, and data are listed in Program 6.1 for the NUMOL integration of equation (6.18) using RKF45 for the ODE integration

Program 6.1 Subroutines INITAL, DERV, PRINT, and Data for Equation (6.18)

```

SUBROUTINE INITAL
C...
C... ONE DIMENSIONAL BURGERS' EQUATION WITH DIRICHLET AND NEUMANN
C... BOUNDARY CONDITIONS, ILLUSTRATING THE NUMERICAL METHOD OF LINES
C... INTEGRATION OF ONE DIMENSIONAL ELLIPTIC, HYPERBOLIC AND PARABOLIC
C... PARTIAL DIFFERENTIAL EQUATIONS (PDES) - EXPLICIT TIME INTEGRATION
C...
C... THIS PROBLEM ILLUSTRATES THE METHOD OF LINES PROGRAMMING OF A
C... ONE DIMENSIONAL HYPERBOLIC PARABOLIC (CONVECTIVE DIFFUSION)
C... PARTIAL DIFFERENTIAL EQUATION (PDE), THE WELL KNOWN BURGERS'
C... EQUATION
C...
C...     U   = -U*U  + VIS*U          (1)
C...     T       X      XX
C...
C... WHERE
C...
C...     U      DEPENDENT VARIABLE TO BE COMPUTED
C...
C...     T      INITIAL VALUE INDEPENDENT VARIABLE (TYPICALLY TIME)
C...
C...     X      BOUNDARY VALUE INDEPENDENT VARIABLE (TYPICALLY SPACE)
C...
C...     VIS    COEFFICIENT, TYPICALLY A VISCOSITY, WHICH CHANGES THE
C...             CHARACTER OF EQUATION (1)
C...
C...             VIS LARGE - EQUATION (1) IS STRONGLY PARABOLIC
C...
C...             VIS SMALL - EQUATION (1) IS STRONGLY HYPERBOLIC
C...
C... A LETTER SUBSCRIPT DENOTES A PARTIAL DERIVATIVE, E.G., U IS THE
C...             T
C... FIRST ORDER PARTIAL DERIVATIVE OF U WITH RESPECT TO T.
C...
C... EQUATION (1) IS A WELL ESTABLISHED TEST PROBLEM FOR PDE SOFTWARE
C... BECAUSE IT
C...
C...     (1) HAS AN EXACT SOLUTION (A), E.G.,
C...
C...             U(X,T) = 1/(1 + EXP(X/(2*VIS) - T/(4*VIS)))      (2)

```

continues

C...
C... (A) BENTON, E. R., AND G. W. PLATZMAN, A TABLE OF
C... SOLUTIONS OF THE ONE DIMENSIONAL BURGERS EQUATION,
C... QUARTERLY OF APPLIED MATHEMATICS, JULY, 1972, PP
C... 195-212.
C...
C... (2) IS A STRINGENT TEST PROBLEM, PARTICULARLY FOR SMALL VIS,
C... FOR WHICH THE SOLUTION EXHIBITS STEEP, MOVING FRONTS.
C...
C... IN THE SUBSEQUENT PROGRAMMING OF EQUATION (1), THE INITIAL AND
C... BOUNDARY CONDITIONS HAVE BEEN CHOSEN TO BE CONSISTENT WITH
C... EQUATION (2). ALSO, VIS = 1.0, WHICH GIVES EQUATION (1) A
C... MIXED HYPERBOLIC PARABOLIC CHARACTER, RESULTS IN A VERY SMOOTH
C... SOLUTION FROM EQUATION (2), I.E., THIS CHOICE OF VIS WAS MADE
C... TO FACILITATE THE CALCULATION OF THE NUMERICAL SOLUTION IN ORDER
C... TO DEMONSTRATE VARIOUS COMPUTATIONAL AND PROGRAMMING PROCEDURES
C... WITH A MODEST COMPUTING EFFORT.
C...
C... EQUATION (1) IS INTEGRATED BY THE NUMERICAL METHOD OF LINES IN
C... THE SUBSEQUENT PROGRAMMING FOR TWO TYPES OF BOUNDARY CONDITIONS
C... AT X = 0 AND X = XL (IN THE SUBSEQUENT DISCUSSION, XL HAS THE
C... NUMERICAL VALUE XL = 1)
C...
C... (1) DIRICHLET BOUNDARY CONDITIONS IN WHICH U(0,T) AND
C... U(XL,T) ARE SPECIFIED
C...
C... (2) NEUMANN BOUNDARY CONDITIONS IN WHICH U (0,T) AND
C... U (XL,T) ARE SPECIFIED
C... X
C...
C... ALSO, TWO APPROACHES TO THE CALCULATION OF THE DERIVATIVE U
C... XX
C... (WHICH WILL SUBSEQUENTLY BE DESIGNATED WITH THE IN LINE NOTATION
C... UXX) IN EQUATION (1) ARE USED
C...
C... (1) STAGEWISE DIFFERENTIATION, I.E., UX IS CALCULATED FROM U
C... AND THEN UXX IS CALCULATED FROM UX
C...
C... (2) DIRECT DIFFERENTIATION, I.E., UXX IS CALCULATED DIRECTLY
C... FROM U
C...
C... THUS, THERE ARE A TOTAL OF FOUR COMBINATIONS (2 TYPES OF BOUNDARY
C... CONDITIONS X 2 TYPES OF UXX CALCULATIONS) IN SUBROUTINES DERV1,
C... DERV2, DERV3 AND DERV4. COMMENTS ARE GIVEN IN EACH OF THESE
C... SUBROUTINES EXPLAINING THE PROGRAMMING IN DETAIL.
C...
C... SINCE THE EXACT SOLUTION, EQUATION (2), IS AVAILABLE, THE ERROR
C... IN THE NUMERICAL SOLUTION CAN BE COMPUTED. IN PARTICULAR, THE
C... ERROR IN U(X=XL/2,T=1.0) IS COMPUTED FOR 11, 21, 31, 41 AND 51
C... GRID POINTS IN X IN THE NUMERICAL METHOD OF LINES SOLUTION. THIS
C... ERROR IS THEN PLOTTED VS THE GRID SPACING LOGARITHMICALLY AT THE
C... END OF THE 20TH RUN (4 COMBINATIONS MENTIONED PREVIOUSLY X 5
C... NUMBERS OF GRID POINTS FOR EACH COMBINATION). THESE CALCULATIONS,
C... AND THE ASSOCIATED PLOTTING, ARE EXPLAINED IN THE COMMENTS IN
C... SUBROUTINE PRINT.
C...
C... THE EXACT SOLUTION IN THIS CASE IS (WITH X = XL/2 = 0.5, T = 1)
C...
C...
$$\begin{aligned} U(0.5, 1.0) &= 1/(1 + \exp(0.5/(2*VIS) - 1/(4*VIS))) \\ &= 1/(1 + \exp(0)) = 1/2 = 0.5 \end{aligned}$$

continues

C...
C... NOTE THAT THIS RESULT IS INDEPENDENT OF VIS. THUS, VIS CAN BE
C... DECREASED TO MAKE THE NUMERICAL SOLUTION MORE DIFFICULT TO
C... COMPUTE, I.E., THE PROBLEM IS MORE HYPERBOLIC AND LESS PARABOLIC,
C... YET THE NUMERICAL SOLUTION SHOULD REMAIN CLOSE TO 0.5 IF REASON-
C... ABLE ACCURACY IS TO BE MAINTAINED.

C...
C... IN ADDITION TO THE COMMENTS IN THE FOLLOWING CODING, THE READER
C... CAN ALSO REFER TO THE COMMENTS IN THE SPATIAL DIFFERENTIATION
C... ROUTINES, DSS004, DSS020 AND DSS044. IN PARTICULAR, CENTERED
C... APPROXIMATIONS IN SUBROUTINES DSS004 AND DSS044 ARE USED TO
C... CALCULATE UXX IN EQUATION (1), I.E., CENTERED APPROXIMATIONS
C... SHOULD BE USED IN PARABOLIC PROBLEMS, AND NONCENTERED (FIVE
C... POINT BIASED UPWIND) APPROXIMATIONS IN SUBROUTINE DSS020 ARE
C... USED TO CALCULATE UX IN EQUATION (1), I.E., NONCENTERED
C... APPROXIMATIONS SHOULD BE USED IN HYPERBOLIC PROBLEMS. ALL
C... OF THESE APPROXIMATIONS ARE DERIVED FROM THE TAYLOR SERIES
C... AS EXPLAINED IN THE THREE SUBROUTINES.

C...
C... TWO SPECIAL CASES OF THIS PROBLEM CAN BE CONSIDERED

C... (1) VIS = 0

C... IN THIS CASE, EQUATION (1) REDUCES TO

$$\frac{U}{T} - \frac{U \cdot U}{X} = -\frac{(U^{**2})}{X} / 2 \quad (3)$$

C...
C... EQUATION (3) HAS THE ANALYTICAL SOLUTION

$$U(X, T) = X / (1 + T) \quad (4)$$

C...
C... THE USE OF EQUATIONS (3) AND (4) IN THE FOLLOWING PROGRAMMING
C... REQUIRES SOME MODIFICATION OF THE PROGRAMMING.

C... (2) BOUNDARY VALUE ORDINARY DIFFERENTIAL EQUATION (ODE)

C...
C... THE STEADY STATE FORM OF EQUATION (1) IS THE BOUNDARY VALUE
C... ORDINARY DIFFERENTIAL EQUATION

$$\frac{U}{XX} - \frac{U \cdot U}{X} = 0 \quad (5)$$

C...
C... (NOTE THAT $U_T = 0$ IN EQUATION (1)), WHICH CAN ALSO BE CONSIDER-
C...
C... AS A ONE DIMENSIONAL ELLIPTIC PDE (WITH ONE INDEPENDENT VARI-
C...
C... ABLE, X).

C...
C... EQUATION (5) CAN BE SOLVED SUBJECT TO TWO BOUNDARY CONDITIONS
C... BY ALLOWING THE SOLUTION OF EQUATION (1) TO PROCEED TO $U_T = 0$.

C...
C... (AN EXACT SOLUTION CAN ALSO BE EASILY DERIVED). IF THE BOUND-
C...
C... ARY CONDITIONS ARE SELECTED AS $U(0, INF) = U(XL, INF) = 1$, THE
C...
C... EXACT SOLUTION IS $U(X, INF) = 1$. THIS SOLUTION CAN BE DEMON-
C...
C... STRATED BY ALLOWING THE NUMERICAL SOLUTION TO EQUATION (1) TO
C...
C... PROCEED TO $T = INF$, I.E., $U_T = 0$.

C...
C... THIS DISCUSSION SUGGESTS A GENERAL METHOD FOR SOLVING BOUNDARY
C...
C... VALUE ODES, I.E., A PSEUDO TIME DERIVATIVE IS ADDED TO THE ODES
C...
C... AND THE RESULTING PDES ARE INTEGRATED TO STEADY STATE FOR WHICH
C...
C... THE TIME DERIVATIVES ESSENTIALLY GO TO ZERO. THIS APPROACH HAS

continues

C... BEEN USED EFFECTIVELY IN THE SOLUTION OF SOME RELATIVELY COMPLEX BOUNDARY VALUE ODES (WHICH, AGAIN, CAN ALSO BE CONSIDERED ONE DIMENSIONAL ELLIPTIC PDES). THIS APPROACH IS ALSO TERMED C... THE METHOD OF FALSE TRANSIENTS.

C... FINALLY, THE VERIFICATION OF EQUATION (2) AS A SOLUTION TO EQUATION (1) FOLLOWS:

C... LET EX = EXP(X/(2*VIS) - T/(4*VIS)). THEN EQUATION (2) CAN BE WRITTEN AS:

C... $U(X,T) = 1/(1 + EX)$

C... THE VARIOUS TERMS IN EQUATION (1) ARE:

C... $\frac{U}{T} = (-1/(1 + EX)^2) * EX * (-1/(4*VIS))$

C... $\frac{U*U}{X} = (1/(1 + EX)) * (-1/(1 + EX)^2) * EX * (1/(2*VIS))$

C... $\frac{-VIS*U}{XX} = -VIS * ((-1/(1 + EX)^2) * EX * (1/(2*VIS))^2$

C... $+ EX * (1/(2*VIS)^2 / (1 + EX)^3) * EX * (1/(2*VIS)))$

C... SUMMATION OF THESE TERMS GIVES:

C... $0 = (1/(1 + EX)^2) * EX * (1/(4*VIS)) * (1 + 1)$

C... $- (1/(1 + EX)^3) * EX * (1/(2*VIS)) * (1 + EX)$

C... $= 0$

C... WE NOW PROCEED TO THE NUMERICAL METHOD OF LINES SOLUTION OF EQUATION (1) SUBJECT TO DIRICHLET AND NEUMANN BOUNDARY CONDITIONS, AND AN INITIAL CONDITION, WHICH ARE CONSISTENT WITH THE EXACT SOLUTION, EQUATION (2).

C... COMMON

COMMON/T/	T,	NSTOP,	NORUN
1 /Y/	U(51)		
2 /F/	UT(51)		
3 /S/	UX(51),	UXX(51)	
4 /R/	VIS,	XL,	X(51)
5 /I/	N,	IP	

C... EQUATION PARAMETERS
VIS=1.0E+00
XL =1.0E+00

C... NUMBER OF GRID POINTS
DIRICHLET BOUNDARY CONDITIONS
IF((NORUN- 1)*(NORUN- 6).EQ.0)N=11
IF((NORUN- 2)*(NORUN- 7).EQ.0)N=21
IF((NORUN- 3)*(NORUN- 8).EQ.0)N=31
IF((NORUN- 4)*(NORUN- 9).EQ.0)N=41
IF((NORUN- 5)*(NORUN-10).EQ.0)N=51

C... NEUMANN BOUNDARY CONDITIONS
IF((NORUN-11)*(NORUN-16).EQ.0)N=11
IF((NORUN-12)*(NORUN-17).EQ.0)N=21
IF((NORUN-13)*(NORUN-18).EQ.0)N=31
IF((NORUN-14)*(NORUN-19).EQ.0)N=41
IF((NORUN-15)*(NORUN-20).EQ.0)N=51

continues

```

C...
C... INITIAL CONDITIONS (T = 0 IN EQUATION (2))
DO 1 I=1,N
  X(I)=XL*FLOAT(I-1)/FLOAT(N-1)
  U(I)=UA(X(I),T,VIS)
1 CONTINUE
C...
C... INITIAL DERIVATIVES
CALL DERV
IP=0
RETURN
END

SUBROUTINE DERV
COMMON/T/           T,      NSTOP,      NORUN
1     /Y/           U(51)
2     /F/           UT(51)
3     /S/           UX(51),    UXX(51)
4     /R/           VIS,      XL,      X(51)
5     /I/           N,        IP

C...
C... DIRICHLET BOUNDARY CONDITIONS, STAGEWISE DIFFERENTIATION
IF((NORUN.GE. 1).AND.(NORUN.LE. 5))CALL DERV1
C...
C... DIRICHLET BOUNDARY CONDITIONS, DIRECT DIFFERENTIATION
IF((NORUN.GE. 6).AND.(NORUN.LE.10))CALL DERV2
C...
C... NEUMANN BOUNDARY CONDITIONS, STAGEWISE DIFFERENTIATION
IF((NORUN.GE.11).AND.(NORUN.LE.15))CALL DERV3
C...
C... NEUMANN BOUNDARY CONDITIONS, DIRECT DIFFERENTIATION
IF((NORUN.GE.16).AND.(NORUN.LE.20))CALL DERV4
RETURN
END

SUBROUTINE DERV1
C...
C... DIRICHLET BOUNDARY CONDITIONS, STAGEWISE DIFFERENTIATION
C...
COMMON/T/           T,      NSTOP,      NORUN
1     /Y/           U(51)
2     /F/           UT(51)
3     /S/           UX(51),    UXX(51)
4     /R/           VIS,      XL,      X(51)
5     /I/           N,        IP

C...
C... X = 0 (X = 0 IN EQUATION (2))
U(1)=UA(X(1),T,VIS)
C...
C... X = XL (X = XL IN EQUATION (2))
U(N)=UA(X(N),T,VIS)
C...
C... SPATIAL DERIVATIVES
C...
UX IN UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS004(0.0E+00,XL,N,U,UX)
C...
UXX (BY FIVE-POINT CENTERED APPROXIMATIONS)
CALL DSS004(0.0E+00,XL,N,UX,UXX)
C...
UX IN U*UX (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS020(0.0E+00,XL,N,U,UX,1.0E+00)
C...
C... BURGERS' EQUATION

```

continues

```

C...
C...      X = 0
C...      UT(1)=0.0E+00
C...
C...      0 LT X LT XL
C...      NM1=N-1
C...      DO 1 I=2,NM1
C...      UT(I)=-U(I)*UX(I)+VIS*UXX(I)
1      CONTINUE
C...
C...      X = XL
C...      UT(N)=0.0E+00
C...      RETURN
C...      END

SUBROUTINE DERV2
C...
C...  DIRICHLET BOUNDARY CONDITIONS, DIRECT DIFFERENTIATION
C...
C...  COMMON/T/           T,      NSTOP,      NORUN
1      /Y/           U(51)
2      /F/           UT(51)
3      /S/           UX(51),    UXX(51)
4      /R/           VIS,     XL,      X(51)
5      /I/           N,       IP
C...
C...      X = 0 (X = 0 IN EQUATION (2))
C...      U(1)=UA(X(1),T,VIS)
C...
C...      X = XL (X = XL IN EQUATION (2))
C...      U(N)=UA(X(N),T,VIS)
C...
C...  SPATIAL DERIVATIVES
C...
C...  UXX (BY SECOND DERIVATIVE APPROXIMATIONS)
C...  CALL DSS044(0.0E+00,XL,N,U,UX,UXX,1,1)
C...
C...  UX IN U*UX (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
C...  CALL DSS020(0.0E+00,XL,N,U,UX,1.0E+00)
C...
C...  BURGERS' EQUATION
C...
C...      X = 0
C...      UT(1)=0.0E+00
C...
C...      0 LT X LT XL
C...      NM1=N-1
C...      DO 1 I=2,NM1
C...      UT(I)=-U(I)*UX(I)+VIS*UXX(I)
1      CONTINUE
C...
C...      X = XL
C...      UT(N)=0.0E+00
C...      RETURN
C...      END

SUBROUTINE DERV3
C...
C...  NEUMANN BOUNDARY CONDITIONS, STAGEWISE DIFFERENTIATION

```

continues

```

C...
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/           U(51)
2      /F/           UT(51)
3      /S/           UX(51),    UXX(51)
4      /R/           VIS,      XL,      X(51)
5      /I/           N,        IP

C...
C... SPATIAL DERIVATIVES
C...
C... UX IN UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS004(0.0E+00,XL,N,U,UX)
C...
C... X = 0 (X = 0 IN EQUATION (2))
UX(1)=DUA(X(1),T,VIS)
C...
C... X = XL (X = XL IN EQUATION (2))
UX(N)=DUA(X(N),T,VIS)
C...
C... UXX (BY FIVE-POINT CENTERED APPROXIMATIONS)
CALL DSS004(0.0E+00,XL,N,UX,UXX)
C...
C... UX IN U*UX (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS020(0.0E+00,XL,N,U,UX,1.0E+00)
C...
C... X = 0 (X = 0 IN EQUATION (2))
UX(1)=DUA(X(1),T,VIS)
C...
C... X = XL (X = XL IN EQUATION (2))
UX(N)=DUA(X(N),T,VIS)
C...
C... BURGERS' EQUATION
C...
C... O LE X LE XL
DO 1 I=1,N
UT(I)=-U(I)*UX(I)+VIS*UXX(I)
1 CONTINUE
RETURN
END

SUBROUTINE DERV4
C...
C... NEUMANN BOUNDARY CONDITIONS, DIRECT DIFFERENTIATION
C...
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/           U(51)
2      /F/           UT(51)
3      /S/           UX(51),    UXX(51)
4      /R/           VIS,      XL,      X(51)
5      /I/           N,        IP

C...
C... X = 0 (X = 0 IN EQUATION (2))
UX(1)=DUA(X(1),T,VIS)
C...
C... X = XL (X = XL IN EQUATION (2))
UX(N)=DUA(X(N),T,VIS)
C...
C... SPATIAL DERIVATIVES
C...
C... UXX (BY SECOND DERIVATIVE APPROXIMATIONS)
CALL DSS044(0.0E+00,XL,N,U,UX,UXX,2,2)

```

continues

```

C...
C...      UX IN U*UX (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
C...      CALL DSS020(0.0E+00,XL,N,U,UX,1.0E+00)
C...
C...      X = 0 (X = 0 IN EQUATION (2))
C...      UX(1)=DUA(X(1),T,VIS)
C...
C...      X = XL (X = XL IN EQUATION (2))
C...      UX(N)=DUA(X(N),T,VIS)
C...
C...      BURGERS' EQUATION
C...
C...      0 LE X LE XL
C...      DO 1 I=1,N
C...      UT(I)=-U(I)*UX(I)+VIS*UNX(I)
1     CONTINUE
      RETURN
      END

      FUNCTION UA(X,T,VIS)
C...
C...      FUNCTION UA IMPLEMENTS THE ANALYTICAL SOLUTION, EQUATION (2)
C...
C...      EX=EXP(X/(2.0E+00*VIS)-T/(4.0E+00*VIS))
C...      UA=1.0E+00/(1.0E+00+EX)
C...      RETURN
C...      END

      FUNCTION DUA(X,T,VIS)
C...
C...      FUNCTION DUA IMPLEMENTS THE DERIVATIVE OF THE ANALYTICAL SOLUTION,
C...      EQUATION (2), WITH RESPECT TO X
C...
C...      EX=EXP(X/(2.0E+00*VIS)-T/(4.0E+00*VIS))
C...      DUA=-1.0E+00*(1.0E+00+EX)**(-2)*EX*(1.0E+00/(2.0E+00*VIS))
C...      RETURN
C...      END

      SUBROUTINE PRINT(NI,NO)
      COMMON/T/           T,          NSTOP,        NORUN
      1     /Y/            U(51)
      2     /F/            UT(51)
      3     /S/            UX(51),    UNX(51)
      4     /R/            VIS,        XL,          X(51)
      5     /I/            N,          IP
C...
C...      DIMENSION THE ARRAYS FOR PLOTTING
      DIMENSION DXP(5), EP1(2,5), EP2(2,5)
C...
C...      PRINT A HEADING FOR THE NUMERICAL SOLUTION
      IP=IP+1
      IF(IP.EQ.1)WRITE(NO,1)
1     FORMAT(11X,'T',7X,'U(XL/2,T)',6X,'U(XL/2,T)',8X,'ERROR',/,
      1             19X,'(NUM)',6X,'(ANAL)')
C...
C...      COMPUTE THE ANALYTICAL SOLUTION AND THE ERROR IN THE NUMERICAL
C...      SOLUTION AT X = XL/2
      N2=N/2+1
      XL2=X(N2)
      UANAL=UA(XL2,T,VIS)
      ERROR=U(N2)-UANAL
C...
C...      PRINT THE NUMERICAL AND ANALYTICAL SOLUTIONS, AND THE ERROR
      WRITE(NO,2)T,U(N2),UANAL,ERROR

```

continues

```

2      FORMAT(F12.2,2F15.6,E16.4)
C...  STORE THE SOLUTION U(X=0.5,T=1.0)FOR PLOTTING
C...
C...  THE ERRORS FOR NORUN = 1 TO 5 (NPLOT = 1), AND NORUN = 6 TO
C...  10 (NPLOT = 2) ARE PLOTTED VS THE GRID SPACING ON THE SAME
C...  LOG-LOG PLOT (DIRICHLET BOUNDARY CONDITIONS)
C...  IF((T.GT.0.95E+00).AND.(T.LT.1.05E+00))THEN
C...
C...    NPLOT=1
C...    IF((NORUN.GE. 1).AND.(NORUN.LE. 5))THEN
C...      NPLOT=1
C...      DXP(NORUN)= ALOG10(XL/FLOAT(N-1))
C...      EP1(NPLOT,NORUN-0)=ALOG10(ABS(ERROR))
C...
C...    NPLOT=2
C...    ELSE IF((NORUN.GE. 6).AND.(NORUN.LE.10))THEN
C...      NPLOT=2
C...      EP1(NPLOT,NORUN-5)=ALOG10(ABS(ERROR))
C...    END IF
C...
C...  THE ERRORS FOR NORUN = 11 TO 15 (NPLOT = 1), AND NORUN = 16 TO
C...  20 (NPLOT = 2) ARE PLOTTED VS THE GRID SPACING ON THE SAME
C...  LOG-LOG PLOT (NEUMANN BOUNDARY CONDITIONS)
C...
C...    NPLOT = 1
C...    IF((NORUN.GE.11).AND.(NORUN.LE.15))THEN
C...      NPLOT=1
C...      EP2(NPLOT,NORUN-10)=ALOG10(ABS(ERROR))
C...
C...    NPLOT=2
C...    ELSE IF((NORUN.GE.16).AND.(NORUN.LE.20))THEN
C...      NPLOT=2
C...      EP2(NPLOT,NORUN-15)=ALOG10(ABS(ERROR))
C...    END IF
C...  END IF
C...
C...  PLOT THE ERROR VS GRID SPACING AT THE END OF FOUR SERIES OF FIVE
C...  RUNS
C...  IF((T.GT.1.95E+00).AND.(T.LT.2.05E+00).AND.(NORUN.EQ.20))THEN
C...
C...    DIRICHLET BC
C...    CALL TPLOTS(2,5,DXP,EP1)
C...
C...    PRINT A LABEL FOR THE PLOT
C...    WRITE(NO,11)
11    FORMAT(
     1 ' DIRICHLET BC, LOG(ERROR) VS LOG(DX) ')
C...
C...    NEUMANN BC
C...    CALL TPLOTS(2,5,DXP,EP2)
C...
C...    PRINT A LABEL FOR THE PLOT
C...    WRITE(NO,12)
12    FORMAT(
     1 ' NEUMANN BC, LOG(ERROR) VS LOG(DX) ')
END IF
RETURN
END

```

continues

ONE-D BURGERS' EQN, DIRICHLET BC, STAGEWISE DIFFERENTIATION, N = 11
 0. 2.0 0.2
 1110000 0.001

ONE-D BURGERS' EQN, DIRICHLET BC, STAGEWISE DIFFERENTIATION, N = 21
 0. 2.0 0.2
 2110000 0.001

ONE-D BURGERS' EQN, DIRICHLET BC, STAGEWISE DIFFERENTIATION, N = 31
 0. 2.0 0.2
 3110000 0.001

ONE-D BURGERS' EQN, DIRICHLET BC, STAGEWISE DIFFERENTIATION, N = 41
 0. 2.0 0.2
 4110000 0.001

ONE-D BURGERS' EQN, DIRICHLET BC, STAGEWISE DIFFERENTIATION, N = 51
 0. 2.0 0.2
 5110000 0.001

ONE-D BURGERS' EQN, DIRICHLET BC, DIRECT DIFFERENTIATION, N = 11
 0. 2.0 0.2
 1110000 0.001

ONE-D BURGERS' EQN, DIRICHLET BC, DIRECT DIFFERENTIATION, N = 21
 0. 2.0 0.2
 2110000 0.001

ONE-D BURGERS' EQN, DIRICHLET BC, DIRECT DIFFERENTIATION, N = 31
 0. 2.0 0.2
 3110000 0.001

ONE-D BURGERS' EQN, DIRICHLET BC, DIRECT DIFFERENTIATION, N = 41
 0. 2.0 0.2
 4110000 0.001

ONE-D BURGERS' EQN, DIRICHLET BC, DIRECT DIFFERENTIATION, N = 51
 0. 2.0 0.2
 5110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, STAGEWISE DIFFERENTIATION, N = 11
 0. 2.0 0.2
 1110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, STAGEWISE DIFFERENTIATION, N = 21
 0. 2.0 0.2
 2110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, STAGEWISE DIFFERENTIATION, N = 31
 0. 2.0 0.2
 3110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, STAGEWISE DIFFERENTIATION, N = 41
 0. 2.0 0.2
 4110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, STAGEWISE DIFFERENTIATION, N = 51
 0. 2.0 0.2
 5110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, DIRECT DIFFERENTIATION, N = 11
 0. 2.0 0.2
 1110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, DIRECT DIFFERENTIATION, N = 21
 0. 2.0 0.2
 2110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, DIRECT DIFFERENTIATION, N = 31
 0. 2.0 0.2
 3110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, DIRECT DIFFERENTIATION, N = 41
 0. 2.0 0.2
 4110000 0.001

ONE-D BURGERS' EQN, NEUMANN BC, DIRECT DIFFERENTIATION, N = 51
 0. 2.0 0.2
 5110000 0.001

END OF RUNS

Subroutines INITIAL, DERV, and PRINT contain comments essentially explaining the operation of the code. As usual, subroutine INITIAL sets the initial condition for equation (6.18), in this case, using the exact solution, equation (6.19) in function UA with $t = 0$, $0 \leq x \leq 1$ and $\mu = 1$; this value of μ gives equation (6.18) significant hyperbolic and parabolic characteristics.

Subroutine DERV then calls one of four subroutines, DERV1 to DERV4, in which boundary conditions are implemented using the exact solution, equation (6.19) in function UA (Dirichlet boundary conditions) and function DUA (Neumann boundary conditions), at $x = 0$ and $x = x_l = 1$. The nonlinear first-derivative term in equation (6.18), $-uu_x$, is computed by subroutine DSS020. The second-derivative term in equation (6.18), μu_{xx} , is calculated by stagewise differentiation (DSS004 called twice to compute u_{xx} from u) or directly (DSS044 called once to compute u_{xx} from x). The data file indicates that each of the four cases (Dirichlet and Neumann boundary conditions with staged and direct differentiation) is executed with $n = 11, 21, 31, 41$, and 51 grid points; thus $4 \times 5 = 20$ runs are programmed.

Subroutine PRINT prints the numerical and exact solutions, and the error (=numerical - exact) as a function of t at $x = 0.5$. Also, $\log_{10}(\text{abs(error)})$ is plotted against $\log_{10}(\Delta x)$ (Δx = grid spacing = $1/(N - 1)$,

Table 6.1 Numerical Output from Program 6.1

```
RUN NO 1 ONE-D BURGERS' EQN, DIRICHLET BC, STAGEWISE DIFFERENTIATION, N = 1
INITIAL T - 0.000E+00
FINAL T - .200E+01
PRINT T - .200E+00
NUMBER OF DIFFERENTIAL EQUATIONS - 11
MAXIMUM INTEGRATION ERROR - .100E-02
```

T	U(XL/2, T) (NUM)	U(XL/2, T) (ANAL)	ERROR
0.00	.437823	.437823	0.0000E+00
.20	.450167	.450166	.9771E-06
.40	.462463	.462570	-.1069E-03
.60	.475021	.475021	.5610E-06
.80	.487504	.487503	.1603E-05
1.00	.500001	.500000	.6995E-06
1.20	.512498	.512497	.4517E-06
1.40	.524980	.524979	.1262E-05
1.60	.537431	.537430	.8570E-06
1.80	.549833	.549834	-.1093E-05
2.00	.562177	.562177	.6191E-06

$N = 11, 21, 31, 41, 51$ at $t = 1$ to observe how the error varies with the grid spacing for both Dirichlet and Neumann boundary conditions.

Execution of this code with main program Program 1.3d (which calls RKF45) produces a substantial amount of output. Therefore, only the abbreviated output from the first run (first set of data) is listed in Table 6.1. Note in particular that $|\text{error}|$ is 10^{-4} or smaller (for all 20 runs). Remember, though, this is a relatively easy (smooth) problem because of the choice of $\mu = 1$. In a later section we shall see that Burgers' equation can be made increasingly difficult to integrate by decreasing μ and, in fact, can produce a discontinuity (shock) in the solution.

6.3 Burgers' Equation in Two Dimensions

Equations (6.18) and (6.19) can be readily extended to two dimensions:

$$u_t = -uu_x + \mu u_{xx} - uu_y + \mu u_{yy} \quad (6.20)$$

$$u(x, y, t) = \frac{1}{1 + e^{x/(2\mu) + y/(2\mu) - 2t/(4\mu)}} \quad (6.21)$$

Subroutines INITAL, DERV, PRINT, and data for the NUMOL solution of equation (6.20), with initial and boundary conditions from equation (6.21), are listed in Program 6.2.

Program 6.2 Subroutines INITAL, DERV, PRINT, and Data for Equation (6.20)

```

SUBROUTINE INITAL
C...
C...  TWO DIMENSIONAL BURGERS' EQUATION WITH DIRICHLET AND NEUMANN
C...  BOUNDARY CONDITIONS, ILLUSTRATING THE NUMERICAL METHOD OF LINES
C...  INTEGRATION OF TWO DIMENSIONAL ELLIPTIC, HYPERBOLIC AND PARABOLIC
C...  PARTIAL DIFFERENTIAL EQUATIONS (PDES)
C...
C...  THIS PROBLEM ILLUSTRATES THE METHOD OF LINES PROGRAMMING OF A
C...  TWO DIMENSIONAL HYPERBOLIC PARABOLIC (CONVECTIVE DIFFUSION)
C...  PARTIAL DIFFERENTIAL EQUATION (PDE), THE WELL KNOWN BURGERS'
C...  EQUATION
C...
C...      U  = -U*U  + VIS*U  - U*U  + VIS*U          (1)
C...      T      X        XX     Y      YY
C...
C... WHERE
C...
C...      U      DEPENDENT VARIABLE TO BE COMPUTED
C...      T      INITIAL VALUE INDEPENDENT VARIABLE (TYPICALLY TIME)
C...      X,Y   BOUNDARY VALUE INDEPENDENT VARIABLES (TYPICALLY SPACE)
C...      VIS    COEFFICIENT, TYPICALLY A VISCOSITY, WHICH CHANGES THE
C...              CHARACTER OF EQUATION (1)

```

continues

C... VIS LARGE - EQUATION (1) IS STRONGLY PARABOLIC
 C... VIS SMALL - EQUATION (1) IS STRONGLY HYPERBOLIC
 C... A LETTER SUBSCRIPT DENOTES A PARTIAL DERIVATIVE, E.G., U_T IS THE
 C... FIRST ORDER PARTIAL DERIVATIVE OF U WITH RESPECT TO T.
 C...
 C... EQUATION (1) IS A WELL ESTABLISHED TEST PROBLEM FOR PDE SOFTWARE
 C... BECAUSE IT
 C...
 C... (1) HAS AN EXACT SOLUTION, E.G.,
 C...
 C... $U(X,Y,T) =$
 C... $1/(1 + \exp(X/(2*VIS) + Y/(2*VIS) - 2*T/(4*VIS)))$ (2)
 C...
 C... (2) IS A STRINGENT TEST PROBLEM, PARTICULARLY FOR SMALL VIS,
 C... FOR WHICH THE SOLUTION EXHIBITS STEEP, MOVING FRONTS.
 C...
 C... IN THE SUBSEQUENT PROGRAMMING OF EQUATION (1), THE INITIAL AND
 C... BOUNDARY CONDITIONS HAVE BEEN CHOSEN TO BE CONSISTENT WITH
 C... EQUATION (2). ALSO, VIS = 1.0, WHICH GIVES EQUATION (1) A
 C... MIXED HYPERBOLIC PARABOLIC CHARACTER, RESULTS IN A VERY SMOOTH
 C... SOLUTION FROM EQUATION (2), I.E., THIS CHOICE OF VIS WAS MADE
 C... TO FACILITATE THE CALCULATION OF THE NUMERICAL SOLUTION IN ORDER
 C... TO DEMONSTRATE VARIOUS COMPUTATIONAL AND PROGRAMMING PROCEDURES
 C... WITH A MODEST COMPUTING EFFORT.
 C...
 C... EQUATION (1) IS INTEGRATED BY THE NUMERICAL METHOD OF LINES IN
 C... THE SUBSEQUENT PROGRAMMING FOR TWO TYPES OF BOUNDARY CONDITIONS
 C... AT X = 0 AND X = XL, Y = 0 AND Y = YL (IN THE SUBSEQUENT DISCUSSION
 C... XL AND YL HAVE THE NUMERICAL VALUES XL = YL = 1)
 C...
 C... (1) DIRICHLET BOUNDARY CONDITIONS IN WHICH $U(0,Y,T)$,
 C... $U(X,0,T)$, $U(XL,Y,T)$ AND $U(X,YL,T)$ ARE SPECIFIED
 C...
 C... (2) NEUMANN BOUNDARY CONDITIONS IN WHICH $U_X(0,Y,T)$,
 C... $U_X(XL,Y,T)$, $U_Y(X,0,T)$ AND $U_Y(X,YL,T)$ ARE SPECIFIED.
 C...
 C... IN THE SUBSEQUENT DISCUSSION, THE DERIVATIVES IN EQUATION (1)
 C... WILL BE DESIGNATED WITH IN LINE NOTATION, E.G., U_{XX} WILL BE
 C... DESIGNATED U_{XX} .
 C...
 C... SINCE THE EXACT SOLUTION, EQUATION (2), IS AVAILABLE, THE ERROR
 C... IN THE NUMERICAL SOLUTION CAN BE COMPUTED. IN PARTICULAR, THE
 C... ERROR IN $U(X=XL/2, Y=YL/2, T=1)$ IS COMPUTED FROM THE FOLLOWING
 C... APPLICATION OF EQUATION (2)
 C...
 C... $U(0.5,0.5,1) = 1/(1 + \exp(0.5/(2*VIS) + 0.5/(2*VIS) - 2*1/(4*VIS)))$
 C...
 C... $= 1/(1 + \exp(0)) = 0.5$
 C...
 C... NOTE THAT THIS RESULT IS INDEPENDENT OF VIS. THUS, VIS CAN BE
 C... DECREASED TO MAKE THE NUMERICAL SOLUTION MORE DIFFICULT TO
 C... COMPUTE, I.E., THE PROBLEM IS MORE HYPERBOLIC AND LESS PARABOLIC,
 C... YET THE NUMERICAL SOLUTION SHOULD REMAIN CLOSE TO 0.5 IF REASON-
 C... ABLE ACCURACY IS TO BE MAINTAINED.

continues

C... IN ADDITION TO THE COMMENTS IN THE FOLLOWING CODING, THE READER
C... CAN ALSO REFER TO THE COMMENTS IN THE SPATIAL DIFFERENTIATION
C... SUBROUTINE, DSS034. IN PARTICULAR, CENTERED APPROXIMATIONS IN
C... SUBROUTINE DSS034 ARE USED TO CALCULATE UXX AND UYY IN EQUATION
C... (1), I.E., CENTERED APPROXIMATIONS SHOULD BE USED IN PARABOLIC
C... PROBLEMS, AND NONCENTERED (FIVE POINT BIASED UPWIND) APPROXI-
C... MATIONS IN SUBROUTINE DSS034 ARE USED TO CALCULATE UX AND UY IN
C... EQUATION (1), I.E., NONCENTERED APPROXIMATIONS SHOULD BE USED
C... IN HYPERBOLIC PROBLEMS. ALL OF THESE APPROXIMATIONS ARE DERIVED
C... FROM THE TAYLOR SERIES AS EXPLAINED IN DSS034, AND SUBORDINATE
C... ROUTINES DSS004 AND DSS020.

C... TWO SPECIAL CASES OF THIS PROBLEM CAN BE CONSIDERED
C...

C... (1) VIS = 0

C... IN THIS CASE, EQUATION (1) REDUCES TO

$$\frac{U}{T} = - \frac{U*U}{X} - \frac{U*U}{Y} = - \frac{(U^{**2})}{X}/2 - \frac{(U^{**2})}{Y}/2 \quad (3)$$

C... EQUATION (3) HAS THE ANALYTICAL SOLUTION

$$U(X, Y, T) = (X + Y)/(1 + 2*T) \quad (4)$$

C... THE USE OF EQUATIONS (3) AND (4) IN THE FOLLOWING PROGRAMMING
C... REQUIRES SOME MODIFICATION OF THE PROGRAMMING.

C... (2) BOUNDARY VALUE PARTIAL DIFFERENTIAL EQUATION (PDE)

C... THE STEADY STATE FORM OF EQUATION (1) IS THE BOUNDARY VALUE
C... PARTIAL DIFFERENTIAL EQUATION

$$\frac{U}{XX} - \frac{U*U}{X} + \frac{U}{YY} - \frac{U*U}{Y} = 0 \quad (5)$$

C... (NOTE THAT $U = 0$ IN EQUATION (1)), WHICH CAN ALSO BE CONSIDER-
C... ED AS A TWO-DIMENSIONAL ELLIPTIC PDE (WITH INDEPENDENT VARI-
C... ABLES X AND Y). THUS, THE FOLLOWING PROGRAMMING ILLUSTRATES A
C... METHOD FOR INTEGRATING ELLIPTIC PDES AS WELL AS HYPERBOLIC
C... AND PARABOLIC PDES (BY ALLOWING THE SOLUTION OF EQUATION (1)
C... TO PROCEED TO $U = 0$, FOR WHICH EQUATION (2), WITH LARGE T,
C... T

C... REDUCES TO:

$$U(X, Y, INF) = 1$$

C... THIS DISCUSSION SUGGESTS A GENERAL METHOD FOR SOLVING BOUNDARY-
C... VALUE PDES, I.E., A PSEUDO TIME DERIVATIVE IS ADDED TO THE PDES
C... AND THE RESULTING PDES ARE INTEGRATED TO STEADY STATE FOR WHICH
C... THE TIME DERIVATIVES ESSENTIALLY GO TO ZERO. THIS APPROACH HAS
C... BEEN USED EFFECTIVELY IN THE SOLUTION OF SOME RELATIVELY COM-
C... PLEX BOUNDARY VALUE PDES. THIS APPROACH IS ALSO TERMED THE
C... METHOD OF FALSE TRANSIENTS.

C... FINALLY, THE VERIFICATION OF EQUATION (2) AS A SOLUTION TO EQUA-
C... TION (1) FOLLOWS:

C... LET EX = EXP(X/(2*VIS) + Y/(2*VIS) - 2*T/(4*VIS)). THEN EQUATION
C... (2) CAN BE WRITTEN AS:

continues

```

C...
C...      U(X,Y,T) = 1/(1 + EX)
C...
C... THE VARIOUS TERMS IN EQUATION (1) ARE:
C...
C...      U   = (-1/(1 + EX)**2)*EX*(-2/(4*VIS))
C...      T
C...
C...      U*X   = (1/(1 + EX))*(-1/(1 + EX)**2)*EX*(1/(2*VIS))
C...      X
C...
C...      -VIS*U  XX = -VIS*((-1/(1 + EX)**2)*EX*(1/(2*VIS))**2
C...
C...      + EX*(1/(2*VIS)*2/(1 + EX)**3)*EX*(1/(2*VIS)))
C...
C...      U*Y   = (1/(1 + EX))*(-1/(1 + EX)**2)*EX*(1/(2*VIS))
C...      Y
C...
C...      -VIS*U  YY = -VIS*((-1/(1 + EX)**2)*EX*(1/(2*VIS))**2
C...
C...      + EX*(1/(2*VIS)*2/(1 + EX)**3)*EX*(1/(2*VIS)))
C...
C... SUMMATION OF THESE TERMS GIVES:
C...
C...      0 = (1/(1 + EX)**2)*EX*(1/(4*VIS))*(2 + 1 + 1)
C...
C...      - (1/(1 + EX)**3)*EX*(1/(2*VIS))*(1 + 1)*(1 + EX)
C...
C...      = 0
C...
C... WE NOW PROCEED TO THE NUMERICAL METHOD OF LINES SOLUTION OF EQUA-
C... TION (1) SUBJECT TO DIRICHLET AND NEUMANN BOUNDARY CONDITIONS, AND
C... AN INITIAL CONDITION, WHICH ARE CONSISTENT WITH THE EXACT SOLU-
C... TION, EQUATION (2).
C...
C... COMMON
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/    U(11,11)
2      /F/    UT(11,11)
3      /S/    UX(11,11),UXX(11,11),
4          UY(11,11),UYY(11,11)
5      /R/    XL,        YL,
6          X(11),      Y(11),
7          VIS
8      /I/    NX,        NY,
9          IP
C...
C... EQUATION PARAMETERS
VIS=1.OE+00
XL=1.OE+00
YL=1.OE+00
C...
C... NUMBER OF GRID POINTS
NX=11
NY=11
C...
C... INITIAL CONDITIONS (T = 0 IN EQUATION (2))
DO 1 I=1,NX
X(I)=XL*FLOAT(I-1)/FLOAT(NX-1)
DO 1 J=1,NY
Y(J)=YL*FLOAT(J-1)/FLOAT(NY-1)
U(I,J)=UA(X(I),Y(J),T,VIS)
1 CONTINUE

```

continues

```

C...
C... INITIAL DERIVATIVES
CALL DERV
IP=0
RETURN
END

SUBROUTINE DERV
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/   U(11,11)
2      /F/   UT(11,11)
3      /S/   UX(11,11),UXX(11,11),
4          UY(11,11),UYY(11,11)
5      /R/   XL,       YL,
6          X(11),     Y(11),
7          VIS
8      /I/   NX,       NY,
9          IP

C...
C... DIRICHLET BOUNDARY CONDITIONS WITH (1) UXX AND UYY CALCULATED BY
C... CENTERED APPROXIMATIONS AND (2) UX AND UY CALCULATED BY BIASED
C... UPWIND APPROXIMATIONS
IF(NORUN.EQ.1)CALL DERV1
C...
C... DIRICHLET BOUNDARY CONDITIONS WITH UXX, UYY, UX AND UY CALCULATED
C... BY CENTERED APPROXIMATIONS
IF(NORUN.EQ.2)CALL DERV2
C...
C... NEUMANN BOUNDARY CONDITIONS WITH (1) UXX AND UYY CALCULATED BY
C... CENTERED APPROXIMATIONS AND (2) UX AND UY CALCULATED BY BIASED
C... UPWIND APPROXIMATIONS
IF(NORUN.EQ.3)CALL DERV3
C...
C... NEUMANN BOUNDARY CONDITIONS WITH UXX, UYY, UX AND UY CALCULATED
C... BY CENTERED APPROXIMATIONS
IF(NORUN.EQ.4)CALL DERV4
RETURN
END

SUBROUTINE DERV1
C...
C... DIRICHLET BOUNDARY CONDITIONS
C...
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/   U(11,11)
2      /F/   UT(11,11)
3      /S/   UX(11,11),UXX(11,11),
4          UY(11,11),UYY(11,11)
5      /R/   XL,       YL,
6          X(11),     Y(11),
7          VIS
8      /I/   NX,       NY,
9          IP

C...
C... X = 0, 0 LE Y LE YL (X = 0 IN EQUATION (2))
DO 1 J=1,NY
U(1,J)=UA(X(1),Y(J),T,VIS)
1 CONTINUE
C...
C... X = XL, 0 LE Y LE YL (X = XL IN EQUATION (2))
DO 2 J=1,NY
U(NX,J)=UA(X(NX),Y(J),T,VIS)
2 CONTINUE

```

continues

```

C...
C...      Y = 0, 0 LE X LE XL (Y = 0 IN EQUATION (2))
DO 3 I=1,NX
      U(I,1)=UA(X(I),Y(1),T,VIS)
3      CONTINUE
C...
C...      Y = YL, 0 LE X LE XL (Y = YL IN EQUATION (2))
DO 4 I=1,NX
      U(I,NY)=UA(X(I),Y(NY),T,VIS)
4      CONTINUE
C...
C...      SPATIAL DERIVATIVES
C...
C...      UX IN UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,XL,NX,NY,1,U,UX,0.E+00)
C...
C...      UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,XL,NX,NY,1,UX,UXX,0.E+00)
C...
C...      UX IN U*UX (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS034(0.E+00,XL,NX,NY,1,U,UX,1.0E+00)
C...
C...      UY IN UYY (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,YL,NX,NY,2,U,UY,0.E+00)
C...
C...      UYY (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,YL,NX,NY,2,UY,UYY,0.E+00)
C...
C...      UY IN U*UY (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS034(0.E+00,YL,NX,NY,2,U,UY,1.0E+00)
C...
C...      BURGERS' EQUATION
C...
      DO 10 I=1,NX
      DO 10 J=1,NY
      UT(I,J)=-U(I,J)*UX(I,J)+VIS*UXX(I,J)
1      -U(I,J)*UY(I,J)+VIS*UYY(I,J)
      10 CONTINUE
      RETURN
      END

      SUBROUTINE DERV2
C...
C...      DIRICHLET BOUNDARY CONDITIONS
C...
      COMMON/T/,          T,      NSTOP,      NORUN
1      /Y/,      U(11,11)
2      /F/,      UT(11,11)
3      /S/,      UX(11,11), UXX(11,11),
4      UY(11,11), UYY(11,11)
5      /R/,      XL,      YL,
6      X(11),      Y(11),
7      VIS
8      /I/,      NX,      NY,
9      IP

C...
C...      X = 0, 0 LE Y LE YL (X = 0 IN EQUATION (2))
DO 1 J=1,NY
      U(1,J)=UA(X(1),Y(J),T,VIS)
1      CONTINUE

```

continues

```

C...
C...      X = XL, 0 LE Y LE YL (X = XL IN EQUATION (2))
DO 2 J=1,NY
      U(NX,J)=UA(X(NX),Y(J),T,VIS)
2      CONTINUE
C...
C...      Y = 0, 0 LE X LE XL (Y = 0 IN EQUATION (2))
DO 3 I=1,NX
      U(I,1)=UA(X(I),Y(1),T,VIS)
3      CONTINUE
C...
C...      Y = YL, 0 LE X LE XL (Y = YL IN EQUATION (2))
DO 4 I=1,NX
      U(I,NY)=UA(X(I),Y(NY),T,VIS)
4      CONTINUE
C...
C...  SPATIAL DERIVATIVES
C...
C...      UX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,XL,NX,NY,1,U,UX,0.E+00)
C...
C...      UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,XL,NX,NY,1,UX,UXX,0.E+00)
C...
C...      UY (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,YL,NX,NY,2,U,UY,0.E+00)
C...
C...      UYY (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,YL,NX,NY,2,UY,UYY,0.E+00)
C...
C...  BURGERS' EQUATION
C...
      DO 10 I=1,NX
      DO 10 J=1,NY
      UT(I,J)=-U(I,J)*UX(I,J)+VIS*UXX(I,J)
      1          -U(I,J)*UY(I,J)+VIS*UYY(I,J)
10     CONTINUE
      RETURN
      END

      SUBROUTINE DERV3
C...
C...  NEUMANN BOUNDARY CONDITIONS
C...
      COMMON/T/,NSTOP,NORUN
      1      /Y/U(11,11)
      2      /F/UT(11,11)
      3      /S/UX(11,11),UXX(11,11),
      4          UY(11,11),UYY(11,11)
      5      /R/XL,YL,
      6          X(11),Y(11),
      7          VIS
      8      /I/NX,NY,
      9          IP

C...
C...  SPATIAL DERIVATIVES
C...
C...      UX IN UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,XL,NX,NY,1,U,UX,0.E+00)
C...
C...      X = 0, 0 LE Y LE YL (X = 0 IN EQUATION (2))
DO 1 J=1,NY
      UX(1,J)=DUA(X(1),Y(J),T,VIS)
1      CONTINUE

```

continues

```

C...
C... X = XL, 0 LE Y LE YL (X = XL IN EQUATION (2))
DO 2 J=1,NY
UX(NX,J)=DUA(X(NX),Y(J),T,VIS)
2 CONTINUE
C...
C... UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,XL,NX,NY,1,UX,UXX,0.E+00)
C...
C... UX IN U*UX (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS034(0.E+00,XL,NX,NY,1,U,UX,1.OE+00)
C...
C... X = 0, 0 LE Y LE YL (X = 0 IN EQUATION (2))
DO 3 J=1,NY
UX(1,J)=DUA(X(1),Y(J),T,VIS)
3 CONTINUE
C...
C... X = XL, 0 LE Y LE YL (X = XL IN EQUATION (2))
DO 4 J=1,NY
UX(NX,J)=DUA(X(NX),Y(J),T,VIS)
4 CONTINUE
C...
C... UY IN UYY (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,YL,NX,NY,2,U,UY,0.E+00)
C...
C... Y = 0, 0 LE X LE XL (Y = 0 IN EQUATION (2))
DO 5 I=1,NX
UY(I,1)=DUA(X(I),Y(1),T,VIS)
5 CONTINUE
C...
C... Y = YL, 0 LE X LE XL (Y = YL IN EQUATION (2))
DO 6 I=1,NX
UY(I,NY)=DUA(X(I),Y(NY),T,VIS)
6 CONTINUE
C...
C... UYY (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS034(0.E+00,YL,NX,NY,2,UY,UYY,0.E+00)
C...
C... UY IN U*UY (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS034(0.E+00,YL,NX,NY,2,U,UY,1.OE+00)
C...
C... Y = 0, 0 LE X LE XL (Y = 0 IN EQUATION (2))
DO 7 I=1,NX
UY(I,1)=DUA(X(I),Y(1),T,VIS)
7 CONTINUE
C...
C... Y = YL, 0 LE X LE XL (Y = YL IN EQUATION (2))
DO 8 I=1,NX
UY(I,NY)=DUA(X(I),Y(NY),T,VIS)
8 CONTINUE
C...
C... BURGERS' EQUATION
C...
      DO 10 I=1,NX
      DO 10 J=1,NY
      UT(I,J)=-U(I,J)*UX(I,J)+VIS*UXX(I,J)
10          -U(I,J)*UY(I,J)+VIS*UYY(I,J)
      1 CONTINUE
      RETURN
      END

```

continues

```

SUBROUTINE DERV4
C... NEUMANN BOUNDARY CONDITIONS
C...
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/   U(11,11)
2      /F/   UT(11,11)
3      /S/   UX(11,11),UXX(11,11),
4          UY(11,11),UYY(11,11)
5      /R/   XL,       YL,
6          X(11),     Y(11),
7          VIS
8      /I/   NX,       NY,
9          IP

C... SPATIAL DERIVATIVES
C...
C... UX BY FIVE POINT CENTERED APPROXIMATIONS
CALL DSS034(O.E+00,XL,NX,NY,1,U,UX,O.E+00)
C...
C... X = 0, 0 LE Y LE YL (X = 0 IN EQUATION (2))
DO 1 J=1,NY
UX(1,J)=DUA(X(1),Y(J),T,VIS)
1 CONTINUE
C...
C... X = XL, 0 LE Y LE YL (X = XL IN EQUATION (2))
DO 2 J=1,NY
UX(NX,J)=DUA(X(NX),Y(J),T,VIS)
2 CONTINUE
C...
C... UXX BY FIVE POINT CENTERED APPROXIMATIONS
CALL DSS034(O.E+00,XL,NX,NY,1,UX,UXX,O.E+00)
C...
C... UY BY FIVE POINT CENTERED APPROXIMATIONS
CALL DSS034(O.E+00,YL,NX,NY,2,U,UY,O.E+00)
C...
C... Y = 0, 0 LE X LE XL (Y = 0 IN EQUATION (2))
DO 5 I=1,NX
UY(I,1)=DUA(X(I),Y(1),T,VIS)
5 CONTINUE
C...
C... Y = YL, 0 LE X LE XL (Y = YL IN EQUATION (2))
DO 6 I=1,NX
UY(I,NY)=DUA(X(I),Y(NY),T,VIS)
6 CONTINUE
C...
C... UYY BY FIVE POINT CENTERED APPROXIMATIONS
CALL DSS034(O.E+00,YL,NX,NY,2,UY,UYY,O.E+00)
C...
C... BURGERS' EQUATION
C...
DO 10 I=1,NX
DO 10 J=1,NY
UT(I,J)=-U(I,J)*UX(I,J)+VIS*UXX(I,J)
1          -U(I,J)*UY(I,J)+VIS*UYY(I,J)
10 CONTINUE
RETURN
END

FUNCTION UA(X,Y,T,VIS)
C... FUNCTION UA IMPLEMENTS THE ANALYTICAL SOLUTION, EQUATION (2)

```

continues

```

C...
EX=EXP(X/(2.0E+00*VIS)+Y/(2.0E+00*VIS)-T/(2.0E+00*VIS))
UA=1.0E+00/(1.0E+00+EX)
RETURN
END

FUNCTION DUA(X,Y,T,VIS)
C...
C... FUNCTION DUA IMPLEMENTS THE DERIVATIVE OF THE ANALYTICAL SOLUTION,
C... EQUATION (2), WITH RESPECT TO X AND Y
C...
EX=EXP(X/(2.0E+00*VIS)+Y/(2.0E+00*VIS)-T/(2.0E+00*VIS))
DUA=-1.0E+00*(1.0E+00+EX)**(-2)*EX*(1.0E+00/(2.0E+00*VIS))
RETURN
END

SUBROUTINE PRINT(NI,NO)
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/   U(11,11)
2      /F/   UT(11,11)
3      /S/   UX(11,11),UXX(11,11),
4          UY(11,11),UYY(11,11)
5      /R/   XL,      YL,
6          X(11),     Y(11),
7          VIS
8      /I/   NX,      NY,
9          IP

C...
C... PRINT A HEADING FOR THE NUMERICAL SOLUTION
IP=IP+1
IF(IP.EQ.1)WRITE(NO,1)
1 FORMAT(11X,'T',5X,'U(XL/2,YL/2,T)',,
1           3X,'U(XL/2,YL/2,T)',5X,'ERROR',/,
2           20X,'(NUM)',7X,'(ANAL)')
C...
C... COMPUTE THE ANALYTICAL SOLUTION AND THE ERROR IN THE NUMERICAL
C... SOLUTION AT X = XL/2, Y = YL/2
NX2=NX/2+1
NY2=NY/2+1
XL2=X(NX2)
YL2=Y(NY2)
UANAL=UA(XL2,YL2,T,VIS)
ERROR=U(NX2,NY2)-UANAL
C...
C... PRINT THE NUMERICAL AND ANALYTICAL SOLUTIONS, AND THE ERROR
WRITE(NO,2)T,U(NX2,NY2),UANAL,ERROR
2 FORMAT(F12.2,2F16.6,E16.4)
RETURN
END

TWO-D BURGERS' EQUATION, DIRICHLET BC
0.          2.0        0.2
121 1000            0.001
TWO-D BURGERS' EQUATION, DIRICHLET BC
0.          2.0        0.2
121 1000            0.001
TWO-D BURGERS' EQUATION, NEUMANN BC
0.          2.0        0.2
121 1000            0.001
TWO-D BURGERS' EQUATION, NEUMANN BC
0.          2.0        0.2
121 1000            0.001
END OF RUNS

```

Program 6.2 closely parallels Program 6.1. However, to calculate the derivatives in equation (6.20) over the two-dimensional domain in x and y , subroutine DSS034 is used in place of DSS004 and DSS020. The arguments of DSS034 are illustrated by the following call:

```
CALL DSS034(X1,X2,NX,NY,ND,U,UX,V)
```

- X1 the lower boundary value of the independent variable appearing in the derivative to be calculated (input)
- X2 the upper boundary value of the independent variable appearing in the derivative to be calculated (input)
- NX number of spatial grid points in the x direction (input)
- NY number of spatial grid points in the y direction (input)
- ND number of the independent variable appearing in the derivative to be calculated; ND = 1 for x , ND = 2 for y (input)
- U two-dimensional array, of size U(NX, NY), containing the dependent variable to be differentiated (input)
- UX two-dimensional array, of size UX(NX, NY), containing the computed first derivative of the dependent variable u with respect to the independent variable with number ND (output)
- V direction indicator for the five-point biased upwind approximations in DSS034; V > 1 for flow in the positive direction; V = 0 for no flow (diffusion only); V < 1 for flow in the negative direction (input)

Documentation comments at the beginning of DSS034 also describe the arguments in detail.

DSS034 actually executes a series of calls to DSS004 (when the last argument is zero) for parabolic (diffusion) derivatives such as μu_{xx} and μu_{yy} in equation (6.20) and executes a series of calls to DSS020 (when the last argument is not zero) for hyperbolic (convective) derivatives such as $-uu_x$ and $-uu_y$. An example of how two-dimensional problems can be accommodated through a series of calls to one-dimensional routines was discussed in Section 2.7.

The data of Program 6.2 are for four runs in which subroutines DERV1 to DERV4 are executed. However, because of space limitations, only the abbreviated output from the first run is listed in Table 6.2. The 11×11 -point grid in x and y (121 ODEs) gives accuracy better than 10^{-7} (absolute). Also, because $\mu = 1$ gives a relatively smooth solution, the hyperbolic terms $-uu_x$ and $-uu_y$ can be calculated by centered differences (in DERV2 for Dirichlet boundary conditions and in DERV4 for Neumann boundary conditions); if equation (6.20) were strongly hyperbolic ($\mu \ll 1$), only five-point biased upwind differences should be used.

Table 6.2 Numerical Output from Program 6.2

RUN NO. - 1 TWO-D BURGERS' EQUATION, DIRICHLET BC

INITIAL T - 0.000E+00

FINAL T - .200E+01

PRINT T - .200E+00

NUMBER OF DIFFERENTIAL EQUATIONS - 121

MAXIMUM INTEGRATION ERROR - .100E-02

T	U(XL/2,YL/2,T) (NUM)	U(XL/2,YL/2,T) (ANAL)	ERROR
0.00	.377541	.377541	0.0000E+00
.20	.401312	.401312	-.3960E-08
.40	.425557	.425557	.2525E-08
.60	.450166	.450166	.4126E-08
.80	.475021	.475021	-.2732E-07
1.00	.500000	.500000	.5859E-08
1.20	.524979	.524979	.8944E-08
1.40	.549834	.549834	-.1420E-07
1.60	.574443	.574443	.9089E-08
1.80	.598688	.598688	.1537E-07
2.00	.622459	.622459	-.4088E-10

6.4 Burgers' Equation in Three Dimensions

Equations (6.20) and (6.21) can be readily extended to three dimensions:

$$u_t = -uu_x + \mu u_{xx} - uu_y + \mu u_{yy} - uu_z + \mu u_{zz} \quad (6.22)$$

$$u(x, y, z, t) = \frac{1}{1 + e^{x/(2\mu) + y/(2\mu) + z/(2\mu) - 3t/(4\mu)}} \quad (6.23)$$

Subroutines INITIAL, DERV, PRINT, and data for the NUMOL solution of equation (6.22), with initial and boundary conditions from equation (6.23), are listed in Program 6.3. Program 6.3 closely parallels Program 6.2 for the two-dimensional Burgers equation. The only essential

Program 6.3 Subroutines INITIAL, DERV, PRINT, and Data for Equation (6.22)

SUBROUTINE INITIAL

```
C...
C... THREE DIMENSIONAL BURGERS' EQUATION WITH DIRICHLET AND NEUMANN
C... BOUNDARY CONDITIONS, ILLUSTRATING THE NUMERICAL METHOD OF LINES
C... INTEGRATION OF THREE DIMENSIONAL ELLIPTIC, HYPERBOLIC AND PARA-
C... BOLIC PARTIAL DIFFERENTIAL EQUATIONS (PDES)
C...
C... THIS PROBLEM ILLUSTRATES THE METHOD OF LINES PROGRAMMING OF A
C... THREE DIMENSIONAL HYPERBOLIC PARABOLIC (CONVECTIVE DIFFUSION)
C... PARTIAL DIFFERENTIAL EQUATION (PDE), THE WELL KNOWN BURGERS'
C... EQUATION
```

continues

```

C...
C...      U = -U*U + VIS*U - U*U + VIS*U - U*U + VIS*U   (1)
C...      T     X     XX    Y     YY    Z     ZZ
C...
C... WHERE
C...
C...     U      DEPENDENT VARIABLE TO BE COMPUTED
C...     T      INITIAL VALUE INDEPENDENT VARIABLE (TYPICALLY TIME)
C...
C...     X,Y,Z  BOUNDARY VALUE INDEPENDENT VARIABLES (TYPICALLY SPACE)
C...
C...     VIS    COEFFICIENT, TYPICALLY A VISCOSITY, WHICH CHANGES THE
C...             CHARACTER OF EQUATION (1)
C...
C...             VIS LARGE - EQUATION (1) IS STRONGLY PARABOLIC
C...
C...             VIS SMALL - EQUATION (1) IS STRONGLY HYPERBOLIC
C...
C... A LETTER SUBSCRIPT DENOTES A PARTIAL DERIVATIVE, E.G., U IS THE
C...             T
C... FIRST ORDER PARTIAL DERIVATIVE OF U WITH RESPECT TO T.
C...
C... EQUATION (1) IS A WELL ESTABLISHED TEST PROBLEM FOR PDE SOFTWARE
C... BECAUSE IT
C...
C... (1) HAS AN EXACT SOLUTION, E.G.,
C...
C...     U(X,Y,Z,T) =
C...
C...     1/(1 + EXP(X/(2*VIS) + Y/(2*VIS) + Z/(2*VIS))           (2)
C...
C...             - 3*T/(4*VIS)))
C...
C... (2) IS A STRINGENT TEST PROBLEM, PARTICULARLY FOR SMALL VIS,
C... FOR WHICH THE SOLUTION EXHIBITS STEEP, MOVING FRONTS.
C...
C... IN THE SUBSEQUENT PROGRAMMING OF EQUATION (1), THE INITIAL AND
C... BOUNDARY CONDITIONS HAVE BEEN CHOSEN TO BE CONSISTENT WITH
C... EQUATION (2). ALSO, VIS = 1.0, WHICH GIVES EQUATION (1) A
C... MIXED HYPERBOLIC PARABOLIC CHARACTER, RESULTS IN A VERY SMOOTH
C... SOLUTION FROM EQUATION (2), I.E., THIS CHOICE OF VIS WAS MADE
C... TO FACILITATE THE CALCULATION OF THE NUMERICAL SOLUTION IN ORDER
C... TO DEMONSTRATE VARIOUS COMPUTATIONAL AND PROGRAMMING PROCEDURES
C... WITH A MODEST COMPUTING EFFORT.
C...
C... EQUATION (1) IS INTEGRATED BY THE NUMERICAL METHOD OF LINES IN
C... THE SUBSEQUENT PROGRAMMING FOR TWO TYPES OF BOUNDARY CONDITIONS
C... AT (1) X = AND X = XL, (2) Y = 0 AND Y = YL AND (3) Z = 0 AND
C... Z = ZL (IN THE SUBSEQUENT DISCUSSION, XL, YL AND ZL HAVE THE
C... NUMERICAL VALUES XL = YL = ZL = 1)
C...
C... (1) DIRICHLET BOUNDARY CONDITIONS IN WHICH U(0,Y,Z,T),
C...     U(XL,Y,Z,T), U(X,0,Z,T), U(X,YL,Z,T), U(X,Y,0,T) AND
C...     U(X,Y,ZL,T) ARE SPECIFIED
C...
C... (2) NEUMANN BOUNDARY CONDITIONS IN WHICH U (0,Y,Z,T),
C...     X
C...     U (XL,Y,Z,T), U (X,0,Z,T), U (X,YL,Z,T), U (X,Y,0,T)
C...     X     Y     Y     Z
C...     AND U (X,Y,ZL,T) ARE SPECIFIED
C...             Z

```

continues

C...
C... IN THE SUBSEQUENT DISCUSSION, THE DERIVATIVES IN EQUATION (1)
C... WILL BE DESIGNATED WITH IN LINE NOTATION, E.G., U_{XX} WILL BE
C... DESIGNATED UXX.
C...
C... SINCE THE EXACT SOLUTION, EQUATION (2), IS AVAILABLE, THE ERROR
C... IN THE NUMERICAL SOLUTION CAN BE COMPUTED. IN PARTICULAR, THE
C... ERROR IN U(X=XL/2,Y=YL/2,Z=ZL/2,T=1) IS COMPUTED FROM THE FOLLOW-
C... ING APPLICATION OF EQUATION (2)
C...
C... U(0.5,0.5,0.5,1) = 1/(1 + EXP(0.5/(2*VIS) + 0.5/(2*VIS)
C... + 0.5/(2*VIS) - 3*1/(4*VIS)))
C... = 1/(1 + EXP(0)) = 0.5
C...
C... NOTE THAT THIS RESULT IS INDEPENDENT OF VIS. THUS, VIS CAN BE
C... DECREASED TO MAKE THE NUMERICAL SOLUTION MORE DIFFICULT TO
C... COMPUTE, I.E., THE PROBLEM IS MORE HYPERBOLIC AND LESS PARABOLIC,
C... YET THE NUMERICAL SOLUTION SHOULD REMAIN CLOSE TO 0.5 IF REASON-
C... ABLE ACCURACY IS TO BE MAINTAINED.
C...
C... IN ADDITION TO THE COMMENTS IN THE FOLLOWING CODING, THE READER
C... CAN ALSO REFER TO THE COMMENTS IN THE SPATIAL DIFFERENTIATION
C... SUBROUTINE, DSS036. IN PARTICULAR, CENTERED APPROXIMATIONS IN
C... SUBROUTINE DSS036 ARE USED TO CALCULATE UXX, UYY, UZZ IN EQUATION
C... (1), I.E., CENTERED APPROXIMATIONS SHOULD BE USED IN PARABOLIC
C... PROBLEMS, AND NONCENTERED (FIVE POINT BIASED UPWIND) APPROXI-
C... MATIONS IN SUBROUTINE DSS036 ARE USED TO CALCULATE UX, UY, UZ IN
C... EQUATION (1), I.E., NONCENTERED APPROXIMATIONS SHOULD BE USED
C... IN HYPERBOLIC PROBLEMS. ALL OF THESE APPROXIMATIONS ARE DERIVED
C... FROM THE TAYLOR SERIES AS EXPLAINED IN DSS036, AND SUBORDINATE
C... ROUTINES DSS004 AND DSS020.
C...
C... TWO SPECIAL CASES OF THIS PROBLEM CAN BE CONSIDERED
C...
C... (1) VIS = 0
C...
C... IN THIS CASE, EQUATION (1) REDUCES TO
C...
C...
$$\begin{aligned} U_T &= -U_{XX} - U_{YY} - U_{ZZ} \\ &\quad - \frac{(U^{**2})}{2} - \frac{(U^{**2})}{2} - \frac{(U^{**2})}{2} \end{aligned} \tag{3}$$

C...
C... EQUATION (3) HAS THE ANALYTICAL SOLUTION
C...
C...
$$U(X, Y, Z, T) = (X + Y + Z)/(1 + 3*T) \tag{4}$$

C...
C... THE USE OF EQUATIONS (3) AND (4) IN THE FOLLOWING PROGRAMMING
C... REQUIRES SOME MODIFICATION OF THE PROGRAMMING.
C...
C... (2) BOUNDARY VALUE PARTIAL DIFFERENTIAL EQUATION (PDE)
C...
C... THE STEADY STATE FORM OF EQUATION (1) IS THE BOUNDARY VALUE
C... PARTIAL DIFFERENTIAL EQUATION
C...
C...
$$U_{XX} - U_{UU_X} + U_{UU_Y} - U_{UU_Y} + U_{UU_Z} - U_{UU_Z} = 0 \tag{5}$$

continues

C...
C... (NOTE THAT U = 0 IN EQUATION (1)), WHICH CAN ALSO BE CONSIDER-
C... T
C... ED AS A THREE DIMENSIONAL ELLIPTIC PDE (WITH INDEPENDENT VARI-
C... ABLES X, Y AND Z). THUS, THE FOLLOWING PROGRAMMING ILLUSTRATES
C... A METHOD FOR INTEGRATING ELLIPTIC PDES AS WELL AS HYPERBOLIC
C... AND PARABOLIC PDES (BY ALLOWING THE SOLUTION OF EQUATION (1)
C... TO PROCEED TO U = 0, FOR WHICH EQUATION (2), WITH LARGE T,
C... T
C... REDUCES TO:
C...
C... U(X,Y,Z,INF) = 1
C...
C... THIS DISCUSSION SUGGESTS A GENERAL METHOD FOR SOLVING BOUNDARY
C... VALUE PDES, I.E., A PSEUDO TIME DERIVATIVE IS ADDED TO THE PDES
C... AND THE RESULTING PDES ARE INTEGRATED TO STEADY STATE FOR WHICH
C... THE TIME DERIVATIVES ESSENTIALLY GO TO ZERO. THIS APPROACH HAS
C... BEEN USED EFFECTIVELY IN THE SOLUTION OF SOME RELATIVELY COM-
C... PLEX BOUNDARY-VALUE PDES. THIS APPROACH IS ALSO TERMED THE
C... METHOD OF FALSE TRANSIENTS.
C...
C... FINALLY, THE VERIFICATION OF EQUATION (2) AS A SOLUTION TO EQUA-
C... TION (1) FOLLOWS:
C...
C... LET EX = EXP(X/(2*VIS) + Y/(2*VIS) + Z/(2*VIS) - 3*T/(4*VIS)).
C... THEN EQUATION (2) CAN BE WRITTEN AS:
C...
C... U(X,Y,Z,T) = 1/(1 + EX)
C...
C... THE VARIOUS TERMS IN EQUATION (1) ARE:
C...
C... U_T = (-1/(1 + EX)**2)*EX*(-3/(4*VIS))
C...
C... U_X = (1/(1 + EX))*(-1/(1 + EX)**2)*EX*(1/(2*VIS))
C...
C... -VIS*U_XX = -VIS*((-1/(1 + EX)**2)*EX*(1/(2*VIS))**2
C...
C... + EX*(1/(2*VIS)**2/(1 + EX)**3)*EX*(1/(2*VIS)))
C...
C... U_Y = (1/(1 + EX))*(-1/(1 + EX)**2)*EX*(1/(2*VIS))
C...
C... -VIS*U_YY = -VIS*((-1/(1 + EX)**2)*EX*(1/(2*VIS))**2
C...
C... + EX*(1/(2*VIS)**2/(1 + EX)**3)*EX*(1/(2*VIS)))
C...
C... U_Z = (1/(1 + EX))*(-1/(1 + EX)**2)*EX*(1/(2*VIS))
C...
C... -VIS*U_ZZ = -VIS*((-1/(1 + EX)**2)*EX*(1/(2*VIS))**2
C...
C... SUMMATION OF THESE TERMS GIVES:
C...
C... 0 = (1/(1 + EX)**2)*EX*(1/(4*VIS))*(3 + 1 + 1 + 1)
C... - (1/(1 + EX)**3)*EX*(1/(2*VIS))*(1 + 1 + 1)*(1 + EX)
C... = 0

continues

```

C...
C... WE NOW PROCEED TO THE NUMERICAL METHOD OF LINES SOLUTION OF EQUA-
C... TION (1) SUBJECT TO DIRICHLET AND NEUMANN BOUNDARY CONDITIONS, AND
C... AN INITIAL CONDITION, WHICH ARE CONSISTENT WITH THE EXACT SOLU-
C... TION, EQUATION (2).
C...
C... COMMON
COMMON/T/           T,      NSTOP,      NORUN
1     /Y/   U(6,6,6)
2     /F/   UT(6,6,6)
3     /S/   UX(6,6,6), UY(6,6,6), UZ(6,6,6),
4           UXX(6,6,6), UYY(6,6,6), UZZ(6,6,6)
5     /R/    XL,        YL,        ZL,
6           X(6),      Y(6),      Z(6),
7           VIS
8     /I/    NX,        NY,        NZ,
9           IP

C...
C... EQUATION PARAMETERS
VIS=1.0E+00
XL=1.0E+00
YL=1.0E+00
ZL=1.0E+00

C...
C... NUMBER OF GRID POINTS
NX=6
NY=6
NZ=6

C...
C... INITIAL CONDITIONS (T = 0 IN EQUATION (2))
DO 1 I=1,NX
X(I)=XL*FLOAT(I-1)/FLOAT(NX-1)
DO 1 J=1,NY
Y(J)=YL*FLOAT(J-1)/FLOAT(NY-1)
DO 1 K=1,NZ
Z(K)=ZL*FLOAT(K-1)/FLOAT(NZ-1)
U(I,J,K)=UA(X(I),Y(J),Z(K),T,VIS)
1 CONTINUE

C...
C... INITIAL DERIVATIVES
CALL DERV
IP=0
RETURN
END

SUBROUTINE DERV
COMMON/T/           T,      NSTOP,      NORUN
1     /Y/   U(6,6,6)
2     /F/   UT(6,6,6)
3     /S/   UX(6,6,6), UY(6,6,6), UZ(6,6,6),
4           UXX(6,6,6), UYY(6,6,6), UZZ(6,6,6)
5     /R/    XL,        YL,        ZL,
6           X(6),      Y(6),      Z(6),
7           VIS
8     /I/    NX,        NY,        NZ,
9           IP

C...
C... DIRICHLET BOUNDARY CONDITIONS WITH (1) UXX, UYY AND UZZ CALCULA-
C... TED BY CENTERED APPROXIMATIONS AND (2) UX, UY AND UZ CALCULATED
C... BY BIASED UPWIND APPROXIMATIONS
IF(NORUN.EQ.1)CALL DERV1

```

continues

```

C...
C... DIRICHLET BOUNDARY CONDITIONS WITH UXX, UYY, UZZ, UX, UY AND UZ
C... CALCULATED BY CENTERED APPROXIMATIONS
C... IF(NORUN.EQ.2)CALL DERV2
C...
C... NEUMANN BOUNDARY CONDITIONS WITH (1) UXX, UYY AND UZZ CALCULA-
C... TED BY CENTERED APPROXIMATIONS AND (2) UX, UY AND UZ CALCULATED
C... BY BIASED UPWIND APPROXIMATIONS
C... IF(NORUN.EQ.3)CALL DERV3
C...
C... NEUMANN BOUNDARY CONDITIONS WITH UXX, UYY, UZZ, UX, UY AND UZ
C... CALCULATED BY CENTERED APPROXIMATIONS
C... IF(NORUN.EQ.4)CALL DERV4
C... RETURN
C... END

SUBROUTINE DERV1
C...
C... DIRICHLET BOUNDARY CONDITIONS
C...
COMMON/T/           T,      NSTOP,      NORUN
1      /Y/   U(6,6,6)
2      /F/   UT(6,6,6)
3      /S/   UX(6,6,6), UY(6,6,6), UZ(6,6,6),
4      UXX(6,6,6), UYY(6,6,6), UZZ(6,6,6)
5      /R/   XL,      YL,      ZL,
6      X(6),     Y(6),     Z(6),
7      VIS
8      /I/   NX,      NY,      NZ,
9      IP

C...
C... X = 0, 0 LE Y LE YL, 0 LE Z LE ZL (X = 0 IN EQUATION (2))
DO 1 J=1,NY
DO 1 K=1,NZ
U(1,J,K)=UA(X(1),Y(J),Z(K),T,VIS)
1 CONTINUE

C...
C... X = XL, 0 LE Y LE YL, 0 LE Z LE ZL (X = XL IN EQUATION (2))
DO 2 J=1,NY
DO 2 K=1,NZ
U(NX,J,K)=UA(X(NX),Y(J),Z(K),T,VIS)
2 CONTINUE

C...
C... Y = 0, 0 LE X LE XL, 0 LE Z LE ZL (Y = 0 IN EQUATION (2))
DO 3 I=1,NX
DO 3 K=1,NZ
U(I,1,K)=UA(X(I),Y(1),Z(K),T,VIS)
3 CONTINUE

C...
C... Y = YL, 0 LE X LE XL, 0 LE Z LE ZL (Y = YL IN EQUATION (2))
DO 4 I=1,NX
DO 4 K=1,NZ
U(I,NY,K)=UA(X(I),Y(NY),Z(K),T,VIS)
4 CONTINUE

C...
C... Z = 0, 0 LE X LE XL, 0 LE Y LE YL (Z = 0 IN EQUATION (2))
DO 5 I=1,NX
DO 5 J=1,NY
U(I,J,1)=UA(X(I),Y(J),Z(1),T,VIS)
5 CONTINUE

```

continues

```

C...
C...      Z = ZL, 0 LE X LE XL, 0 LE Y LE YL (Z = ZL EQUATION (2))
DO 6 I=1,NX
DO 6 J=1,NY
U(I,J,NZ)=UA(X(I),Y(J),Z(NZ),T,VIS)
6      CONTINUE
C...
C...  SPATIAL DERIVATIVES
C...
C...      UX IN UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS036(0.E+00,XL,NX,NY,NZ,1,U,UX,0.E+00)
C...
C...      UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS036(0.E+00,XL,NX,NY,NZ,1,UX,UX,0.E+00)
C...
C...      UX IN U*UX (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS036(0.E+00,XL,NX,NY,NZ,1,U,UX,1.0E+00)
C...
C...      UY IN UYY (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS036(0.E+00,YL,NX,NY,NZ,2,U,UY,0.E+00)
C...
C...      UYY (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS036(0.E+00,YL,NX,NY,NZ,2,UY,UYY,0.E+00)
C...
C...      UY IN U*UY (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS036(0.E+00,YL,NX,NY,NZ,2,U,UY,1.0E+00)
C...
C...      UZ IN UZZ (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,U,UZ,0.E+00)
C...
C...      UZZ (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,U,UZ,0.E+00)
C...
C...      UZ IN U*UZ (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,U,UZ,1.0E+00)
C...
C...  BURGERS' EQUATION
C...
      DO 10 I=1,NX
      DO 10 J=1,NY
      DO 10 K=1,NZ
      UT(I,J,K)=-U(I,J,K)*UX(I,J,K)+VIS*UXX(I,J,K)
1           -U(I,J,K)*UY(I,J,K)+VIS*UYY(I,J,K)
2           -U(I,J,K)*UZ(I,J,K)+VIS*UZZ(I,J,K)
10     CONTINUE
      RETURN
      END

      SUBROUTINE DERV2
C...
C...  DIRICHLET BOUNDARY CONDITIONS
C...
      COMMON/T/,          T,      NSTOP,      NORUN
1           /Y/,    U(6,6,6)
2           /F/,    UT(6,6,6)
3           /S/,    UX(6,6,6), UY(6,6,6), UZ(6,6,6),
4           UXX(6,6,6), UYY(6,6,6), UZZ(6,6,6)
5           /R/,    XL,      YL,      ZL,
6           X(6),    Y(6),    Z(6),
7           VIS
8           /I/,    NX,      NY,      NZ,
9           IP

```

continues

```

C...
C...     X = 0, 0 LE Y LE YL, 0 LE Z LE ZL (X = 0 IN EQUATION (2))
      DO 1 J=1,NY
      DO 1 K=1,NZ
      U(1,J,K)=UA(X(1),Y(J),Z(K),T,VIS)
1      CONTINUE
C...
C...     X = XL, 0 LE Y LE YL, 0 LE Z LE ZL (X = XL IN EQUATION (2))
      DO 2 J=1,NY
      DO 2 K=1,NZ
      U(NX,J,K)=UA(X(NX),Y(J),Z(K),T,VIS)
2      CONTINUE
C...
C...     Y = 0, 0 LE X LE XL, 0 LE Z LE ZL (Y = 0 IN EQUATION (2))
      DO 3 I=1,NX
      DO 3 K=1,NZ
      U(I,1,K)=UA(X(I),Y(1),Z(K),T,VIS)
3      CONTINUE
C...
C...     Y = YL, 0 LE X LE XL, 0 LE Z LE ZL (Y = YL IN EQUATION (2))
      DO 4 I=1,NX
      DO 4 K=1,NZ
      U(I,NY,K)=UA(X(I),Y(NY),Z(K),T,VIS)
4      CONTINUE
C...
C...     Z = 0, 0 LE X LE XL, 0 LE Y LE YL (Z = 0 IN EQUATION (2))
      DO 5 I=1,NX
      DO 5 J=1,NY
      U(I,J,1)=UA(X(I),Y(J),Z(1),T,VIS)
5      CONTINUE
C...
C...     Z = ZL, 0 LE X LE XL, 0 LE Y LE YL (Z = ZL EQUATION (2))
      DO 6 I=1,NX
      DO 6 J=1,NY
      U(I,J,NZ)=UA(X(I),Y(J),Z(NZ),T,VIS)
6      CONTINUE
C...     SPATIAL DERIVATIVES
C...
C...     UX BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,XL,NX,NY,NZ,1,U,UX,0.E+00)
C...
C...     UXX BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,XL,NX,NY,NZ,1,UX,UXX,0.E+00)
C...
C...     UY BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,YL,NX,NY,NZ,2,U,UY,0.E+00)
C...
C...     UYY BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,YL,NX,NY,NZ,2,UY,UYY,0.E+00)
C...
C...     UZ BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,U,UZ,0.E+00)
C...
C...     UZZ BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,UZ,UZZ,0.E+00)
C...     BURGERS' EQUATION
C...
      DO 10 I=1,NX
      DO 10 J=1,NY
      DO 10 K=1,NZ

```

continues

```

UT(I,J,K)=-U(I,J,K)*UX(I,J,K)+VIS*UXX(I,J,K)
1           -U(I,J,K)*UY(I,J,K)+VIS*UYY(I,J,K)
2           -U(I,J,K)*UZ(I,J,K)+VIS*UZZ(I,J,K)
10      CONTINUE
      RETURN
      END

      SUBROUTINE DERV3
C...
C...  NEUMANN BOUNDARY CONDITIONS
C...
COMMON/T/          T,      NSTOP,      NORUN
1      /Y/    U(6,6,6)
2      /F/    UT(6,6,6)
3      /S/    UX(6,6,6), UY(6,6,6), UZ(6,6,6),
4           UXX(6,6,6), UYY(6,6,6), UZZ(6,6,6)
5      /R/    XL,        YL,        ZL,
6           X(6),      Y(6),      Z(6),
7           VIS
8      /I/    NX,        NY,        NZ,
9           IP

C...
C...  SPATIAL DERIVATIVES
C...
C...  UX IN UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS036(0.E+00,XL,NX,NY,NZ,1,U,UX,0.E+00)
C...
C...  X = 0, 0 LE Y LE YL, 0 LE Z LE ZL (X = 0 IN EQUATION (2))
DO 1 J=1,NY
DO 1 K=1,NZ
UX(1,J,K)=DUA(X(1),Y(J),Z(K),T,VIS)
1      CONTINUE

C...
C...  X = XL, 0 LE Y LE YL, 0 LE Z LE ZL (X = XL IN EQUATION (2))
DO 2 J=1,NY
DO 2 K=1,NZ
UX(NX,J,K)=DUA(X(NX),Y(J),Z(K),T,VIS)
2      CONTINUE

C...
C...  UXX (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS036(0.E+00,XL,NX,NY,NZ,1,UX,UXX,0.E+00)
C...
C...  UX IN U*UX (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
CALL DSS036(0.E+00,XL,NX,NY,NZ,1,U,UX,1.0E+00)
C...
C...  X = 0, 0 LE Y LE YL, 0 LE Z LE ZL (X = 0 IN EQUATION (2))
DO 7 J=1,NY
DO 7 K=1,NZ
UX(1,J,K)=DUA(X(1),Y(J),Z(K),T,VIS)
7      CONTINUE

C...
C...  X = XL, 0 LE Y LE YL, 0 LE Z LE ZL (X = XL IN EQUATION (2))
DO 8 J=1,NY
DO 8 K=1,NZ
UX(NX,J,K)=DUA(X(NX),Y(J),Z(K),T,VIS)
8      CONTINUE

C...
C...  UY IN UYY (BY FIVE POINT CENTERED APPROXIMATIONS)
CALL DSS036(0.E+00,YL,NX,NY,NZ,2,U,UY,0.E+00)
C...
C...  Y = 0, 0 LE X LE XL, 0 LE Z LE ZL (Y = 0 IN EQUATION (2))
DO 3 I=1,NX
DO 3 K=1,NZ
UY(I,1,K)=DUA(X(I),Y(1),Z(K),T,VIS)
3      CONTINUE

```

continues

```

C...
C...     Y = YL, 0 LE X LE XL, 0 LE Z LE ZL (Y = YL IN EQUATION (2))
C...     DO 4 I=1,NX
C...     DO 4 K=1,NZ
C...     UY(I,NY,K)=DUA(X(I),Y(NY),Z(K),T,VIS)
4      CONTINUE
C...
C...     UYY (BY FIVE POINT CENTERED APPROXIMATIONS)
C...     CALL DSS036(0.E+00,YL,NX,NY,NZ,2,UY,UYY,0.E+00)
C...
C...     UY IN U*UY (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
C...     CALL DSS036(0.E+00,YL,NX,NY,NZ,2,U,UY,1.0E+00)
C...
C...     Y = 0, 0 LE X LE XL, 0 LE Z LE ZL (Y = 0 IN EQUATION (2))
C...     DO 9 I=1,NX
C...     DO 9 K=1,NZ
C...     UY(I,1,K)=DUA(X(I),Y(1),Z(K),T,VIS)
9      CONTINUE
C...
C...     Y = YL, 0 LE X LE XL, 0 LE Z LE ZL (Y = YL IN EQUATION (2))
C...     DO 10 I=1,NX
C...     DO 10 K=1,NZ
C...     UY(I,NY,K)=DUA(X(I),Y(NY),Z(K),T,VIS)
10    CONTINUE
C...
C...     UZ IN UZZ (BY FIVE POINT CENTERED APPROXIMATIONS)
C...     CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,U,UZ,0.E+00)
C...
C...     Z = 0, 0 LE X LE XL, 0 LE Y LE YL (Z = 0 IN EQUATION (2))
C...     DO 5 I=1,NX
C...     DO 5 J=1,NY
C...     UZ(I,J,1)=DUA(X(I),Y(J),Z(1),T,VIS)
5      CONTINUE
C...
C...     Z = ZL, 0 LE X LE XL, 0 LE Y LE YL (Z = ZL EQUATION (2))
C...     DO 6 I=1,NX
C...     DO 6 J=1,NY
C...     UZ(I,J,NZ)=DUA(X(I),Y(J),Z(NZ),T,VIS)
6      CONTINUE
C...
C...     UZZ (BY FIVE POINT CENTERED APPROXIMATIONS)
C...     CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,UZZ,0.E+00)
C...
C...     UZ IN U*UZ (BY FIVE POINT BIASED UPWIND APPROXIMATIONS)
C...     CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,U,UZ,1.0E+00)
C...
C...     Z = 0, 0 LE X LE XL, 0 LE Y LE YL (Z = 0 IN EQUATION (2))
C...     DO 11 I=1,NX
C...     DO 11 J=1,NY
C...     UZ(I,J,1)=DUA(X(I),Y(J),Z(1),T,VIS)
11    CONTINUE
C...
C...     Z = ZL, 0 LE X LE XL, 0 LE Y LE YL (Z = ZL EQUATION (2))
C...     DO 12 I=1,NX
C...     DO 12 J=1,NY
C...     UZ(I,J,NZ)=DUA(X(I),Y(J),Z(NZ),T,VIS)
12    CONTINUE
C...
C...     BURGERS' EQUATION
C...
DO 13 I=1,NX
DO 13 J=1,NY
DO 13 K=1,NZ

```

continues

```

1      UT(I,J,K)=-U(I,J,K)*UX(I,J,K)+VIS*UXX(I,J,K)
2              -U(I,J,K)*UY(I,J,K)+VIS*UYY(I,J,K)
3              -U(I,J,K)*UZ(I,J,K)+VIS*UZZ(I,J,K)
13     CONTINUE
      RETURN
      END

      SUBROUTINE DERV4
C...
C...  NEUMANN BOUNDARY CONDITIONS
C...
      COMMON/T/           T,      NSTOP,      NORUN
1      /Y/   U(6,6,6)
2      /F/   UT(6,6,6)
3      /S/   UX(6,6,6), UY(6,6,6), UZ(6,6,6),
4          UXX(6,6,6), UYY(6,6,6), UZZ(6,6,6)
5      /R/   XL,        YL,        ZL,
6          X(6),       Y(6),       Z(6),
7          VIS
8      /I/   NX,        NY,        NZ,
9          IP

C...
C...  SPATIAL DERIVATIVES
C...
C...  UX BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,XL,NX,NY,NZ,1,U,UX,0.E+00)
C...
C...  X = 0, 0 LE Y LE YL, 0 LE Z LE ZL (X = 0 IN EQUATION (2))
      DO 1 J=1,NY
      DO 1 K=1,NZ
      UX(1,J,K)=DUA(X(1),Y(J),Z(K),T,VIS)
1     CONTINUE

C...  X = XL, 0 LE Y LE YL, 0 LE Z LE ZL (X = XL IN EQUATION (2))
      DO 2 J=1,NY
      DO 2 K=1,NZ
      UX(NX,J,K)=DUA(X(NX),Y(J),Z(K),T,VIS)
2     CONTINUE

C...  UXX BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,XL,NX,NY,NZ,1,UX,UXX,0.E+00)
C...
C...  UY BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,YL,NX,NY,NZ,2,U,UY,0.E+00)
C...
C...  Y = 0, 0 LE X LE XL, 0 LE Z LE ZL (Y = 0 IN EQUATION (2))
      DO 3 I=1,NX
      DO 3 K=1,NZ
      UY(I,1,K)=DUA(X(I),Y(1),Z(K),T,VIS)
3     CONTINUE

C...  Y = YL, 0 LE X LE XL, 0 LE Z LE ZL (Y = YL IN EQUATION (2))
      DO 4 I=1,NX
      DO 4 K=1,NZ
      UY(I,NY,K)=DUA(X(I),Y(NY),Z(K),T,VIS)
4     CONTINUE

C...  UYY BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,YL,NX,NY,NZ,2,UY,UYY,0.E+00)
C...
C...  UZ BY FIVE POINT CENTERED APPROXIMATIONS
      CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,U,UZ,0.E+00)

```

continues

```

C...
C...      Z = 0, 0 LE X LE XL, 0 LE Y LE YL (Z = 0 IN EQUATION (2))
C...      DO 5 I=1,NX
C...      DO 5 J=1,NY
C...      UZ(I,J,1)=DUA(X(I),Y(J),Z(1),T,VIS)
5       CONTINUE
C...
C...      Z = ZL, 0 LE X LE XL, 0 LE Y LE YL (Z = ZL EQUATION (2))
C...      DO 6 I=1,NX
C...      DO 6 J=1,NY
C...      UZ(I,J,NZ)=DUA(X(I),Y(J),Z(NZ),T,VIS)
6       CONTINUE
C...
C...      UZZ BY FIVE POINT APPROXIMATIONS
C...      CALL DSS036(0.E+00,ZL,NX,NY,NZ,3,UZ,UZZ,0.E+00)
C...
C...      BURGERS' EQUATION
C...
        DO 10 I=1,NX
        DO 10 J=1,NY
        DO 10 K=1,NZ
        UT(I,J,K)=-U(I,J,K)*UX(I,J,K)+VIS*UXX(I,J,K)
1           -U(I,J,K)*UY(I,J,K)+VIS*UYY(I,J,K)
2           -U(I,J,K)*UZ(I,J,K)+VIS*UZZ(I,J,K)
10      CONTINUE
        RETURN
        END

        FUNCTION UA(X,Y,Z,T,VIS)
C...
C...      FUNCTION UA IMPLEMENTS THE ANALYTICAL SOLUTION, EQUATION (2)
C...
        EX=EXP(X/(2.0E+00*VIS)
1           +Y/(2.0E+00*VIS)
2           +Z/(2.0E+00*VIS)
3           -3.0E+00*T/(4.0E+00*VIS))
        UA=1.0E+00/(1.0E+00+EX)
        RETURN
        END

        FUNCTION DUA(X,Y,Z,T,VIS)
C...
C...      FUNCTION DUA IMPLEMENTS THE DERIVATIVE OF THE ANALYTICAL SOLUTION,
C...      EQUATION (2), WITH RESPECT TO X, Y AND Z
C...
        EX=EXP(X/(2.0E+00*VIS)
1           +Y/(2.0E+00*VIS)
2           +Z/(2.0E+00*VIS)
3           -3.0E+00*T/(4.0E+00*VIS))
        DUA=-1.0E+00*(1.0E+00+EX)**(-2)*EX*(1.0E+00/(2.0E+00*VIS))
        RETURN
        END

        SUBROUTINE PRINT(NI,NO)
        COMMON/T/           T,          NSTOP,         NORUN
1           /Y/   U(6,6,6)
2           /F/   UT(6,6,6)
3           /S/   UX(6,6,6), UY(6,6,6), UZ(6,6,6),
4           UXX(6,6,6), UYY(6,6,6), UZZ(6,6,6)
5           /R/    XL,          YL,          ZL,
6           X(6),          Y(6),          Z(6),
7           VIS
8           /I/    NX,          NY,          NZ,
9           IP

```

continues

```

C...
C... PRINT A HEADING FOR THE NUMERICAL SOLUTION
IP=IP+1
IF(IP.EQ.1)WRITE(NO,1)
1 FORMAT(11X,'T',9X,'U(0.6,0.6,0.6,T)',,
1           5X,'U(0.6,0.6,0.6,T)',5X,'ERROR',/,
2           25X,'(NUM)',12X,'(ANAL)')
C...
C... COMPUTE THE ANALYTICAL SOLUTION AND THE ERROR IN THE NUMERICAL
C... SOLUTION. NOTE THAT FOR AN EVEN NUMBER OF POINTS, E.G., 6, THE
C... MID POINTS X = XL/2, Y = YL/2, Z = ZL/2 ARE NOT AVAILABLE.
C... THEREFORE, THE POINTS X = 0.6*XL, Y = 0.6*YL, Z = 0.6*ZL ARE
C... USED FOR WHICH THE EXACT SOLUTION WITH T = 1.2 IS:
C...
C... U(0.6,0.6,0.6,1.2) =
C...
C...     = 1/(1 + EXP(0.6/(2*VIS) + 0.6/(2*VIS) + 0.6/(2*VIS)
C...
C...     - 3*1.2/(4*VIS)) = 1/(1 + EXP(0)) = 0.5
C...
NX2=NX/2+1
NY2=NY/2+1
NZ2=NZ/2+1
XL2=X(NX2)
YL2=Y(NY2)
ZL2=Z(NZ2)
UANAL=UA(XL2,YL2,ZL2,T,VIS)
ERROR=U(NX2,NY2,NZ2)-UANAL
C...
C... PRINT THE NUMERICAL AND ANALYTICAL SOLUTIONS, AND THE ERROR
WRITE(NO,2)T,U(NX2,NY2,NZ2),UANAL,ERROR
2 FORMAT(F12.2,2F21.6,E16.4)
RETURN
END

THREE-D BURGERS' EQUATION, DIRICHLET BC
0.          2.0          0.2
216 1000          0.001
END OF RUNS
THREE-D BURGERS' EQUATION, DIRICHLET BC
0.          2.0          0.2
216 1000          0.001
THREE-D BURGERS' EQUATION, NEUMANN BC
0.          2.0          0.2
216 1000          0.001
THREE-D BURGERS' EQUATION, NEUMANN BC
0.          2.0          0.2
216 1000          0.001
END OF RUNS

```

differences are: (a) the use of a $6 \times 6 \times 6$ grid (rather than 11×11 in the two-dimensional case) and (b) calls to subroutine DSS036 (rather than DSS034) to calculate the boundary-value derivatives in equation (6.22). The arguments of DSS036 are illustrated by the following call:

```
CALL DSS036(X1,X2,NX,NY,NZ,ND,U,UX,V)
```

X1 the lower boundary value of the independent variable appearing in the derivative to be calculated (input)

- X2 the upper boundary value of the independent variable appearing in the derivative to be calculated (input)
- NX number of spatial grid points in the x direction (input)
- NY number of spatial grid points in the y direction (input)
- NZ number of spatial grid points in the z direction (input)
- ND number of the independent variable appearing in the derivative to be calculated; ND = 1 for x , ND = 2 for y , ND = 3 for z (input)
- U three-dimensional array, of size U(NX, NY, NZ), containing the dependent variable to be differentiated (input)
- UX three-dimensional array, of size UX(NX, NY, NZ), containing the computed first derivative of the dependent variable u with respect to the independent variable with number ND (output)
- V direction indicator for the five-point biased upwind approximations in DSS036; V > 1 for flow in the positive direction; V = 0 for no flow (diffusion only); V < 1 for flow in the negative direction (input)

Documentation comments at the beginning of DSS036 also describe the arguments in detail.

DSS036 actually executes a series of calls to DSS004 (when the last argument is zero) for parabolic (diffusion) derivatives such as μu_{xx} , μu_{yy} , and μu_{zz} in equation (6.22), and executes a series of calls to DSS020 (when the last argument is not zero) for hyperbolic (convective) derivatives such as $-uu_x$, $-uu_y$, and $-uu_z$.

The data of Program 6.3 are for four runs in which subroutines DERV1 to DERV4 are executed. However, because of space limitations, only the abbreviated output from the first run is listed in Table 6.3. The $6 \times 6 \times 6$ -point grid in x , y , and z (216 ODEs) gives accuracy better than 10^{-5} (absolute). Also, because $\mu = 1$ gives a relatively smooth solution, the hyperbolic terms $-uu_x$, $-uu_y$, and $-uu_z$ can be calculated by centered differences (in DERV2 for Dirichlet boundary conditions and in DERV4 for Neumann boundary conditions); if equation (6.22) were strongly hyperbolic ($\mu \ll 1$), only five-point biased upwind differences should be used.

This completes the discussion of the NUMOL solution of multidimensional PDEs. Additional information is available by studying subroutines DSS034 and DSS036. Also, the preceding examples should indicate how systems of multidimensional PDEs can be integrated by the NUMOL.

We now proceed to the final example in this book, Burgers' one-dimensional equation with a strong hyperbolic characteristic [$\mu = 0.05$ in

Table 6.3 Numerical Output from Program 6.3

```
RUN NO. - 1 THREE-D BURGERS' EQUATION, DIRICHLET BC
INITIAL T - 0.000E+00
FINAL T - .200E+01
PRINT T - .200E+00
NUMBER OF DIFFERENTIAL EQUATIONS - 216
MAXIMUM INTEGRATION ERROR - .100E-02
```

T	$U(0.6, 0.6, 0.6, T)$ (NUM)	$U(0.6, 0.6, 0.6, T)$ (ANAL)	ERROR
0.00	.289050	.289050	0.0000E+00
.20	.320820	.320821	-.1211E-05
.40	.354344	.354344	-.1249E-06
.60	.389366	.389361	.5424E-05
.80	.425561	.425557	.3914E-05
1.00	.462572	.462570	.1704E-05
1.20	.500000	.500000	.4396E-07
1.40	.537431	.537430	.1333E-05
1.60	.574443	.574443	.2808E-06
1.80	.610635	.610639	-.3778E-05
2.00	.645655	.645656	-.1446E-05

equation (6.18)]. The solution for this case exhibits steep moving fronts that are difficult to resolve with the fixed spatial grid used in the previous examples. We will therefore employ an adaptive grid that automatically concentrates the grid points where they are needed to resolve the steep fronts.

6.5 Adaptive Grids

If we compare the advection equation discussed in Section 3.5 (renumbered here)

$$u_t = -vu_x \quad (6.24)$$

and the one-dimensional Burgers equation

$$u_t = -uu_x + \mu u_{xx} \quad (6.18)$$

we see that they are similar, particularly if $\mu = 0$. Note that equation (6.18) has a velocity u in analogy with v in equation (6.24). However, the velocity in equation (6.18) is not constant, but is the dependent variable u . Thus, we might expect some characteristics of the solution of equation (6.18) that were not observed in the case of equation (6.24) as discussed in Section 3.5.

In particular, if the initial condition for equation (6.18) is a positive, monotonically decreasing function of x (as x increases, u decreases at

$t = 0$), then equation (6.18) initially has a higher velocity for small x than for large x . In other words, the solution will move faster in the positive x direction for small x than for large x (recall how the solutions to equation (6.24) moved in the positive x direction for $v > 0$, as discussed in Section 3.5). This causes the initial condition to become sharper with time, a so-called *front-sharpening* characteristic. In fact, if t increases to a sufficiently large value, a *shock* or *discontinuity* will form in the solution, making the calculation of the solution very difficult. Before this actually occurs, the solution will have such a steep front that it cannot be resolved by the fixed-grid methods we have discussed previously. The solution will actually have a significant variation within a distance that is less than the grid spacing. If this happens, we cannot resolve the solution spatially unless we use a grid that concentrates the grid points where the solution changes rapidly in space. If we set up a NUMOL grid that concentrates the grid points automatically where they are needed, we will be using an *adaptive grid*. In the following example, we demonstrate how to program an adaptive grid for the one-dimensional Burgers equation [equation (6.18)].

Subroutine INITAL is listed in Program 6.4a. The coding in INITAL requires some explanation:

Program 6.4a Subroutine INITAL to Initialize an Adaptive Grid

```

SUBROUTINE INITAL
C...
C...  SOLUTION OF BURGERS' EQUATION BY SPLINE APPROXIMATION ON AN
C...  ADAPTIVE GRID
C...
C... SUBROUTINE INITAL
C...
C...      (1) DEFINES THE INITIAL ADAPTIVE GRID AS THE BASIC UNIFORM
C...          GRID VIA A CALL TO SUBROUTINE GRIDE
C...
C...      (2) DEFINES THE INITIAL CONDITION ON THE ADAPTIVE GRID OF
C...          (1) BY A CALL TO FUNCTION PHI WHICH IMPLEMENTS THE EXACT
C...          SOLUTION TO BURGERS' EQUATION
C...
C... COMMON/N/ IS ACCESSED SO THE NUMBER OF ODES DEFINED BY THE
C... ADAPTIVE GRID, N, CAN BE CHANGED AS THE GRID EVOLVES
C... COMMON/N/      N
C...
C... THE FOLLOWING COMMON AREAS ARE BRIEFLY EXPLAINED BELOW
C...
C...      /T/ /Y/ /F/      STANDARD COMMON AREAS DIMENSIONED FOR UP TO
C...                      150 ODES
C...
C...      /B/              THE SOLUTION, CB, AND ITS SECOND DERIVATIVE,
C...                      CBXX, DEFINED ON THE BASIC GRID
C...
C...      /SV/             INDEPENDENT VARIABLE DEFINED ON THE ADAPTIVE
C...                      GRID

```

continues

```

C...
C...      /SC/          CUBIC SPLINE COEFFICIENTS (LINEAR, QUADRATIC
C...                                         AND CUBIC COEFFICIENTS RESPECTIVELY)
C...
C...      /C/          CONSTANTS PERTAINING TO THE PROBLEM
C...
C...      /AB1/         PARAMETERS WHICH CONTROL THE ADAPTIVE GRID
C...
COMMON   /T/           T
1        /Y/           CA(150)
2        /F/           CT(150)
3        /B/           CB(120), CBXX(120)
4        /SV/          XA(150)
5        /SC/          CX(150), CXX(150), CXXX(150)
6        /C/           XL,          XU,          V,          VIS,
7                  SN,          NB,          IP,          JP
COMMON/AB1/             DL,          XFL,          XFR,          NF,
1                  DXF,         XB(120),    LXB(120),    NOB(120)
C...
C...  SET THE VELOCITY AND VISCOSITY OF BURGERS EQUATION
V=1.0
VIS=0.003
C...
C...  INITIALIZE THE COUNTERS FOR PRINTED AND PLOTTED OUTPUT USED
C...  IN SUBROUTINE PRINT, THE RUNNING SUM OF THE NUMBER OF ODES
C...  USED IN THE ADAPTIVE GRID
IP=0
JP=0
SN=0.0
C...
C...  DEFINE THE LENGTH OF THE X AXIS
XL=0.0
XU=1.0
C...
C...  DEFINE THE NUMBER OF BASIC GRID POINTS, NUMBER OF SUBINTERVALS
C...  BETWEEN THE BASIC GRID POINTS WHEN ADAPTIVE POINTS ARE ADDED
NB=41
NF=2
C...
C...  DEFINE AN INITIAL UNIFORM GRID OF NB POINTS
CALL GRIDE(NB,XL,XU,DXB,XB)
C...
C...  THE NUMBER OF ADAPTIVE GRID POINTS, N, INITIALLY EQUALS THE
C...  NUMBER OF BASIC GRID POINTS, NB
N=NB
C...
C...  DEFINE THE ADAPTIVE GRID INTERVAL, DXF, FROM THE BASIC GRID
C...  INTERVAL, DXB
DXF=DXB/FLOAT(NF)
C...
C...  DEFINE THE THRESHOLD FOR THE SECOND DERIVATIVE, ABOVE WHICH
C...  ADAPTIVE GRID POINTS ARE ADDED
DL=20.0
C...
C...  DEFINE THE MAXIMUM DISTANCES TO THE LEFT AND RIGHT OF A BASIC
C...  GRID POINT, XFL AND XFR, BEYOND WHICH ADAPTIVE GRID POINTS WILL
C...  NOT BE ADDED
SFL=1.0
SFR=1.5
TP=0.02
XFL=SFL*V*TP
XFR=SFR*V*TP

```

continues

```

C...
C... DEFINE THE INITIAL BASIC AND ADAPTIVE GRIDS, EACH WITH NB POINTS
DO 1 I=1,NB
C...
C... INITIALIZE ARRAY LXB WHICH SPECIFIES THE EXISTENCE OF ADAPTIVE
C... GRID POINTS
LXB(I)=0
C...
C... STORE THE SUBSCRIPTS OF THE BASIC GRID IN ARRAY NOB
NOB(I)=I
C...
C... DEFINE THE INITIAL CONDITIONS OF THE BASIC AND ADAPTIVE GRIDS
C... FROM THE EXACT SOLUTION TO BURGERS' EQUATION
CB(I)=PHI(0.0,XB(I))
CA(I)=CB(I)
XA(I)=XB(I)
1 CONTINUE
C...
C... INITIALIZE THE DERIVATIVE CALCULATIONS IN SUBROUTINE DERV
CALL DERV
RETURN
END

```

(1) Since we will still be integrating a system of ODEs, the usual COMMON/T/, /Y/, and /F/ are required (note that the solution on the adaptive grid, CA(150), is in COMMON/Y/). Additionally, COMMON/B/ is defined for the solution on the basic grid, CB(120), and its second derivative in x , CBXX(120). The distinction between the solution on the adaptive grid, CA(150), and on the basic grid, CB(120), will be discussed subsequently. COMMON/SV/ contains the values of x on the adaptive grid XA(150), and COMMON/SC/ contains the first-, second-, and third-order coefficients of the cubic spline used to approximate the solution, CX(150), CXX(150), and CXXX(150), respectively. COMMON/C/ contains some constants used in other subroutines, and COMMON/AB1/ contains some constants and variables for the adaptive grid, and in particular, XB(120), the values of x on the basic grid.

(2) After setting some constants (V, VIS, . . . , XL, XU), the number of grid points in a basic grid, NB, is set to 41. The basic grid remains unchanged during the solution, and additional points are inserted between the basic grid points to make up the adaptive grid. These inserted points can be removed later if they are not required, but the basic grid remains. The maximum number of adaptive grid subintervals that can be inserted between two basic grid points is also set (NF), i.e., NF – 1 adaptive grid points are inserted between two basic grid points.

(3) Subroutine GRIDE is then called to set up the basic grid of NB points over the spatial interval $XL \leq x \leq XU$ with the grid point locations returned in array XB. The basic grid need not be uniform, but can, in fact, have unequal spacing in x , particularly if some advance knowledge of where the solution changes rapidly in x is known; however, once the basic grid is defined, it remains unchanged throughout the solution.

(4) The number of adaptive grid points, N , equals the number of basic grid points, N_B , initially. Then, as adaptive grid points are inserted and deleted during the course of the solution, N will change.

(5) The spacing of the adaptive grid, DX_F , is initially equal to the spacing of the basic grid, DX_B (returned from subroutine GRIDE), divided by the maximum number of subintervals between basic grid points, N_F .

(6) The decision to insert adaptive grid points is made on the basis of the magnitude of the second derivative of the numerical solution. If this second derivative is above a threshold, DL , at a particular x , N_F subintervals ($N_F - 1$ adaptive grid points) are inserted between two basic grid points (because the solution is changing rapidly, as reflected by the relatively large second derivative, so that additional grid points in x are required to resolve the solution with reasonable accuracy). If the second derivative of the numerical solution is below the threshold DL at a particular x , no additional adaptive grid points are added since they are not required to resolve the solution. Thus, the adaptive grid operates on a second derivative criterion, although any other criterion, or combination of criteria, can be used to define when adaptive grid points should be inserted. Similarly, if for a given value of x where N_F intervals have been already been inserted, the second derivative of the solution drops below DL , the inserted subintervals (or adaptive grid points) are removed and only the basic grid points are retained at that x .

(7) A large value of the second derivative of the numerical solution may exist over an interval in x to the left and to the right of a basic grid point. These intervals over which the adaptive grid algorithm should search to determine whether adaptive grid points should be added to the left and right of a basic grid point are defined by X_{FL} and X_{FR} , respectively. In the present case, they are computed from the approximate velocity of the moving front of the Burgers equation solution, V , and the distance the front would move at this velocity during the print interval, TP of the solution (although they can be specified in any way that seems to be a reasonable approximation for the width of the rapidly changing portion of the solution). Items (6) and (7) indicate that some “tuning” of the adaptive grid is required for a particular problem, i.e., the specification of DL , X_{FL} , and X_{FR} .

(8) Once the basic adaptive grid parameters are specified, the array LXB , which specifies the existence of adaptive grid points (at basic grid point index I), is initialized to zero; later, when adaptive grid points are added at basic grid point I , $LXB(I)$ will be changed to one. Similarly, if adaptive grid points at basic grid point I are deleted, $LXB(I)$ will be set back to zero. Finally, the subscripts of the basic grid points in the total (basic plus adaptive) grid are stored in array NOB so that the identity and

location of the basic grid points is retained; as expected, initially, the subscripts of the basic and total grid points are the same.

(9) Since the solution on the adaptive grid, CA, is in COMMON/Y/, it must be given initial conditions, which in this case are produced from the exact solution to Burgers' equation at $t = 0$, implemented in subroutine PHI. Also, the solutions on the basic and adaptive grids, and the locations of points on the basic and adaptive grids, are the same initially.

This completes the definition of the adaptive grid parameters, the initialization of the basic and adaptive grids, and the initial conditions for the adaptive grid ODEs. Subroutine DERV, listed in Program 6.4b, then defines the boundary conditions for Burgers's equation [equation (6.18)], computes the spatial derivatives in the RHS of Burgers' equation (u_x and u_{xx}), and computes the RHS of Burgers' equation ($-uu_x + \mu u_{xx}$) so that the ODE derivatives are available in COMMON/F/. In other words, subroutine DERV in this case performs the same operations as previously.

The only essential difference is the calculation of the spatial derivatives u_x and u_{xx} , which are available from the cubic spline coefficients, CX,

Program 6.4b Subroutine DERV to Compute the ODE Time Derivatives on an Adaptive Grid

```

SUBROUTINE DERV
C...
C...  SUBROUTINE DERV
C...
C...      (1) COMPUTES THE FIRST AND SECOND DERIVATIVES, UX AND UXX,
C...          VIA A CALL TO SUBROUTINE NCSPLE
C...
C...      (2) USES DERIVATIVES UX AND UXX TO ASSEMBLE BURGERS' EQUATION
C...          IN THE USUAL METHOD OF LINES FORMAT
C...
COMMON /N/           N
COMMON /T/           T
1     /Y/   CA(150)
2     /F/   CT(150)
3     /B/   CB(120), CBXX(120)
4     /SV/  XA(150)
5     /SC/  CX(150), CXX(150), CXXX(150)
6     /C/    XL,      XU,      V,      VIS,
7           SN,      NB,      IP,      JP
COMMON/AB1/
1           DL,      XFL,      XFR,      NF,
           DXF,     XB(120),  LXB(120), NOB(120)
C...
C...  APPLY THE BOUNDARY CONDITIONS FOR BURGERS' EQUATION AT X = 0
C...  AND X = 1
C...  CA(1)=PHI(T,0.0)
C...  CA(N)=PHI(T,1.0)
C...
C...  COMPUTE THE FIRST AND SECOND SPATIAL DERIVATIVES IN BURGERS'
C...  EQUATION, UX AND UXX, BY THE NORMAL CUBIC SPLINE
CALL NCSPLE(N, XA, CA, CX, CXX, CXXX)

```

continues

```

C...
C... ASSEMBLE BURGERS' EQUATION OVER GRID POINTS 2 TO N-1. COMMON
C... /SC/ ACTUALLY CONTAINS THE COEFFICIENTS OF THE CUBIC SPLINE
C... C(X) = A0 + A1*(X - XI) + A2*(X - XI)**2 + A3*(X - XI)**3,
C... I.E., COMMON/SC/A1(150),A2(150),A3(150), AFTER NCSPLE IS
C... CALLED.  THUS CX(XI) = A1, CXX(XI) = 2*A2, CXXX(XI) = 6*A3,
C... AND THE SECOND ARRAY IN /SC/ MUST BE MULTIPLIED BY 2 TO OBTAIN
C... THE SECOND DERIVATIVE, CXX, IN BURGERS' EQUATION. ALSO, FROM
C... THE EQUATION FOR THE SPLINE, U(XI) = A0 = CA(XI) WHERE ARRAY
C... CA IS IN COMMON/Y/. SUBROUTINE NCSPLE THEREFORE NUMERICALLY
C... DIFFERENTIATES CA TO CX, CXX AND CXXX AS WELL AS EVALUATING
C... THE COEFFICIENTS OF THE CUBIC SPLINE (THESE TWO OPERATIONS
C... ARE EQUIVALENT)
NM1=N-1
DO 1 I=2,NM1
CT(I)=2.0*VIS*CXX(I)-CA(I)*CX(I)
1 CONTINUE
C...
C... ZERO THE TIME DERIVATIVES AT I = 1, N TO INSURE THE BOUNDARY
C... CONDITIONS ARE OBSERVED
CT(1)=0.0
CT(N)=0.0
RETURN
END

```

CXX, and CXXX returned by subroutine NCSPLE. Cubic splines are used in this case because an approximation (interpolant) is required which operates on a nonuniform grid (the insertion and deletion of adaptive grid points results in a nonuniform grid spacing in x). Also as the comments explain, the second derivative u_{xx} is $2.0 * CXX$, which accounts for the factor of 2 in the calculation of the time derivative u_t in equation (6.18) [=CT(I) computed in DO loop 1]. Otherwise, subroutine NCSPLE operates in the same way as the previous spatial differentiation routines, e.g., DSS002 and DSS004.

We should also note that the number of ODEs, N, changes as adaptive grid points are added and deleted since an ODE is associated with each grid point (this variable number of ODEs contrasts with all of the previous solutions for which the number of spatial grid points was fixed initially and remained at that value throughout the solution so that the number of ODEs remained constant). The number of ODEs, N, is available through COMMON/N/. The value of N is set in subroutine PRINT, which is listed in Program 6.4C and discussed below.

Program 6.4c Subroutine PRINT to Print the Numerical Solution and Update the Adaptive Grid

```

SUBROUTINE PRINT(NI ,NO)
C...
C... SUBROUTINE PRINT
C...
C...      (1) REDEFINES THE ADAPTIVE GRID IN ACCORDANCE WITH CHANGES
C...          IN THE NUMERICAL SOLUTION VIA A CALL TO SUBROUTINE ANUGB1
C...
C...      (2) PLOTS THE NUMBER OF POINTS IN THE ADAPTIVE GRID VS T AT

```

continues

```

C...      THE END OF THE RUN AND PRINTS THE AVERAGE NUMBER OF GRID
C...      POINTS DURING THE RUN
C...
COMMON /N/          N
COMMON /T/          T
1   /Y/    CA(150)
2   /F/    CT(150)
3   /B/    CB(120), CBXX(120)
4   /SV/   XA(150)
5   /SC/   CX(150), CXX(150), CXXX(150)
6   /C/    XL,        XU,        V,        VIS,
7           SN,        NB,        IP,        JP
COMMON/AB1/         DL,        XFL,       XFR,       NF,
1           DXF,       XB(120),  LXB(120), NOB(120)
C...
C...  DIMENSION THE ARRAYS FOR THE PLOTTED SOLUTION
DIMENSION PN(51), TN(51)
C...
C...  INCREMENT THE INDEX FOR THE PLOTTED SOLUTION
IP=IP+1
C...
C...  COMPUTE THE RUNNING SUM OF THE NUMBER OF ODES
SN=SN+FLOAT(N)
C...
C...  STORE THE NUMBER OF ODES FOR SUBSEQUENT PLOTTING
PN(IP)=FLOAT(N)
TN(IP)=T
C...
C...  CALL SUBROUTINE NCSPLE TO UPDATE THE SPLINE COEFFICIENTS USED
C...  IN THE NEW ADAPTIVE GRID
CALL NCSPLE(N, XA, CA, CX, CXX, CXXX)
C...
C...  STORE THE DEPENDENT VARIABLE AND ITS SECOND DERIVATIVE ON THE
C...  ADAPTIVE GRID, CA(I) AND CXX(I), AT THE BASIC GRID POINTS WITH
C...  SUBSCRIPT K IN PREPARATION FOR UPDATING THE ADAPTIVE GRID.  THE
C...  FACTOR OF 2 MULTIPLYING CXX(K) IS EXPLAINED IN SUBROUTINE DERV
DO 1 I=1,NB
K=NOB(I)
CB(I)=CA(K)
CBXX(I)=2.0*CXX(K)
1 CONTINUE
C...
C...  CALL SUBROUTINE ANUGB1 TO REDEFINE THE ADAPTIVE GRID
CALL ANUGB1(NB, CB, CBXX, N, XA, CA)
C...
C...  AT THE END OF THE RUN, PLOT THE NUMBER OF ODES VS T
IF(IP.LT.51)RETURN
CALL TPLOTS(1, IP, TN, PN)
C...
C...  COMPUTE AND PRINT THE AVERAGE NUMBER OF ODES
AN=SN/FLOAT(IP)
WRITE(NO,5)AN
5 FORMAT(1H ,//,
1 7H N VS T,3X,8H AVE N = ,F6.2)
RETURN
END

```

Note also that the boundary conditions at $x = 0$ and $x = 1$ are specified by using the exact solution to Burgers' equation in function PHI. In this way, the initial conditions (set in subroutine INITIAL) and boundary conditions for the numerical and exact solutions are the same (and there-

fore, any differences between these two solutions are due to the errors in the numerical solution).

Subroutine PRINT (see Program 6.4c), in addition to printing and plotting the numerical solution, serves another essential function, the updating of the adaptive grid. The following details should be noted in particular:

(1) At the beginning of subroutine PRINT, some basic information is stored for subsequent plotting. In particular, the number of grid points, N (also the number of ODEs), is stored in array PN and the corresponding time is stored in array TN. Then, at the end of the solution, a plot of PN vs. TN will show how the number of ODEs varied with time.

(2) Subroutine NCSPLE is called to update the cubic spline coefficients. In particular, the second derivative of the solution, $2.0 * CX$, is updated for use in the redefinition of the adaptive grid (recall again, from the discussion of subroutine INITAL, that the second derivative of the solution is used to determine whether adaptive grid points should be added or deleted).

(3) In DO loop 1, which cycles through the basic grid points ($I = 1$ to NB), the subscripts of the basic grid points in NOB(I) are used to redefine the solution [CB(I)] and its second derivative [CBXX(I)] on the basic grid.

(4) Subroutine ANUGB1 is then called to redefine the adaptive grid; i.e., adaptive grid points are inserted (or deleted) where required by the second derivative CBXX. The new number of grid points (and therefore ODEs) N, grid point locations XA, and solution values CA are returned by ANUGB1 for the next section of the solution computed until subroutine PRINT is called again. Therefore, the adaptive grid is updated each time subroutine PRINT is called, and is used without modification in computing the section of the solution to the next call to PRINT. This, in effect, defines a new ODE problem each time PRINT is called. The values of CA from ANUGB1 serve as initial conditions for this new ODE problem; these initial conditions are computed in ANUGB1 by interpolation of the solution on the basic grid (from DO loop 1) using the cubic spline coefficients (updated by the call to NCSPLE). Subroutine ANUGB1 is not listed here because of limited space, but is included on the diskettes available from the author; it is the simplest of a series of adaptive grid routines (the most complex routine, ANUGB6, provides for five grid spacings differing by a factor of $\frac{1}{2}$, and therefore covering the range $\frac{1}{2}$ to $\frac{1}{2}^5$, rather than the single adaptive grid spacing of ANUGB1 set as DXF in subroutine INITAL).

Program 6.4d Function PHI for the Exact Solution to Burgers' Equation

```

REAL FUNCTION PHI(T,X)
C...
C...  FUNCTION PHI(T,X) COMPUTES THE EXACT SOLUTION TO BURGERS'
C...  EQUATION. IT IS USED IN THREE PLACES
C...
C...      (1) TO PROVIDE THE EXACT SOLUTION IN SUBROUTINE PRINT FOR
C...          COMPARISON WITH THE NUMERICAL SOLUTION.
C...
C...      (2) TO PROVIDE THE INITIAL CONDITION IN SUBROUTINE INITAL.
C...
C...      (3) TO PROVIDE THE BOUNDARY CONDITIONS IN SUBROUTINE DERV.
C...
C... ARGUMENT LIST
C...
C...      T    INITIAL-VALUE INDEPENDENT VARIABLE IN BURGERS' EQUATION
C...            (INPUT)
C...
C...      X    BOUNDARY-VALUE INDEPENDENT VARIABLE IN BURGERS' EQUATION
C...            (INPUT)
C...
C... THE VALUE OF THE FUNCTION (PHI) IS THE EXACT SOLUTION TO BURGERS'
C... EQUATION AT X AND T.
C...
COMMON /C/           XL,          XU,          V,          VIS
A=(0.05/VIS)*(X-0.5+4.95*T)
B=(0.25/VIS)*(X-0.5+0.75*T)
C=( 0.5/VIS)*(X-0.375)
EA=EXP(-A)
EB=EXP(-B)
EC=EXP(-C)
PHI=(0.1*EA+0.5*EB+EC)/(EA+EB+EC)
RETURN
END

```

Function PHI computes the exact solution, and is used in subroutines INITAL and DERV to set the initial and boundary conditions for the numerical solution as discussed previously; see Program 6.4d.

The data for the preceding Fortran are listed in Program 6.4e. Note, in particular, that the system starts with 41 ODEs (the solution on the basic grid), but, of course, this number changes (variable N defined by the call to the adaptive grid routine, ANUGB1, in subroutine PRINT at an interval of 0.02 in time).

The numerical output is listed in Table 6.4. The average number of ODEs for the complete solution (variable AN computed in subroutine PRINT) is 48.92, indicating that the increase in the number of ODEs due to the adaptive gridding is not large; rather, the adaptive grid is quite

Program 6.4e Data for the Adaptive Grid Solution of Burgers' Equation

```

NUMOL SOLUTION OF BURGERS' EQUATION BY SPLINES AND LSODE
O.          1.0          0.02
41          ABS 0.001
END OF RUNS

```

Table 6.4 Numerical Output from Program 6.4

```

RUN NO. - 1 NUMOL SOLUTION OF BURGERS' EQUATION BY SPLINES AND LSODE
INITIAL T - 0.000E+00
FINAL T - .100E+01
PRINT T - .200E-01

NUMBER OF DIFFERENTIAL EQUATIONS - 41
MAXIMUM INTEGRATION ERROR - .100E-02

N VS T    AVG N = 48.92

COMPUTATIONAL STATISTICS

LAST STEP SIZE           .522E-02
LAST ORDER OF THE METHOD 2
TOTAL NUMBER OF STEPS TAKEN 5
NUMBER OF FUNCTION EVALUATIONS 56
NUMBER OF JACOBIAN EVALUATIONS 1

```

efficient in placing the grid points where they are required, and in deleting grid points when they are no longer needed. This feature of the numerical solution is illustrated by the graphical output of Program 6.4 given in Figures 6.1a to 6.1c. Figure 6.1a is a plot of the analytical solution (solid curves) and numerical solution (points) to Burgers' equation [equation (6.18)] (this plot was not generated by the coding in subroutine PRINT of Program 6.4c, but, rather, calls to a continuous plotter were subsequently added). The location of the adaptive grid points is indicated at the bottom of the figure; the concentration of the grid points where the solution changes rapidly (in x) is evident.

Figure 6.1b is a plot of the number of ODEs vs. T , which starts at 41, and increases and decreases at times when the solution requires more or less grid points to resolve the solution according to the second derivative criterion. The second derivative criterion is plotted in Figure 6.1c vs. T to indicate the variation in the criterion resulting in the addition and deletion of adaptive grid points.

The effectiveness of the adaptive grid is evident from Figure 6.1a in the sense that the use of a fixed grid would require substantially more grid points (and therefore ODEs) to resolve the solution with equal accuracy. Generally this has been the experience, i.e., adaptive grids are more complex than fixed grids to code, including a certain amount of "tuning" of the grid parameters, but they are computationally more efficient for comput-

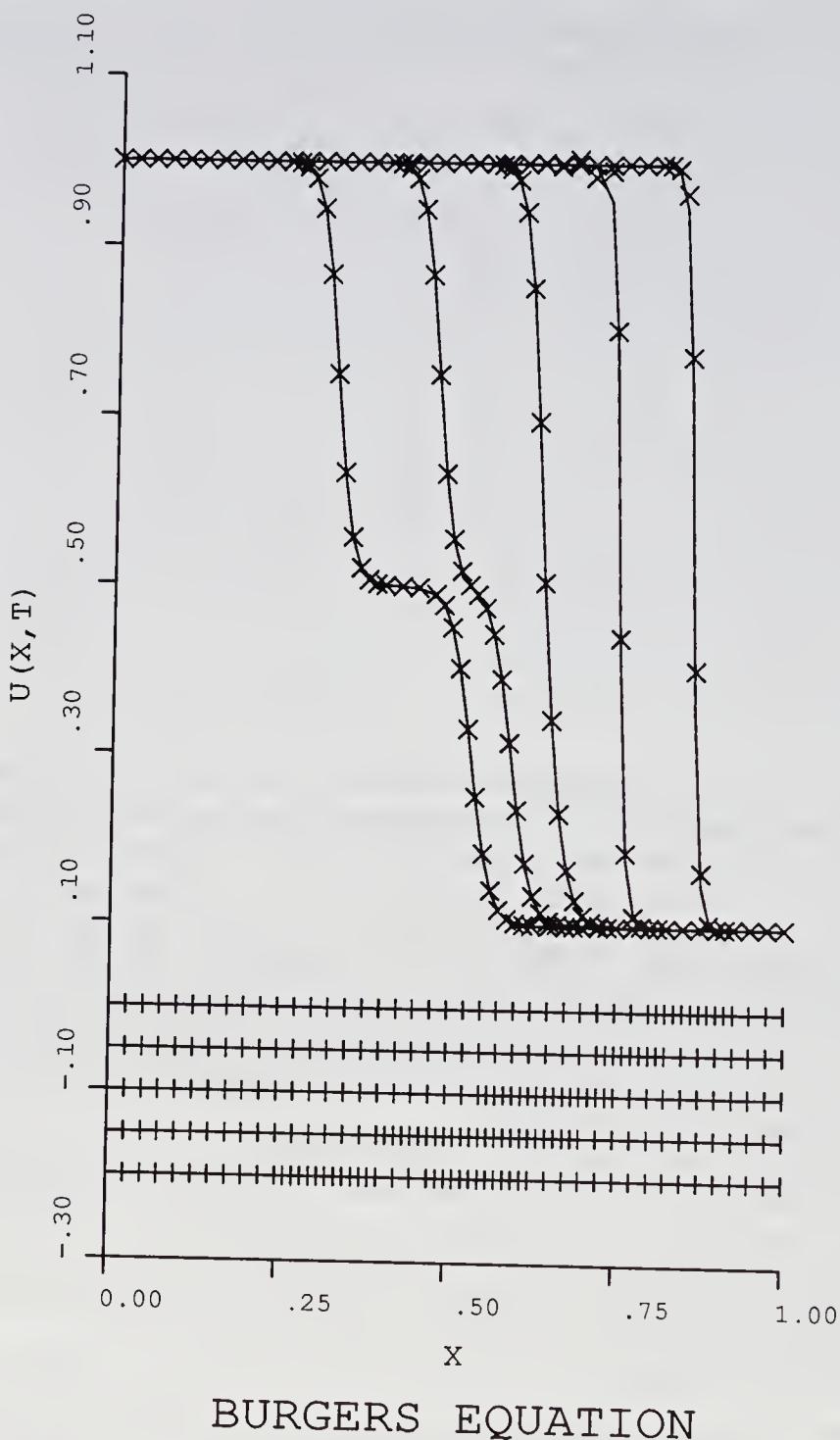


Figure 6.1a Numerical and analytical solutions from Program 6.4.

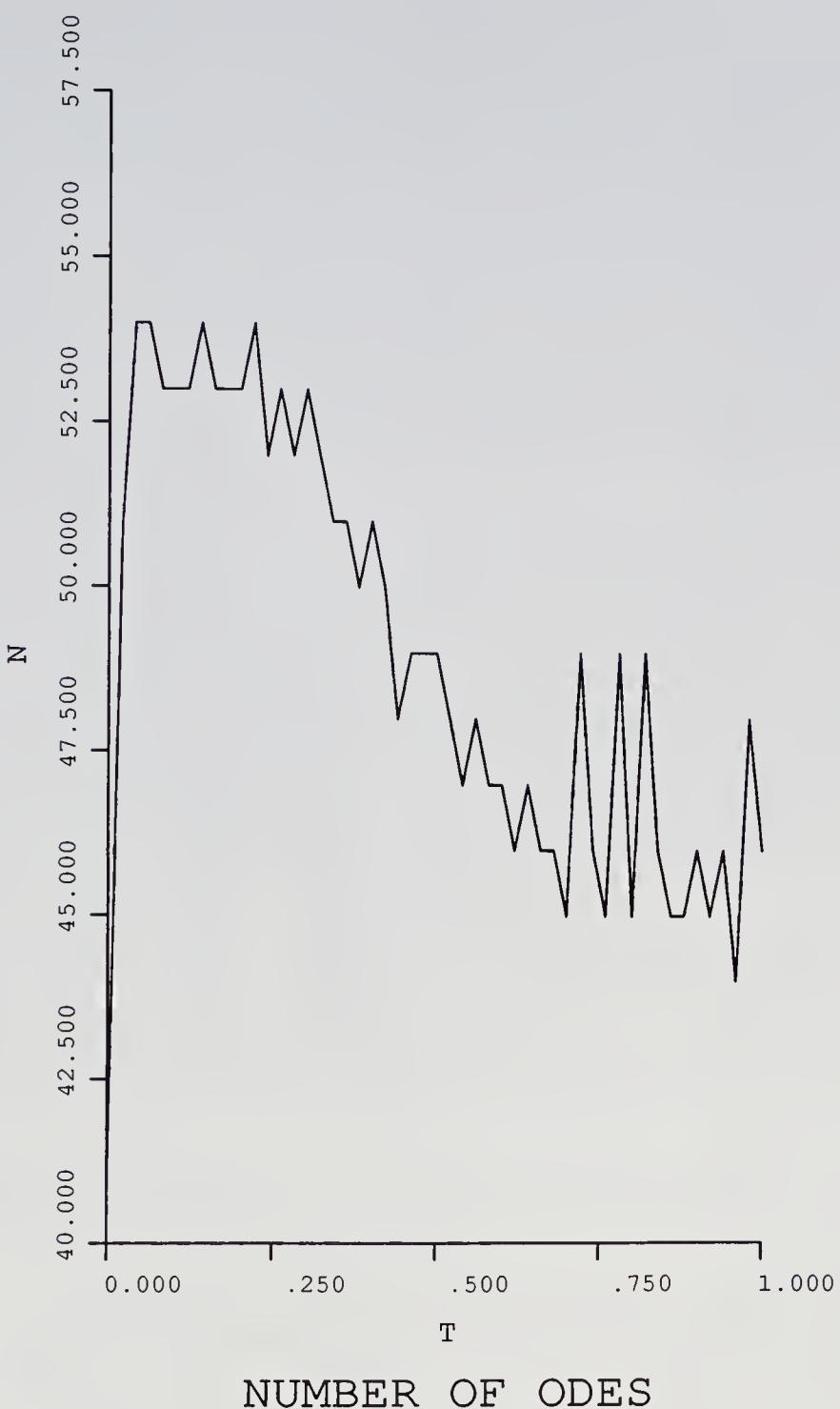


Figure 6.1b Number of adaptive grid points from Program 6.4.

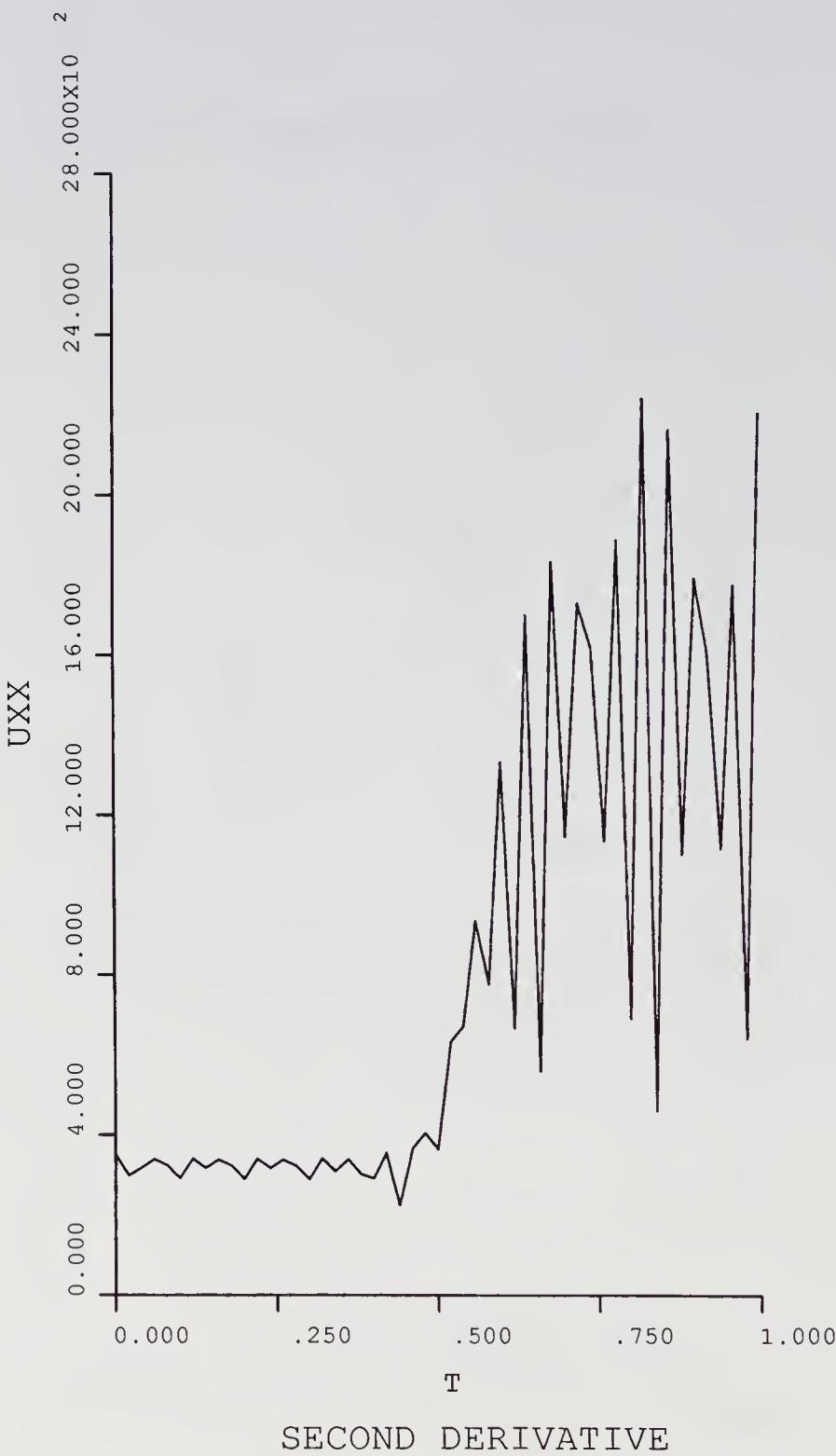


Figure 6.1c Second derivative of the solution from Program 6.4.

ing solutions with sharp spatial variations, and in some cases, provide numerical solutions that would be unattainable with fixed-grid methods.

Finally, as the preceding discussion indicated, particularly for subroutine PRINT, a new initial value ODE problem is defined each time subroutine ANUGB1 is called to update the adaptive grid. This can cause problems with ODE integrators unless some precautions are taken to ensure that the ODE integrator starts correctly each time the new ODE problem is initiated. In the present case, since LSODE was used, we set ISTATE=1 before each call to LSODE, as indicated in Program 6.4f (which initializes LSODE). This results in some inefficiency since by using ISTATE=1, LSODE is reset to a first-order method (the implicit Euler method), and therefore the improved efficiency of the higher-order BDF methods is lost, at least at the beginning of each new solution after the adaptive grid is redefined.

In addition to the present approach of adding and deleting grid points on a basic, unchanging grid, other approaches to adaptive grids could be considered. For example, the total number of grid points could be fixed, but the location of the points could be changed as the solution evolves. What would be needed in this case is some type of “equation of motion” for the grid points that moves them to locations where they will be effective in resolving the spatial variation of the solution; this is actually the approach in the method of *moving finite elements*. Care must be taken to ensure that the movement of the points does not cause them to cross or interact in other unfavorable ways. Also, different criteria (other than the second derivative) can be used to move the points.

Thus, adaptive grids are possible in many forms, and in fact, this is a very active area of research [Flaherty et al. (6.2)]. In the case of two- and three-dimensional problems, *domain decomposition* to regrid the spatial domain may be a central consideration. The efficiency of the adaptive grid approach can be quite high for multidimensional problems because of the

Program 6.4f Portion of the Main Program That Calls LSODE in the Adaptive Grid Solution of Burgers' Equation

```
C... CALL SUBROUTINE LSODE TO COVER ONE PRINT INTERVAL. NOTE THAT LSODE
C... IS INITIALIZED EACH TIME IT IS CALLED (ISTATE = 1) SO THAT THE ORDER
C... OF THE INTEGRATOR IS THEREFORE RESET TO 1 (THE IMPLICIT EULER
C... METHOD). THIS OCCURS AT THE END OF EACH PRINT INTERVAL BECAUSE
C... THE ADAPTIVE GRID IS REDEFINED BY A CALL TO SUBROUTINE ANUGB1 IN
C... SUBROUTINE PRINT. THIS REDEFINTION OF THE ADAPTIVE GRID, IN
C... EFFECT, ALSO DEFINES A NEW ODE PROBLEM (THE NUMBER OF ODES, NEQN,
C... CAN CHANGE THROUGH COMMON/N/), AND THEREFORE LSODE MUST BE INI-
C... TIALIZED FOR THE NEW ODE PROBLEM
4      ISTATE=1
      TOUT=TV+TP
      CALL LSODE(FCN,NEQN,YV,TV,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,
1           IOPT,RWORK,LRW,IWORK,LIW,JAC,MF)
```

rapid increase in the number of grid points with increasing dimensions for fixed-grid methods. Success in the adaptive-grid resolution of the solutions of higher-dimensional problems that have very complex spatial variations has been quite impressive, and continuing advances are expected as the power of computers continues to increase.

6.6 Summary and Conclusions

In the preceding chapters, we have attempted to give the essential ideas behind the NUMOL of PDEs, principally through examples based on readily available Fortran software. The preceding discussion contains very little mathematical theory, but rather, represents the approach of a scientist or engineer who is interested primarily in producing a numerical solution to a PDE problem, and is willing to experiment without requiring deeper insights into the underlying mathematics. This is not entirely satisfactory, since a better understanding of the mathematical aspects of the NUMOL can provide more effective use of the method, and help to avoid undetected errors.

We hope that the preceding discussion illustrates the flexibility of the NUMOL for the solution of a broad spectrum of PDEs. In fact, the NUMOL is limited essentially only by the imagination of the analyst to put together different approximations and the associated software. Much of the appeal and acceptance of the NUMOL is due to recent progress in numerical methods for ODEs, and as this field advances further, along with new approaches to the approximation of spatial derivatives, we expect that the NUMOL will continue to contribute new and ingenious approaches to the solution of PDEs.

References

- (6.1) Byrne, G. D., and A. C. Hindmarsh, "Stiff ODE Solvers: A Review of Current and Coming Attractions," *J. Comput. Phys.*, **70**, 1–62 (1987).
- (6.2) Flaherty, J. E., P. J. Paslow, M. S. Shephard, and J. D. Vasilakis, *Adaptive Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1988.

Problems

- (6.1) Execute Program 6.1 for the 20 cases and summarize your conclusions about the plotted output [$\log_{10}(\text{abs(error)})$ vs. $\log_{10}(\Delta x)$], particularly the slopes of the plots.

- (6.2) Consider again the nature of the solution to the one-dimensional Burgers equation [equation (3.18)] as discussed at the beginning of Section 6.4. If the initial condition for equation (3.18) is a positive, monotonically *increasing* function of x , what will happen to this initial condition with increasing time t ?

The Laplacian Operator in Various Coordinate Systems

One approach to the derivation of PDE models, sometimes termed *distributed models* or *distributed parameter models* (because of the spatial distribution of the model dependent variables), is to express one or more basic principles for an incremental volume, then take the limit as the dimensions of this volume go to zero, to arrive at the final PDEs. This approach was illustrated in Sections 1.2, 2.5, and 3.5. An alternative approach is to start with general PDEs, then simplify according to the requirements of the particular problem. Such general sets of PDEs are available in various scientific and engineering disciplines, e.g., Maxwell's equations in electromagnetic field theory, the Navier–Stokes equations in fluid dynamics, and the equations of change in transport theory.

The discussion of these general sets of equations is outside the scope of this book. However, we illustrate the idea with an example. Fourier's second law can be written generally as

$$\rho C_p \frac{\partial T}{\partial t} = k \nabla^2 T \quad (\text{A.1.1})$$

where ∇^2 is the *Laplacian operator*, with the following definitions in Cartesian, cylindrical, and spherical coordinates

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$$

$$\nabla^2 = \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2}{\partial \theta^2} + \frac{\partial^2}{\partial z^2}$$

$$\nabla^2 = \frac{1}{r^2} \frac{\partial}{\partial r} \left(r^2 \frac{\partial r}{\partial r} \right) + \frac{1}{r^2 \sin^2 \phi} \frac{\partial^2}{\partial \theta^2} + \frac{1}{r^2 \sin \phi} \frac{\partial}{\partial \phi} \left(\sin \phi \frac{\partial}{\partial \phi} \right)$$

If we start the analysis of a problem with Fourier's second law in cylindrical

coordinates, then

$$\rho C_p \frac{\partial T}{\partial t} = k \nabla^2 T = k \left[\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial T}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 T}{\partial \theta^2} + \frac{\partial^2 T}{\partial z^2} \right] \quad (\text{A.1.2})$$

If the problem system has no variation of T with θ and z , equation (A.1.2) reduces to

$$\rho C_p \frac{\partial T}{\partial t} = k \nabla^2 T = k \left(\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial T}{\partial r} \right) \right) = k \left(\frac{\partial^2 T}{\partial r^2} + \frac{1}{r} \frac{\partial T}{\partial r} \right) \quad (\text{A.1.3})$$

which is equation (2.10).

Problem _____

- (A.1.1)** Derive equation (A.1.2) starting with an incremental volume in r , θ , and z .

Spatial Differentiation

Routines

Several spatial differentiation routines, with names beginning with DSS, e.g., DSS002, DSS020, have been discussed in some detail. They are part of a library of routines for which a brief description follows:

Routine	Description
DSS002	One-dimensional, three-point centered approximations for first-order derivatives
DSS004	One-dimensional, five-point centered approximations for first-order derivatives
DSS006	One-dimensional, seven-point centered approximations for first-order derivatives
DSS008	One-dimensional, nine-point centered approximations for first-order derivatives
DSS010	One-dimensional, 11-point centered approximations for first-order derivatives
DSS012	One-dimensional, two-point upwind approximations for first-order derivatives
DSS014	One-dimensional, three-point upwind approximations for first-order derivatives
DSS016	One-dimensional, five-point upwind approximations for first-order derivatives
DSS018	One-dimensional, four-point biased upwind approximations for first-order derivatives
DSS020	One-dimensional, five-point biased upwind approximations for first-order derivatives
DSS022	Two-dimensional, three-point centered approximations for first-order derivatives
DSS024	Two-dimensional, five-point centered approximations for first-order derivatives
DSS026	Two-dimensional, seven-point centered approximations for first-order derivatives

DSS028	Two-dimensional, nine-point centered approximations for first-order derivatives
DSS030	Two-dimensional, 11-point centered approximations for first-order derivatives
DSS032	One-dimensional, five-point centered and biased upwind approximation for first-order derivatives on a user-defined nonuniform grid
DSS034	Two-dimensional, five-point centered and biased upwind approximations for first-order derivatives
DSS036	Three-dimensional, five-point centered and biased upwind approximations for first-order derivatives
DSS038	One-dimensional, cubic spline differentiator for first-order derivatives on a user-defined nonuniform grid
DSS040	One-dimensional, cubic spline differentiator for first- and second-order derivatives on a user-defined non-uniform grid
DSS042	One-dimensional, three-point centered approximations for second-order derivatives with Dirichlet and Neumann boundary conditions
DSS044	One-dimensional, five-point centered approximations for second-order derivatives with Dirichlet and Neumann boundary conditions
DSS046	One-dimensional, orthogonal collocation approximations for first-order derivatives with user-specified finite element collocation and user-selected second-, fourth-, . . . , tenth-order orthogonal polynomials
DSS048	Two-dimensional, orthogonal collocation approximations for first-order derivatives with user-specified finite element collocation and user-selected second-, fourth-, . . . , tenth-order orthogonal polynomials
DSS050	Three dimensional, orthogonal collocation approximations for first-order derivatives with user-specified finite element collocation and user selected second-, fourth-, . . . , tenth-order orthogonal polynomials

The increasing numbers of these differentiators, e.g., DSS002, then DSS004, reflect their historical development. Also, there is some redundancy, since as we gained experience in writing these differentiators (in particular, new approximations, better ways to organize and code the subroutines), we repeated some of the capabilities of earlier routines. For example, DSS034 is a direct replacement of DSS024 since they are both for five-point centered approximations applied over two-dimensional domains.

but DSS034 also has five-point biased upwind approximations; further, the coding of the centered approximations of DSS034 is significantly simpler than in DSS024 because of calls to DSS004 and DSS020. All of these routines are available from the author, with example applications, in single- and double-precision Fortran 77 formats.

Library of ODE and PDE Applications

Physics

- 1 Solution of the steady-state and transient Sine–Gordon equations by quasilinearization and the method of lines
- 2 1 + 1 Sine–Gordon equation
- 3 Liouville’s equation
- 4 Transient response of a resonant *RCL* (resistance, capacitance, inductance) circuit
- 5 Dynamic response of a manometer
- 6 Simulation of a spacecraft modeled as a triangular truss
- 7 Dynamics of a discrete-distributed mass-spring system
- 8 Numerical integration of a Fokker–Planck equation (van der Meer antiproton accumulator equation)
- 9 Einstein equation for the scale factor of the universe
- 10 Time-dependent Schrodinger equation in two dimensions
- 11 Dynamics of a rocket
- 12 Euler integration of the differential equations for the dynamics of a rocket
- 13 Dynamic analysis by difference and differential equations
- 14 Integration of the Gibbs–Duhem equation
- 15 Integration of Bessel’s equation of order zero and one
- 16 Bessel’s equation of order zero (second approach)
- 17 Numerical integration of the normal probability distribution function
- 18 Integration of the solar-radiation spectrum
- 19 Coupled linear harmonic oscillators
- 20 Shuttle launch simulation
- 21 Settling of a particle in a fluid
- 22 A good test problem in partial differential equations for the numerical method of lines
- 23 Laplace transform solution of simultaneous ODEs

- 24 Davidenko's method (a differential form of Newton's method)
- 25 Flow through porous media
- 26 Dynamics of a double pendulum
- 27 One-dimensional, time-dependent Maxwell's equations
- 28 Damping in a second-order linear system
- 29 Effect of damping on a mass-spring system
- 30 Eigenvalue and transient response analysis of a resistance-inductance circuit
- 31 Linear stability analysis of a system of nonlinear ordinary differential equations
- 32 One-dimensional, time-dependent, Maxwell's equations—explicit programming of the spatial derivatives.

Mathematics

- 1 Alternate implementation of time-dependent Dirichlet boundary conditions in the numerical method of lines
- 2 The quasi-steady-state approximation
- 3 Multiple steady states of a nonlinear distributed (PDE) system
- 4 Frequency of the Jacobian matrix updating in Davidenko's method
- 5 Automatic updating of the Jacobian matrix in Davidenko's method
- 6 A new finite difference approximation of second-order derivatives with Neumann boundary conditions
- 7 Ordering of operations in the numerical method of lines
- 8 Investigation of the order of the Euler and modified Euler methods
- 9 One-dimensional parabolic partial differential equation with a time-dependent nonhomogeneous term
- 10 Euler integration applied to integrals
- 11 Comparison of the analytical and numerical solutions and eigenvalues of a second-order, nonoscillatory ordinary differential equation
- 12 Comparison of the analytical and numerical solutions and eigenvalues of a second-order, oscillatory ordinary differential equation
- 13 Testing energy conservation of one-dimensional, finite difference operators
- 14 A comparison of stagewise differentiation and direct differentiation in the numerical method of lines—linear case

- 15 A comparison of stagewise differentiation and direct differentiation in the numerical method of lines—nonlinear case
- 16 Explicit and implicit ordinary differential equations
- 17 Even when the analytical solution to an ODE is available, the numerical solution may be much easier to compute and use
- 18 Effect of ODE stiffness on explicit integrator performance
- 19 Linear stability analysis of a nonlinear, second-order ordinary differential equation
- 20 PDE application executed under DASSL with specification of the ODE Jacobian matrix bandwidth
- 21 One-dimensional, nonlinear, parabolic PDE by second- and fourth-order formulas for second-order spatial derivatives
- 22 The logistic equation
- 23 A differential form of the Henon “strange attractor” equations
- 24 An analysis of the nonlinear structure of the Henon “strange attractor” equations
- 25 Crisis transition in the chaotic Lorenz equations
- 26 Intermittency in the chaotic Lorenz equations
- 27 One-dimensional Burgers equation with Dirichlet and Neumann boundary conditions, illustrating the numerical method of lines integration of one-dimensional, elliptic, hyperbolic, and parabolic partial differential equations—explicit time integration
- 28 One-dimensional Burgers equation with Dirichlet and Neumann boundary conditions, illustrating the numerical method of lines integration of one dimensional, elliptic, hyperbolic, and parabolic partial differential equations—implicit (stiff) time integration by LSODE
- 29 Two-dimensional Burgers equation with Dirichlet and Neumann boundary conditions, illustrating the numerical method of lines integration of one-dimensional, elliptic, hyperbolic, and parabolic partial differential equations
- 30 Three-dimensional Burgers equation with Dirichlet and Neumann boundary conditions, illustrating the numerical method of lines integration of one-dimensional, elliptic, hyperbolic, and parabolic partial differential equations
- 31 Comparison of the analytical and numerical solutions for a first-order and second-order ODE system
- 32 A parabolic partial differential equation with inconsistent initial and boundary conditions
- 33 Numerical integration of a partial differential equation over a semiinfinite domain

- 34 An introduction to finite integral transforms
- 35 LSODE solution of two linear ODEs with a stiffness ratio of 10^6 —printing of the numerical and exact solutions
- 36 LSODE solution of two linear ODEs with a stiffness ratio of 10^6 —plotting of the computational statistics
- 37 DASSL solution of two linear ODEs with a stiffness ratio of 10^6 —printing of the numerical and exact solutions
- 38 Krogh's ODE problem—solution of a nonstiff case by Runge-Kutta integration
- 39 Krogh's ODE problem—solution of a stiff case by LSODE
- 40 Bandwidth reduction in the numerical method of lines integration of partial differential equations
- 41 Two-dimensional partial differential equation with a mixed partial derivative
- 42 Effect of discontinuous initial and boundary conditions on the numerical method of lines solution of a two-dimensional, parabolic partial differential equation
- 43 Two ODEs with a stiffness ratio of 10^6 integrated by IMSL routine IVPAG—printing of the numerical and exact solutions
- 44 Two ODEs with a stiffness ratio of 10^6 integrated by IMSL routine IVPAG—printing of the computational statistics
- 45 Solution of two fourth-order PDEs
- 46 Adaptive grid solution of the one-dimensional Burgers equation with BDF time integration via LSODE
- 47 DIFFPACK—a collection of problems illustrating the NUMOL solution of initial- and boundary-value ODEs, and elliptic, hyperbolic, and parabolic PDEs in one, two, and three dimensions

Biochemical, Biomedical, and Environmental Systems

- 1 Solution of steady-state equations of the human cardiovascular mechanics by Davidenko's method
- 2 Oxygen transfer in a red blood cell
- 3 Biosynthesis of the antibiotic erythromycin
- 4 Glucose tolerance test
- 5 Solution of dynamic equations of human cardiovascular mechanics
- 6 River pollution model

- 7 Diffusion and reaction in an immobilized enzyme catalyst
- 8 Kinetics of a biochemical reaction when two species compete for the same substrate
- 9 An analysis of population growth in Asia
- 10 Forrester World 2 model
- 11 Boyd modification of the Forrester World 2 model
- 12 Two-sector Globe 6 model
- 13 Nomenclature for the Globe 6 model
- 14 A low-order, nonlinear, generic model for the dynamics of HIV transmission

Separation Systems

- 1 Dynamics of a binary batch distillation column
- 2 Dynamic simulation of a deisobutanizer
- 3 Diffusion and nonlinear adsorption in a pore
- 4 Dynamics of a countercurrent mixer–settler system
- 5 Dynamics of a decanter
- 6 Dynamic model of a three-stage mixer–settler system
- 7 Multicomponent chromatographic separation
- 8 Startup and control of a plate absorber
- 9 Packed-column absorber response
- 10 Multicomponent chromatographic separation with cosine pulse input
- 11 Performance of a moving-bed adsorption column
- 12 PI (proportional-integral) control of a binary batch distillation column
- 13 PI control of a double-effect evaporator
- 14 Jacobian matrix map and temporal eigenvalues of the model for a batch distillation column
- 15 Nonlinear diffusion
- 16 Proportional-integral control of a humidifier
- 17 Evaporation from a tank
- 18 Countercurrent liquid–liquid extractor for a partially miscible system
- 19 Nonequilibrium model for fixed-bed, multicomponent, adiabatic adsorption (chromatography)
- 20 Diffusion in a porous solid—linear case
- 21 Diffusion in a porous solid—nonlinear case
- 22 Control of a triple-effect evaporator

- 23 Solution of the Beattie–Bridgeman equation of state via the Davidenko algorithm
- 24 Solution of the Beattie–Bridgeman equation of state via minimum squared error
- 25 Batch separation of *n*-heptane and *n*-octane without automatic control
- 26 Batch separation of *n*-heptane and *n*-octane with automatic control
- 27 Dynamic simulation of a packed-column absorber

Kinetics and Reactor Models

- 1 Nonisothermal catalytic tubular reactor
- 2 Dynamic two-dimensional tubular reactor
- 3 Dynamics of an ethylene oxide reactor
- 4 Ethane pyrolysis in a tubular reactor
- 5 Hydrodealkylation of toluene to benzene
- 6 First-order kinetics of consecutive, irreversible reactions
- 7 Decay of Plutonium 239
- 8 Three continuous-stirred tank reactors (CSTRs) with PI control
- 9 Three CSTRs with proportional-integral control
- 10 Transient response of two CSTRs with variable holdup
- 11 Polymerization with two initiators
- 12 PI control of a CSTR with competing reactions
- 13 Oxidation of naphthalene to phthalic anhydride: Model 1—resistance to heat and mass transfer at the catalyst surface is neglected
- 14 Oxidation of naphthalene to phthalic anhydride: Model 2—resistance to heat and mass transfer at the catalyst surface is included
- 15 CSTR with proportional temperature control
- 16 Three-component CSTR with temperature control
- 17 Adsorption with chemical reaction in a tray column
- 18 Nonlinear kinetics of consecutive, irreversible reactions
- 19 Effect of the axial Peclet number on the solution of convective-diffusion (hyperbolic–parabolic) partial differential equations
- 20 Recycle in dynamic simulation
- 21 Hydrolysis of acetic anhydride
- 22 Moving-boundary problem

- 23 Dynamics of a tubular reactor with axial dispersion
- 24 Kinetics of catalytic cracking selectivity
- 25 Conduction, diffusion, and reaction in a catalyst particle
- 26 Temperature profiles in a tubular reactor
- 27 Relative effects of surface mass-transfer resistance, internal diffusion, and internal reaction on the performance of a spherical catalyst particle
- 28 Solution of a second-order, nonlinear, two-point boundary-value ordinary differential equation by quasilinearization—application in chemical kinetics
- 29 Esterification of acetic acid and ethyl alcohol
- 30 Semibatch production of hexamethylenetetramine

Heat Transfer

- 1 Method of lines integration of boundary-value ODEs
- 2 Solution of the one-dimensional heat-conduction equation with Neumann boundary conditions and boundary conditions of the third type
- 3 Sizing of a vapor-condensing heat exchanger
- 4 Laminar-flow fluid temperature profiles in a cylindrical tube
- 5 One-dimensional heat conduction with plotted temperature profiles
- 6 Simplified transient analysis of a LOCA (loss of coolant accident) for Three Mile Island reactor no. 2
- 7 Air temperature within a large city
- 8 Dynamics of a nonisothermal holding tank
- 9 Integration of the Planck distribution law over the visible spectrum—first approach
- 10 Integration of the Planck distribution law over the visible spectrum—second approach
- 11 Regenerator simulated by CSTRs in series and centered differences
- 12 Dynamics of a double-pipe heat exchanger
- 13 Thermal bonding of solids
- 14 Temperature rise from mechanical agitation of a fluid
- 15 Dynamics of a regenerator with plotting of the spatial profiles
- 16 Temperature distribution in a nuclear fuel rod assembly
- 17 Dynamics of a three-pass shell-and-tube heat exchanger
- 18 Improvement in the performance of a multipass heat exchanger from adding another pass

- 19 PI control of a three-pass shell-and-tube heat exchanger
- 20 Parameter estimation in heat-transfer systems
- 21 Heat conduction in a sphere
- 22 Convective cooling of a moving polymer sheet
- 23 Dynamics of a cross-flow heat exchanger
- 24 Temperature control of a nuclear fuel rod assembly
- 25 Control of a triple-effect evaporator
- 26 Dynamics of a two-stage heat-exchanger system
- 27 PI control of a three-pass shell-and-tube heat exchanger (case 2)
- 28 Temperature control of a nuclear fuel rod
- 29 Validation of dynamic models through conservation principles
- 30 A comparison of two-point upwind and five-point biased upwind approximations in the numerical method of lines integration of first-order hyperbolic partial differential equations
- 31 Effect of heat loss from a heat exchanger
- 32 Water cooling of sulfuric acid
- 33 Cooling of sulfuric acid with limited cooling water
- 34 Heat conduction in a finite cylinder
- 35 Heat conduction with internal heat generation
- 36 Heating a liquid in a tube with axial conduction

Fluid Flow

- 1 Dynamics of liquid transfer from a tank car
- 2 Dynamics of a holding tank with two long lines and a pump
- 3 Drainage from a vertical cylindrical tank
- 4 Drainage from a horizontal cylindrical tank
- 5 Steady-state simulation via dynamic analysis
- 6 Liquid transfer between tanks via a centrifugal pump
- 7 Isentropic discharge of a perfect gas from a duct—first approach
- 8 Isentropic discharge of a perfect gas from a duct—second approach
- 9 Dynamics of a gravity flow tank
- 10 Transient response of a first-order hydraulic system
- 11 Transient response of a pneumatic system with a centrifugal compressor
- 12 Dynamics of two tanks connected by a long line with feed and discharge lines
- 13 Transients in a hydraulic system with a long line
- 14 Advection equation with variable velocity

- 15 Numerical instability due to incorrect modeling of flow reversals
- 16 Nonlinear advection equation
- 17 Dynamics of two tanks connected by a long line
- 18 Dynamics of two tanks connected by a long line with proportional control of the liquid height in the second tank
- 19 Liquid-level control of two cascaded tanks
- 20 Pressures and flows in a pump–pipe–tank network by the multidimensional Davidenko algorithm (differential form of the Newton–Raphson method)
- 21 Aquifer simulation in Cartesian coordinates
- 22 Euler integration of three differential equations for the dynamics of two tanks connected by a long line
- 23 Euler integration of the differential equation for draining of a tank
- 24 Hydraulic system dynamics
- 25 Euler integration of the differential equation for a manometer
- 26 Eigenvalue analysis of a fluid-flow system with a fast momentum balance
- 27 Least-squares analysis of friction factor–Reynolds number data—case 1
- 28 Least-squares analysis of friction factor–Reynolds number data—case 2
- 29 Time to drain a tank through a vertical pipe
- 30 Liquid-level dynamics of two tanks with a long discharge line

Automatic Control

- 1 Simulation of a linear system with a time delay
- 2 PI control of the liquid levels in two interacting holding tanks
- 3 Comparative evaluation of controlling the inlet vs. outlet flows of a dynamic unit
- 4 Sinusoidal response of a nonlinear system
- 5 Ultimate control of a feedback control system
- 6 Effect of saturation on control system performance
- 7 Eigenvalue stability analysis of feedback control systems
- 8 Effect of time delays on the performance of feedback control systems
- 9 Root locus plotting for an n th-order system
- 10 Ultimate gain and tuning of the PI control of a plate absorber
- 11 Direct-frequency response testing

- 12 Root locus stability analysis of a fourth-order system
- 13 Frequency response stability analysis of a fourth-order system
- 14 Offset–stability tradeoff in the proportional control of dynamic systems
- 15 Root locus stability analysis
- 16 Feedback control with sensor dynamics
- 17 Damping in a second-order linear system
- 18 Simulation and root locus analysis of a third-order system with PI control

Index

- Adaptive grids, 287–302
advantages, 297–302
applied to Burgers' equation, 287–302
criterion for grid point insertion, 291, 300
efficiency, 297–302
implemented in ANUGB1, 293–295
insertion and deletion of grid points, 290–302
spatial differentiation by cubic splines, 290–293
variation in the number of ODEs, 290–299
- Advection equation, 123, 217–218
Burgers' equation as a nonlinear variant, 250
exact solution, 123–141
NUMOL approximations, 209–213
eigenvalues, 209–213
stability of centered approximation, 209–210
stability of upwind approximation, 211–212
propagation of discontinuities, 124–125
solution by upwind approximation, 130–134
- Algorithms, 9
see also Integration algorithm
ANUGB1 used in adaptive grid, 292–295
Arrhenius temperature dependence, 64
Auxiliary conditions, 6
required number, 6
- Backward differentiation formulas *see* BDF methods
Banded matrices, 150, 165
Bessel functions, 62
BDF methods, *see also* LSODE; LSODES; LSODI; DASSL
coefficients for various orders, 161
computational statistics, 166
convergence of the corrector, 165–166
- efficiency for stiff ODEs, 166
exploiting the Jacobian matrix structure, 165–166, 178
general equation, 160
implicit structure, 160–161
Newton's method for linear algebra, 164–165
number of function evaluations, 166
numerical evaluation of Jacobian matrix, 166, 178
orders of stepping formulas, 161
solution of corrector equation, 164–165
stability, 162–163
stiffly stable formulas, 162
updating the Jacobian matrix, 165–166
variable-order implementation, 161
- Biased upwind approximations, 135–141
- Boundary conditions, 6
Dirichlet, *see* Dirichlet boundary conditions
general form, 247–248
inconsistency with initial conditions, 49–54, 74–75, 249–250
maximum order, 7
Neumann, *see* Neumann boundary conditions
nonlinear, 54–55
programming in NUMOL, 12
required number, 6, 75, 247
of third type, 7, 91
- Boundary-value independent variables, 6
Boundary-value ODEs, 248
- Burgers' equation
Dirichlet, Neumann, and third type boundary conditions, 250–262
in one dimension, 250–262
exact solution, 250–251
in two dimensions, 262–273
exact solution, 262
in three dimensions, 273–287
exact solution, 273

- Cartesian coordinates, 5, 247
 Laplacian operator, 305
 Central finite differences, 99
 applied to advection equation, 128–129
 CFL condition, *see* Courant–Friedrichs–Lewy condition
 Classification of PDEs, 250
COMMON
 use in NUMOL code, 31–41, 45–48
 structure for ODEs, 32–33
 Computational stencil, 100–101
 Conservation of energy, 3, 217–218
 Conservation principles, 3, 215–218
 Control system, 228–229, 245
 gain, 228
 integral time, 228
 NUMOL programming, 228–229
 proportional-integral control law, 228, 245
 set point, 228, 245
 Convective diffusion PDEs, 145
 Coordinate systems for PDEs, 60–61, 247
 Courant–Friedrichs–Lewy condition, 212
 Cubic spline differentiation, 122
 in adaptive grids, 290–293
 by DSS038 and DSS040, 122
 by NCSPLE, 292–293
 Cylindrical coordinates, 60–63, 247
 Laplacian operator, 305
- DAEs**, 188,191
see also LSODI; DASSL
 specification of initial conditions, 192
DASSL, 191–205, 207
 application to DAEs, 191–192
 application to implicit ODEs, 191–192
 arguments, 192–193
 call, 192
 options, 193–204
 programming DAEs, 192
 specification of initial conditions, 192
 Debugging a NUMOL code, 46
 Difference equations, 153
 characteristic equation, 153
 eigenvalues, 153
 general solution, 153
 superposition of solutions, 154
 Differential-algebraic equations, *see* DAEs
 Differentiation, *see* Spatial differentiation
 Differentiation matrix, 100–101, 108
 weighting coefficients, 100–101
 Dirichlet boundary conditions, 6, 24–31, 91
 Discontinuities
 from Burgers' equation, 288
 Gibbs' phenomenon, 53, 62
 propagation, 53–54, 74–75, 124
 Distributed system, 215, 305
 Domain decomposition, 301
 Double precision Fortran, 18
DSS002, 11
 programming, 100–102
DSS004, 12, 15, 27, 34, 86–90, 248, 255–258, 272, 286
 programming, 109–110
DSS012, 130–134
 programming, 130
DSS014, 132–135
 programming, 133–134
DSS020, 135–141, 230, 249, 255–258, 272, 286
 programming, 135–136
DSS034, 267–270
 arguments, 272
 for solution of Burgers' equation, 267–270
 for two-dimensional PDEs, 267–270
DSS036, 273–287
 arguments, 285–286
 for solution of Burgers' equation, 273–287
 for three-dimensional PDEs, 273–287
DSS044, 115–122, 257–258
 Dirichlet and Neumann boundary conditions, 115
 programming, 115–117
- EIGEN**, 236–244
 call, 234
 Eigenvalue problem
 characteristic equation, 147
 definition, 149
 of ODEs, 147–150
 and stability analysis, 148
 Eigenvalues, *see* Difference equations; EIGEN; Eigenvalue problem; Jacobian matrix; JMAP; ODEs; Spectrum; Stiffness

- Eigenvectors, 148
EISPACK, 243
 Elliptic PDEs, 60, 75–90
 classification, 248
 conversion to parabolic PDEs, 81
 Error tolerances
 absolute, 24
 in ODE integration, 24, 36
 relative, 24
 Errors in NUMOL solutions
 analysis, 19–24
 decay, 17
 estimates, 19
 exact errors, 16–17, 19, 28–31, 41
 in PDE solutions, 9
 order, 21, 30
 variation with number of grid points,
 20–21, 29–31
 Explicit Euler method, *see* Euler method
 Explicit ODE integration
 computational requirements, 158
 limited stability region, 152–158
 limitations for stiff systems, 152–158
 Euler method
 programming, 43
 stability criterion, 154, 159
 stability of explicit method, 152–158
 stability of implicit method, 158–160
 Exact solutions, 8
 calculation, 15–16, 35
- False transients
 required initial condition, 76
 to solve elliptic PDEs, 76–90, 247
 Feedback control, *see* Control system
 Fictitious points, 99–100, 131
 Finite differences, 10
 Finite elements, 10, 187
 moving, 301
 Finite volumes, 10
 First-order hyperbolic PDEs, *see* Hyperbolic PDEs
 Fourier number, 209, 244
 Fourier's first law, 5
 Fourier's second law, 5
 in Cartesian coordinates, 5
 in cylindrical coordinates, 60–63
 from Laplacian operator, 305–306
 NUMOL approximations, 213–215
- eigenvalues, 214
 stability of centered approximation,
 214–215
 stability criterion, 214
 in spherical coordinates, 63–69
 in two dimensions, 76
 Fourth-order differentiation formulas
 for first-order derivatives, 105–111
 for second-order derivatives, 111–122
 Front-sharpening characteristic, of Burgers' equation, 288
 Full matrix, 150, *see also* LSODE
- Gauss row reduction, 165
 Geometric classification of PDEs, 60,
 246–250
- Heat conduction, 3
 Heat conduction equation, *see* Fourier's second law
 Heaviside function, 124
 Helmholtz's equation, 75–90, 248
 Humidification, 215–244
 Hyperbolic PDEs, 60, 70–75
 classification, 248–249
 first-order, 91, 96, 122–141, 248–249
 second-order, 70–75, 96, 249
- Implicit Euler method, *see* Euler method
 Implicit ODE integration
 computational requirements, 160
 limitations for stiff systems, 158–167
 stability region, 158–167
 Implicit ODEs, 187–188
 fully implicit, 191–192
 linearly implicit, 187–188
 Independent variables, 2
 number in PDEs, 2
 Indeterminate forms
 evaluation by l'Hospital's rule, 60, 66
 Initial conditions, 6
 general form, 247–248
 required number, 6, 70, 247
 inconsistency with boundary conditions,
 49–54, 74–75, 249–250

- Initial-value independent variables, 6
 final value, 17
- Insulated boundary conditions, 7
- Integration algorithm
 accuracy, 23, 146, 156
 stability, 22, 146, 156
- Jacobian matrix
 eigenvalues, 151–152
 calculation, 215–244
 from EIGEN, 234–236
 mapping, 179, 215–244
 from JMAP, 234–236
 numerical evaluation, 166
 of ODEs, 151, 165
 structure, 165
see also Banded matrices; Sparse matrices
- JMAP, 236–244
 call, 234
- Laplace’s equation, 75–90, 248
- Laplacian operator, 305–306
 in Cartesian coordinates, 305
 in cylindrical coordinates, 305
 in spherical coordinates, 305
- Linear PDEs, 9
 basic properties of solutions, 69
- Linearization of ODEs, 151
- LSODE, 171–191, 206
 used with an adaptive grid, 297
 arguments, 177
 call, 172
 computational statistics, 172–173, 176
 efficiency, 175
 full and banded matrix options, 178
 method flag, 172
 work arrays, 178
- LSODES, 180–187, 207
 arguments, 181–187
 call, 180
 computational statistics, 180–181
 determining sparsity structure, 180
 sparse Jacobian matrix features, 180
- LSODI, 187–191, 207
 application to DAEs, 187–188
 application to implicit ODEs, 187–188
 arguments, 189–191
- call, 188
 definition of coupling matrix, 187–188
- Machine constants, 171
 set in R1MACH and D1MACH, 171
- Mapping Jacobian matrix, 214–244
 by JMAP, 234–236
- Mathematical models, 2
 completeness, 4
- Matrix-free ODE integrators, 204
- Modified Euler method programming, 44
- Moving finite elements, 301
- NCSPLE for adaptive grids, 292–295
- Neumann boundary conditions, 7, 24–31
- Newton’s method
 used in BDF methods, 164–165
 used in implicit integration, 160
- Nonhomogeneous terms, 90
- Nonlinearity
 in ODEs, 150–152
 in PDEs, 54–59, 63–69, 90–95, 215–244
 effect on PDE solutions, 57–59, 69
- Nonuniform grids, 122
 with DSS032, 122
 with NCSPLE, 292–295
- Numerical algorithms, 9
- Numerical diffusion, 132
- Numerical errors, in PDE solutions, 9
see also Errors in NUMOL solutions
- Numerical Jacobian matrix, 166, 178
- Numerical method of lines, *see* NUMOL
- NUMOL (numerical method of lines)
 debugging a code, 46
 example derivative subroutine, 15, 27
 example main program, 12–14, 25–26, 33–39
 general equations, 246–250
 introduction, 10–18
 origin of the name, 42
 structure of the ODEs, 236–244
- ODEs
 applications, 310–319
 in automatic control, 318–319

- in biochemical, biomedical, and environmental systems, 313–314
- in fluid flow, 317–318
- in heat transfer, 316–317
- in kinetics and reactors, 315–316
- in mathematics, 311–313
- in physics, 310–311
- in separation systems, 314–315
- BDF solution, *see* BDF methods
- calculation of eigenvalues, 215–244
- by EIGEN, 234–236
- characteristic equation, 147
- constant coefficient, 146
- data file, 40
- eigenvalue problem, 147–150
- eigenvalue spectrum, 213
- general problem, 31–33
- implicit forms, *see* Implicit ODEs
- Jacobian matrix, 151, 164
- mapping of eigenvalues, 215–244
- by JMAP, 234–236
- matrix-free integrators, 204
- nonlinear, 150–152
- from the NUMOL, 10
- precision requirements, 169–171
- programmed in subroutines INITIAL, DERV, PRINT, 33–35
- RKF45 integrator, 12
- spectrum of eigenvalues, 213–215
- stiffness, 22, 121, 213
- structure of coefficient matrix, 150, 236–244
- structure of Jacobian matrix, 165, 236–244
- time scale, 156, 243
- Operations count, in linear algebra, 165–166
- Ordinary differential equations, *see* ODEs
- Orthogonal collocation, 122
- by DSS046, DSS048, DSS050, 122
- Oscillation in PDE solutions, 110, 129
- Parabolic PDEs**
- classification, 248
 - Fourier's second law, 5, 60–63, 63–69
- Parabolic-hyperbolic PDEs, 91, 145
- classification, 250
 - see also* Burgers' equation
- Partial differential equations, *see* PDEs
- PDEs**
- algebraic approximations, 10
 - analytical solution, 8
 - applications, 310–319
 - in automatic control, 318–319
 - in biochemical, biomedical, and environmental systems, 313–314
 - in fluid flow, 317–318
 - in heat transfer, 316–317
 - in kinetics and reactors, 315–316
 - in mathematics, 311–313
 - in physics, 310–311
 - in separation systems, 314–315
- closed-form solution, 8
- definition, 2
- degree, 54
- evaluation of solutions, 58–59
- exact solution, 8
- geometric classification, 60, 246–250
- linearity, 9, 54
- meaning of a solution, 8
- nonhomogeneous terms, 90
- numerical solution, 9, 16–17
- product solution, 209
- second order in time, 70–75
- separated solution, 209
- solution smoothness, 9, 53
- subscript notation, 7
- variable coefficients, 60
- Poisson's equation, 75–90, 248
- Polynomial approximations, 97–98, 103–104
- in spatial differentiation, 97–98
 - oscillation, 110
- Product solution, 209
- Pulse function, 139–141, 144
- Ramp function, 137–138, 143
- RKF45, 12
- call, 13, 26, 38
 - meaning of name, 22
 - stability limit, 22, 157
- Runge–Kutta methods, *see also* RKF45
- order, 22
 - stability limit, 157
 - stability region, 157
- SDRIV2, 167–172, 206
- automatic switching between stiff and nonstiff options, 168

- call, 167
- root finding, 168–169
- options, 168
- work array, 168
- Second-order differentiation formulas for
 - first-order derivatives, 98–105
- Separated solution, 209
- Shocks from Burgers' equation, 288
- Single precision Fortran, 17–18
- Sparse matrices, 150, 180
 - see also* LSODES
- Spatial derivatives, 6
 - by cubic splines, 122
 - error at boundaries, 104
 - error at interior points, 104
 - of a constant, 101
 - of a cubic function, 103
 - of a linear function, 103
 - of a quadratic function, 103
 - subroutines to calculate, 11, 307–309
- Spectrum of ODE eigenvalues, 213
- Spherical coordinates, 63–69, 247
 - Laplacian operator, 305
- Stability
 - of BDF methods, 162–163
 - defined by eigenvalues, 148–149
 - of nonlinear ODEs, 150–152
 - of ODE integration, 146, 152–167
- Stagewise differentiation, 111, 251–258
- Steady state, approach of solutions, 52–53, 81, 245
- Stefan–Boltzmann law, 59
- Step function, 124
- Stiffness
 - effect of spatial grid spacing, 213–215
 - effect on integration, 156
 - of ODEs, 22
 - precision requirements, 169–171
- ratio, 170
- and separation of eigenvalues, 149, 159
- Stiffly stable BDF methods, 162
- Subscript notation for PDEs, 7
- Superaccuracy, 103
- Superposition of ODE solutions, 148
- Taylor series
 - as basis for ODE integration algorithms, 152
 - as a polynomial approximation, 97–98
 - used to derive differentiation formulas, 105–108, 112–115, 244
- Temporal derivatives
 - calculation, 11
 - integration, 11
- Thermal diffusivity, 5
- Time scales of ODEs, 156, 243
- Unconditional stability, implicit Euler method, 159
- Upwind approximations, 130–134, 211–212
 - see also* Biased upwind approximations
- Variable coefficients in PDEs, 60
- VODE, 204
- von Neumann stability analysis, 209
 - Fourier number, 209, 244
- Wave equation, 70–75, 96, 249
- Weighted residuals, 187
- Weighting coefficients, in differentiation formulas, 100–101











DATE DUE / DATE DE RETOUR

JAN 15 1994

JAN 05 1994

NOV 10 1994

MAR 25 2004

CARR MCLEAN

38-297

TRENT UNIVERSITY



0 1164 0228246 5

0-12-624130-9