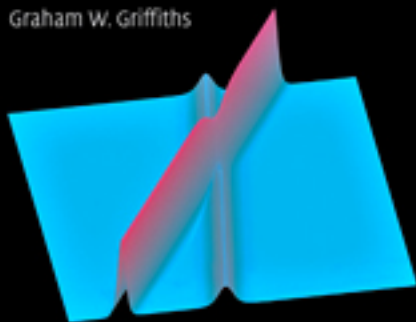


A Compendium of Partial Differential Equation Models

Method of Lines Analysis with Matlab

William E. Schiesser
Graham W. Griffiths



CAMBRIDGE

This page intentionally left blank

A Compendium of Partial Differential Equation Models: Method of Lines Analysis with Matlab

The mathematical modeling of physical and chemical systems is used extensively throughout science, engineering, and applied mathematics. In order to make use of mathematical models, it is necessary to have solutions to the model equations. Generally, this requires numerical methods because of the complexity and number of equations.

A Compendium of Partial Differential Equation Models presents numerical methods and associated computer codes in Matlab for the solution of a spectrum of models expressed as partial differential equations (PDEs), one of the most widely used forms of mathematics in science and engineering. The authors focus on the method of lines (MOL), a well-established numerical procedure for all major classes of PDEs in which the boundary-value partial derivatives are approximated algebraically by finite differences. This reduces the PDEs to ordinary differential equations (ODEs) and thus makes the computer code easy to understand, implement, and modify. Also, the ODEs (via MOL) can be combined with any other ODEs that are part of the model (so that MOL naturally accommodates ODE/PDE models).

This book uniquely includes a detailed, line-by-line discussion of computer code as related to the associated equations of the PDE model.

William E. Schiesser is the Emeritus R. L. McCann Professor of Chemical Engineering and a Professor of Mathematics at Lehigh University. He is also a visiting professor at the University of Pennsylvania and the coauthor of the Cambridge book *Computational Transport Phenomena*.

Graham W. Griffiths is a visiting professor in the School of Engineering and Mathematical Sciences of City University, London, and a principal consultant with Honeywell Systems. He is also a founder of Special Analysis and Simulation Technology Ltd. and has worked extensively in the field of dynamic simulation of chemical processes.

A COMPENDIUM OF PARTIAL DIFFERENTIAL EQUATION MODELS

Method of Lines Analysis
with Matlab

William E. Schiesser

Lehigh University, Bethlehem, PA, USA

Graham W. Griffiths

City University, London, UK



CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS

Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press

The Edinburgh Building, Cambridge CB2 8RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org

Information on this title: www.cambridge.org/9780521519861

© William E. Schiesser and Graham W. Griffiths 2009

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

First published in print format 2009

ISBN-13 978-0-511-50134-0 eBook (Adobe Reader)

ISBN-13 978-0-521-51986-1 hardback

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

To Dolores and Patricia for their patience and support

Contents

<i>Preface</i>	<i>page ix</i>
1 An Introduction to the Method of Lines	1
2 A One-Dimensional, Linear Partial Differential Equation	18
3 Green's Function Analysis	36
4 Two Nonlinear, Variable-Coefficient, Inhomogeneous Partial Differential Equations	70
5 Euler, Navier Stokes, and Burgers Equations	90
6 The Cubic Schrödinger Equation	114
7 The Korteweg–deVries Equation	141
8 The Linear Wave Equation	171
9 Maxwell's Equations	203
10 Elliptic Partial Differential Equations: Laplace's Equation	229
11 Three-Dimensional Partial Differential Equation	261
12 Partial Differential Equation with a Mixed Partial Derivative	291
13 Simultaneous, Nonlinear, Two-Dimensional Partial Differential Equations in Cylindrical Coordinates	306
14 Diffusion Equation in Spherical Coordinates	342
Appendix 1 Partial Differential Equations from Conservation Principles: The Anisotropic Diffusion Equation	381

Appendix 2 Order Conditions for Finite-Difference Approximations 398

**Appendix 3 Analytical Solution of Nonlinear, Traveling Wave
 Partial Differential Equations** 414

Appendix 4 Implementation of Time-Varying Boundary Conditions 420

Appendix 5 The Differentiation in Space Subroutines Library 441

Appendix 6 Animating Simulation Results 445

Index 469

Color plates follow page 474

Preface

In the analysis and the quest for an understanding of a physical system, generally, the formulation and use of a mathematical model that is thought to describe the system is an essential step. That is, a mathematical model is formulated (as a system of equations) that is thought to quantitatively define the interrelationships between phenomena that define the characteristics of the physical system. The mathematical model is usually tested against observations of the physical system, and if the agreement is considered acceptable, the model is then taken as a representation of the physical system, at least until improvements in the observations lead to refinements and extensions of the model. Often the model serves as a guide to new observations. Ideally, this process of refinement of the observations and model leads to improvements of the model and thus enhanced understanding of the physical system.

However, this process of comparing observations with a proposed model is not possible until the model equations are solved to give a solution that is then the basis for the comparison with observations. The solution of the model equations is often a challenge. Typically in science and engineering this involves the integration of systems of ordinary and partial differential equations (ODE/PDEs). The intent of this volume is to assist scientists and engineers in the process of solving differential equation models by explaining some numerical, computer-based methods that have generally been proved to be effective for the solution of a spectrum of ODE/PDE system problems.

For PDE models, we have focused on the method of lines (MOL), a well-established numerical procedure in which the PDE spatial (boundary-value) partial derivatives are approximated algebraically, in our case, by finite differences (FDs). The resulting differential equations have only one independent variable remaining, an initial-value variable, typically time in a physical application. Thus, the MOL approximation replaces a PDE system with an initial-value ODE system. This ODE system is then integrated using a standard routine, which, for the Matlab analysis used in the example applications, is one of the Matlab library integrators. In this way, we can take advantage of the recent progress in ODE numerical integrators.

However, while we have presented our MOL solutions in terms of Matlab code, it is not our intention to provide optimized Matlab code but, rather, to provide code

that will be readily understood and that can be easily converted to other computer languages. This approach has been adopted in view of our experience that there is considerable interest in numerical solutions written in other computer languages such as Fortran, C, C++, and Java. Nevertheless, discussion of specific Matlab proprietary routines is included where this is thought to be of benefit to the reader.

Important variations on the MOL are possible. For example, the PDE spatial derivatives can be approximated by finite elements, finite volumes, weighted residual methods, and spectral methods. All of these approaches have been used and are described in the numerical analysis literature. For our purposes, and to keep the discussion to a reasonable length, we have focused on FDs. Specifically, we provide library routines for FDs of orders 2–10.

Our approach to describing the numerical methods is by example. Each chapter has a common format consisting of:

- An initial statement of the concepts in mathematics and computation discussed in the chapter.
- A statement of the equations to be solved numerically. These equations are a mathematical model that can originate from the analysis of a physical system. However, we have broadened the usual definition of a mathematical model for a physical system to also include equations that test a numerical method or algorithm, and in this sense, they are a model for the algorithm.

Parenthetically, the selected PDE applications include some of the classical (we might even say “famous”) PDEs. For example, we discuss the Euler and Navier Stokes equations of fluid dynamics with the Burgers equation as a special case, the Maxwell equations of electromagnetic field theory with the wave equation as a special case, and the Korteweg–deVries equation to illustrate some basic properties of solitons (as illustrated on the cover). The versatility of the MOL analysis is illustrated by linear and nonlinear PDEs in one dimension (1D), 2D, and 3D with a variety of boundary conditions, for example, Dirichlet, Neumann, and third type.

- A listing of a complete, commented computer program or code, written in Matlab, to solve the model equations. Thus, the programming is all in one place, and therefore a back-and-forth study of the chapter and programming located elsewhere (e.g., on a CD or in a Web link) is not required (although all of the Matlab routines are available from the Web site <http://www.pdecomp.net>).
- A step-by-step explanation of the code, with emphasis on the associated mathematics and computational algorithms at each step.
- A discussion of the output from the code, both numerically tabulated and plotted. In particular, the details of the solution that demonstrate features of the model equations and characteristics of the numerical algorithm are highlighted. The graphical output is typically in 2D and 3D, and in some applications includes movies/animations.
- The output is also evaluated with respect to accuracy, either by comparison with an analytical (exact) solution when available or by inference from changes in the approximations used in the numerical algorithms.
- A summary at the end of the chapter to reiterate (a) the general features and limitations of the numerical algorithm, and (b) the class of problems that the numerical algorithm can address.

All of the models in this volume are based on PDEs. However, because of the use of the MOL (again, in which the PDEs are replaced by systems of approximating ODEs), both ODEs and PDEs are covered along with associated algorithms. Our expectation is that the different types of models, covering all of the major classes of PDEs (parabolic, hyperbolic, elliptic), will provide a starting point for the numerical study of the ODE/PDE system of interest. This might be a straightforward modification of a computer code or extend to the development of a new code based on the ideas presented in one or more examples.

To this end, the chapters are essentially self-contained; they do not require reading the preceding chapters. Rather, we have tried to explain all of the relevant ideas within each chapter, which in some instances requires some repetition between chapters. Also, other chapters are occasionally mentioned for additional details, but it is not necessary to read those chapters. Six appendices are also included to cover concepts that are relevant to more than a single chapter.

We hope this format of self-contained chapters, rather than a chapter-to-chapter format, will be helpful in minimizing the reading and studying required to start the solution to the ODE/PDE system of interest. We welcome your comments about this organization, and your questions about any of the concepts and details presented, as reflected in the following lists of topics. We think these lists, along with the table of contents and the concluding index, will point to the chapters and pages relevant to the problem of interest.

Topic	Chapter
Burgers equation	5
Characteristics of hyperbolic PDEs	8
Complex PDEs	6
Conservation principles	13
Continuation methods	11
Coordinate-free operators	5, 9, 14
Cylindrical coordinates	13
Cubic Schrodinger equation	6
D'Alembert solution	8
Differential-algebraic equations (DAEs)	12
Differential operators	5, 9
Dirichlet boundary conditions	2, 5, 11
Discontinuous solutions	5, 8
Euler equations	5
Exact (analytical) solutions	1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12
Finite-difference library routines	1, 2, 3, 5, 9, 14
Finite differences (FDs)	1, 2
Front sharpening	5
Green's function analysis	3
h - and p -refinement	5, 9, 11
Helmholtz's equation	10
Higher-order FDs	1, 4
Implicit ODEs	12
Infinite spatial domains	6, 7, 8

Topic	Chapter
Inhomogeneous PDEs	4, 11
Integral invariants	3, 7
Jacobian matrix	7, 9
Korteweg–deVries equation	7
Laplace’s equation	10, 11
Linear PDEs	2
Maxwell’s equations	9
Method of lines (MOL)	1
Mixed boundary conditions	11, 13
Mixed (hyperbolic–parabolic) PDEs	9, 13
Mixed partial derivatives	12
Navier Stokes equations	5
Neumann boundary conditions	2, 5, 11, 13, 14
Nonlinear PDEs	4, 5, 6, 7, 13
Numerical quadrature	3, 7
Numerical solution accuracy	1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12
PDE derivations	5, 13
PDE simplification	5, 9
PDE test problems	9
Poisson’s equation	10
Robin boundary conditions	11, 13
Second-order PDEs	8, 9
Shock formation	5
Simultaneous PDEs	4, 6, 13
Singularities	13, 14
Solitons	7
Source terms	11, 13, 14
Sparse matrix integration	6, 7, 9
Spatial convergence	5, 9
Spherical coordinates	14
Stagewise differentiation	3, 5, 12
Tensors	5
Third-type boundary conditions	11, 13
Three-dimensional PDEs	11
Traveling wave solutions	5, 6, 7, 8
Two-dimensional PDEs	10, 13, 14
Units in PDEs	13
Variable-coefficient PDEs	4, 13, 14
Vector operators	5, 9
Wave equation	8, 9

This list contains primarily mathematical topics. The programming in each of the chapters is also a major topic.

The six appendices cover the following topics:

Topic	Appendix
Algebraic grid points	4
Analytical solutions	3
Anisotropic diffusion	1
Cartesian coordinates	1
Conservation principles	1
Cylindrical coordinates	1
Differential grid points	4
Differential operators	1
Dirichlet boundary conditions	4
Finite-difference order conditions	2
Finite-difference test problems	2
Finite differences (FDs)	2, 5
Library FD routines	5
Movies/animations	6
PDE derivations	1
Spherical coordinates	1
Tensors	1
Time-varying boundary conditions	4
Traveling waves	3
Truncation error	2
Vector operators	1

We have assumed a background of basic calculus and ODEs. Since the central algorithm is the MOL, we begin with a MOL introduction in Chapter 1. Then the chapters progress through example applications of increasing complexity and diversity. The preceding list serves as a guide for specific topics.

We have not included exercises at the end of the chapters since we think variations in the applications and the associated Matlab codes provide ample opportunities for exploration and further study. References are provided at the end of the chapters and appendices when we think they would provide useful additional background, but we have not attempted a comprehensive list of references on any particular topic.

Our intent for this volume is to present mathematical and computational methods that can be applied to a broad spectrum of ODE/PDE models. In particular, we are attempting to assist engineers and scientists who have an ODE/PDE problem of interest and who wish to produce an accurate numerical solution with reasonable computational effort without having to first delve into the myriad details of numerical methods and computer programming. We hope this book is of assistance toward meeting this objective.

William E. Schiesser
Bethlehem, PA, USA

Graham W. Griffiths
Nayland, Suffolk, UK
August 1, 2008

An Introduction to the Method of Lines¹

The chapters in this book pertain particularly to mathematical models expressed as *partial differential equations* (PDEs). The computer-based numerical solution of the PDE models is implemented primarily through the *method of lines* (MOL). We therefore start with this chapter, which is an introduction to the MOL. Although the reader may be familiar with the MOL, we suggest reading this chapter since it describes some aspects and details of our use of the MOL that appear in the subsequent chapters. We start with some basic features of PDEs.

SOME PDE BASICS

Our physical world is most generally described in scientific and engineering terms with respect to three-dimensional (3D) space and time, which we abbreviate as *spacetime*. PDEs provide a mathematical description of physical spacetime, and they are therefore among the most widely used forms of mathematics. As a consequence, methods for the solution of PDEs, such as the MOL, are of broad interest in science and engineering.

As a basic illustrative example of a PDE, we consider

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2} \quad (1.1)$$

where

- u dependent variable (dependent on x and t)
- t independent variable representing time
- x independent variable representing one dimension of 3D space
- D constant explained next

Note that Eq. (1.1) has two independent variables, x and t , which is the reason it is classified as a PDE (any differential equation with more than one independent

¹ This chapter is based on an article that originally appeared in the online encyclopedia *Scholarpedia* [1].

variable is a PDE). A differential equation with only one independent variable is generally termed an *ordinary differential equation* (ODE); we will consider ODEs later as part of the MOL.

Equation (1.1) is termed the *diffusion equation*. When applied to heat transfer, it is *Fourier's second law*; the dependent variable u is temperature and D is the *thermal diffusivity*. When Eq. (1.1) is applied to mass diffusion, it is *Fick's second law*; u is mass concentration and D is the *coefficient of diffusion* or the *diffusivity*.

$\partial u / \partial t$ is a partial derivative of u with respect to t (x is held constant when taking this partial derivative, which is why *partial* is used to describe this derivative). Equation (1.1) is *first order in t* since the highest-order partial derivative in t is first order; it is *second order in x* since the highest-order derivative in x is second order. Equation (1.1) is *linear* or *first degree* since all of the terms are to the first power (note that *order* and *degree* can easily be confused).

INITIAL AND BOUNDARY CONDITIONS

Before we consider a solution to Eq. (1.1), we must specify some *auxiliary conditions* to complete the statement of the PDE problem. The number of required auxiliary conditions is determined by the *highest-order derivative in each independent variable*. Since Eq. (1.1) is first order in t and second order in x , it requires one auxiliary condition in t and two auxiliary conditions in x . To have a complete, *well-posed* problem, some additional conditions may have to be included, for example, that specify valid ranges for coefficients. However, this is a more advanced topic and will not be developed further here.

t is termed an *initial-value variable* and therefore requires *one initial condition* (IC). It is an initial-value variable since it starts at an initial value, t_0 , and moves forward over a *finite interval* $t_0 \leq t \leq t_f$ or a *semi-infinite interval* $t_0 \leq t \leq \infty$ without any additional conditions being imposed. Typically in a PDE application, the initial-value variable is time, as in the case of Eq. (1.1).

x is termed a *boundary-value variable* and therefore requires *two boundary conditions* (BCs). It is a boundary-value variable since it varies over a *finite interval* $x_0 \leq x \leq x_f$, a *semi-infinite interval* $x_0 \leq x \leq \infty$, or a *fully infinite interval* $-\infty \leq x \leq \infty$, and at *two different values of x* , conditions are imposed on u in Eq. (1.1). Typically, the two values of x correspond to boundaries of a physical system, and hence the name *boundary conditions*.

As examples of auxiliary conditions for Eq. (1.1),

- An IC could be

$$u(x, t = 0) = u_0 \quad (1.2)$$

where u_0 is a given function of x (typically a constant) for $x_0 \leq x \leq x_f$.

- Two BCs could be

$$u(x = x_0, t) = u_b \quad (1.3a)$$

$$\frac{\partial u(x = x_f, t)}{\partial x} = 0 \quad (1.3b)$$

where u_b is a given boundary (constant) value of u for all t .

- Another common possibility is where the IC is given as earlier and the BCs are $u(x = x_0, t) = f_0(t)$ and $u_x(x = x_f, t) = f_b(t)$.

An important consideration is the possibility of *discontinuities at the boundaries*, produced, for example, by differences in ICs and BCs at the boundaries, which can cause computational difficulties, particularly for *hyperbolic PDEs* (such as the classic linear wave equation $\partial^2 u / \partial t^2 = \partial^2 u / \partial x^2$).

BCs can be of three types:

1. If the dependent variable is specified, as in BC (1.3a), the BC is termed *Dirichlet*.
2. If the derivative of the dependent variable is specified, as in BC (1.3b), the BC is termed *Neumann*.
3. If both the dependent variable and its derivative appear in the BC, it is termed a *BC of the third type* or a *Robin BC*.

TYPES OF PDE SOLUTIONS

Equations (1.1)–(1.3) constitute a complete (*well-posed*) PDE problem and we can now consider what we mean by a solution to this problem. Briefly, the solution of a PDE problem is a *function that defines the dependent variable as a function of the independent variables* – in this case, $u(x, t)$. In other words, we seek a function that when substituted in the PDE and all of its auxiliary conditions satisfies simultaneously all of these equations.

The solution can be of two types:

1. If the solution is an actual mathematical function, it is termed an *analytical solution*. While analytical solutions are the “gold standard” for PDE solutions in the sense that they are exact, they are also generally difficult to derive mathematically for all but the simplest PDE problems (in much the same way that solutions to nonlinear algebraic equations generally cannot be derived mathematically except for certain classes of nonlinear equations).
2. If the solution is in numerical form, for example, $u(x, t)$ tabulated numerically as a function of x and t , it is termed a *numerical solution*. Ideally, the numerical solution is simply a numerical evaluation of the analytical solution. But since an analytical solution is generally unavailable for realistic PDE problems in science and engineering, the *numerical solution is an approximation to the analytical solution*, and our expectation is that it represents the analytical solution with good accuracy. However, numerical solutions can be computed with modern-day computers for very complex problems, and they will generally have good accuracy (even though this cannot be established directly by comparison with the analytical solution since the latter is usually unknown).

The focus of the MOL is *the calculation of accurate numerical solutions*.

PDE SUBSCRIPT NOTATION

Before we go on to the general classes of PDEs that the MOL can handle, we briefly discuss an alternative notation for PDEs. Instead of writing the partial derivatives

as in Eq. (1.1), we adopt a *subscript notation* that is easier to state and bears a closer resemblance to the associated computer coding. For example, we can write Eq. (1.1) as

$$u_t = Du_{xx} \quad (1.4)$$

where, for example, u_t is subscript notation for $\partial u / \partial t$. In other words, a partial derivative is represented as the dependent variable with a subscript that defines the independent variable. For a derivative that is of order n , the independent variable is repeated n times; for example, for Eq. (1.1), u_{xx} represents $\partial^2 u / \partial x^2$.

A GENERAL PDE SYSTEM

Using the subscript notation, we can now consider some general PDEs. For example, a general PDE first order in t can be considered:

$$\bar{u}_t = \bar{f}(\bar{x}, t, \bar{u}, \bar{u}_{\bar{x}}, \bar{u}_{\bar{x}\bar{x}}, \dots) \quad (1.5)$$

where an overbar (overline) denotes a vector. For example, \bar{u} denotes a vector of n dependent variables

$$\bar{u} = (u_1, u_2, \dots, u_n)^T$$

that is, a system of n simultaneous PDEs. Similarly, \bar{f} denotes an n vector of derivative functions

$$\bar{f} = (f_1, f_2, \dots, f_n)^T$$

where T denotes a *transpose* (here a row vector is transposed to a column vector). Note also that \bar{x} is a vector of spatial coordinates, so that, for example, in *Cartesian coordinates* $\bar{x} = (x, y, z)^T$, while in *cylindrical coordinates* $\bar{x} = (r, \theta, z)^T$. Thus, Eq. (1.5) can represent PDEs in one, two, and three spatial dimensions.

Since Eq. (1.5) is first order in t , it requires one IC

$$\bar{u}(\bar{x}, t = 0) = \bar{u}_0(\bar{x}, \bar{u}, \bar{u}_{\bar{x}}, \bar{u}_{\bar{x}\bar{x}}, \dots) \quad (1.6)$$

where \bar{u}_0 is an n vector of IC functions

$$\bar{u}_0 = (u_{10}, u_{20}, \dots, u_{n0})^T$$

The derivative vector \bar{f} of Eq. (1.5) includes functions of various spatial derivatives, $(\bar{u}, \bar{u}_{\bar{x}}, \bar{u}_{\bar{x}\bar{x}}, \dots)$, and therefore we cannot state a priori the required number of BCs. For example, if the highest-order derivative in \bar{x} in all of the derivative functions is second order, then we require $2 \times d \times n$ BCs, where d is the number of spatial dimensions. Thus, for Eq. (1.4), the number of required BCs is 2 (second order in x) $\times 1$ (one dimensional) $\times 1$ (one PDE) = 2 .

We state the general BC requirement of Eq. (1.5) as

$$\bar{f}_b(\bar{x}_b, \bar{u}, \bar{u}_{\bar{x}}, \bar{u}_{\bar{x}\bar{x}}, \dots) = 0 \quad (1.7)$$

where the subscript b denotes *boundary*. The vector of BC functions, \bar{f}_b , has a length (number of functions) determined by the highest-order derivative in \bar{x} in each PDE (in Eq. (1.5)), as discussed previously.

PDE GEOMETRIC CLASSIFICATION

Equations (1.5)–(1.7) constitute a general PDE system to which the MOL can be applied. Before proceeding to the details of how this might be done, we need to discuss the three basic forms of the PDEs as classified geometrically. This *geometric classification* can be done rigorously if certain mathematical forms of the functions in Eqs. (1.5)–(1.7) are assumed. However, we will adopt a somewhat more descriptive (less rigorous but more general) form of these functions for the specification of the three geometric classes.

If the derivative functions in Eq. (1.5) contain *only first-order derivatives in \bar{x}* , the PDEs are classified as *first-order hyperbolic*. As an example, the equation

$$u_t + vu_x = 0 \quad (1.8)$$

is generally called the *linear advection equation*; in physical applications, v is a linear or flow velocity. Although Eq. (1.8) is possibly the simplest PDE, this simplicity is deceptive in the sense that it can be very difficult to integrate numerically since it *propagates discontinuities*, a distinctive feature of first-order hyperbolic PDEs.

Equation (1.8) is termed a *conservation law* since it typically expresses conservation of *mass, energy, or momentum* under the conditions for which it is derived, that is, the *assumptions on which the equation is based*. Conservation laws are a bedrock of PDE mathematical models in science and engineering, and an extensive literature pertaining to their solution, both analytical and numerical, has evolved over many years.

An example of a *first-order hyperbolic system* (using the notation $u_1 \Rightarrow u$, $u_2 \Rightarrow v$) is

$$u_t = v_x \quad (1.9a)$$

$$v_t = u_x \quad (1.9b)$$

Equations (1.9a) and (1.9b) constitute a system of *two linear, constant-coefficient, first-order hyperbolic PDEs*.

Differentiation and algebraic substitution can occasionally be used to eliminate some dependent variables in systems of PDEs. For example, if Eq. (1.9a) is differentiated with respect to t and Eq. (1.9b) is differentiated with respect to x

$$u_{tt} = v_{xt}$$

$$v_{tx} = u_{xx}$$

we can then eliminate the mixed partial derivative between these two equations (assuming v_{xt} in the first equation equals v_{tx} in the second equation) to obtain

$$u_{tt} = u_{xx} \quad (1.10)$$

Equation (1.10) is the *second-order hyperbolic wave equation*.

If the derivative functions in Eq. (1.5) contain *only second-order derivatives in \bar{x}* , the PDEs are classified as *parabolic*. Equation (1.1) is an example of a parabolic PDE.

Finally, if a PDE contains no derivatives in t (e.g., the LHS of Eq. (1.5) is zero), it is classified as *elliptic*. As an example,

$$u_{xx} + u_{yy} = 0 \quad (1.11)$$

is *Laplace's equation*, where x and y are spatial independent variables in Cartesian coordinates. Note that with no derivatives in t , *elliptic PDEs require no ICs*; that is, they are entirely boundary-value PDEs.

PDEs with *mixed geometric characteristics* are possible and, in fact, are quite common in applications. For example, the PDE

$$u_t = -u_x + u_{xx} \quad (1.12)$$

is *hyperbolic-parabolic*. Since it frequently models convection (hyperbolic) through the term u_x and diffusion (parabolic) through the term u_{xx} , it is generally termed a *convection-diffusion equation*. If additionally, it includes a function of the dependent variable u such as

$$u_t = -u_x + u_{xx} + f(u) \quad (1.13)$$

then it might be termed a *convection-diffusion-reaction equation* since $f(u)$ typically models the rate of a chemical reaction. If the function is only for the independent variables, that is,

$$u_t = -u_x + u_{xx} + g(x, t) \quad (1.14)$$

the equation could be labeled an *inhomogeneous PDE*.

This discussion clearly indicates that PDE problems come in a very wide variety, depending, for example, on linearity, types of coefficients (constant, variable), coordinate system, geometric classification (hyperbolic, elliptic, parabolic), number of dependent variables (number of simultaneous PDEs), number of independent variables (number of dimensions), types of BCs, smoothness of the IC, and so on, so it might seem impossible to formulate numerical procedures with any generality that can address a relatively broad spectrum of PDEs. But in fact, the MOL provides a surprising degree of generality, although the success in applying it to a new PDE problem depends to some extent on the experience and inventiveness of the analyst; that is, MOL is not a single, straightforward, clearly defined approach to PDE problems, but rather is a general concept (or philosophy) that requires specification of details for each new PDE problem. We now proceed to illustrate the formulation of a MOL numerical algorithm, with the caveat that this will not be a general discussion of MOL as it is applied to any conceivable PDE problem.

ELEMENTS OF THE MOL

The basic idea of the MOL is to *replace the spatial (boundary-value) derivatives in the PDE with algebraic approximations*. Once this is done, the spatial derivatives are no longer stated explicitly in terms of the spatial independent variables. Thus, in effect, *only the initial-value variable, typically time in a physical problem, remains*. In other words, with only one remaining independent variable, we have a *system of ODEs that approximate the original PDE*. The challenge, then, is to formulate the approximating system of ODEs. Once this is done, we can apply any integration

algorithm for initial-value ODEs to compute an approximate numerical solution to the PDE. Thus, one of the salient features of the MOL is the use of *existing, and generally well-established, numerical methods for ODEs*.

To illustrate this procedure, we consider the MOL solution of Eq. (1.8). First we need to replace the spatial derivative u_x with an algebraic approximation. In this case we will use a *finite difference* (FD), such as

$$u_x \approx \frac{u_i - u_{i-1}}{\Delta x} \quad (1.15)$$

where i is an *index designating a position along a grid in x and Δx is the spacing in x along the grid*. Thus, for the left-end value of x , $i = 1$, and for the right-end value of x , $i = M$; that is, the grid in x has M points. Then the MOL approximation of Eq. (1.8) is

$$\frac{du_i}{dt} = -v \frac{u_i - u_{i-1}}{\Delta x}, \quad 1 \leq i \leq M \quad (1.16)$$

Note that Eq. (1.16) is written as an ODE since there is now *only one independent variable, t* . Note also that Eq. (1.16) represents a system of M ODEs.

This transformation of a PDE, Eq. (1.8), to a system of ODEs, Eq. (1.16), illustrates the essence of the MOL, namely, *the replacement of the spatial derivatives, in this case u_x , so that a system of ODEs approximates the original PDE*. Then, *to compute the solution of the PDE, we compute a solution to the approximating system of ODEs*. But before considering this integration in t , we have to complete the specification of the PDE problem. Since Eq. (1.8) is first order in t and first order in x , it requires one IC and one BC. These will be taken as

$$u(x, t = 0) = f(x) \quad (1.17a)$$

$$u(x = 0, t) = g(t) \quad (1.17b)$$

Since Eq. (1.16) constitutes M first-order, initial-value ODEs, M initial conditions are required, and from Eq. (1.17a), these are

$$u(x_i, t = 0) = f(x_i), \quad 1 \leq i \leq M \quad (1.18a)$$

Also, application of BC (1.17b) gives for grid point $i = 1$

$$u(x_1, t) = g(t) \quad (1.18b)$$

Equations (1.16) and (1.18) now constitute the complete MOL approximation of Eq. (1.8) subject to Eqs. (1.17a) and (1.17b). The solution of this ODE system gives the M functions

$$u_1(t), u_2(t), \dots, u_{M-1}(t), u_M(t) \quad (1.19)$$

that is, an approximation to $u(x, t)$ at the grid points $i = 1, 2, \dots, M$.

Before we go on to consider the numerical integration of the approximating ODEs, in this case Eq. (1.16), we briefly consider further the FD approximation of Eq. (1.15), which can be written as

$$u_x \approx \frac{u_i - u_{i-1}}{\Delta x} + O(\Delta x) \quad (1.20)$$

where $O(\Delta x)$ denotes *of order* Δx ; that is, the truncation error of the approximation of Eq. (1.16) is *proportional to* Δx *to the first power* (*varies linearly with* Δx); thus, Eq. (1.20) is also termed a *first-order FD* (since Δx is to the first power in the order or truncation error term). The term *truncation error* reflects the fact that the FD of Eq. (1.15) comes from a *truncated Taylor series*.

Note that the numerator of Eq. (1.15), $u_i - u_{i-1}$, is a difference in two values of u . Also, the denominator Δx remains finite (nonzero). Hence the name *finite difference* (and it is an approximation because of the truncated Taylor series, so a more complete description is *first-order FD approximation*). In fact, in the limit $\Delta x \rightarrow 0$, the approximation of Eq. (1.15) becomes *exactly the derivative*. However, in a practical computer-based calculation, Δx remains finite, so Eq. (1.15) remains an approximation.

Also, Eq. (1.8) typically describes the flow of a physical quantity such as concentration of a chemical species or temperature, represented by u , from left to right with respect to x with velocity v . Then, using the FD approximation of Eq. (1.20) at i involves u_i and u_{i-1} . In a flowing system, u_{i-1} is to the left (in x) of u_i or is *upstream* or *upwind* of u_i (to use a nautical analogy). Thus, Eq. (1.20) is termed a *first-order upwind FD approximation*. Generally, for strongly convective systems such as that modeled by Eq. (1.8), *some form of upwinding is required in the numerical solution of the descriptive PDEs*; we will look at this requirement further in the subsequent discussion.

ODE INTEGRATION WITHIN THE MOL

We now consider briefly the numerical integration of the M ODEs of Eq. (1.16). If the derivative du_i/dt is approximated by a first-order FD

$$\frac{du_i}{dt} \approx \frac{u_i^{n+1} - u_i^n}{\Delta t} + O(\Delta t) \quad (1.21)$$

where n is an index for the variable t (t moves forward in steps denoted or indexed by n), then an FD approximation of Eq. (1.16) is

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -v \frac{u_i^n - u_{i-1}^n}{\Delta x}$$

or solving for u_i^{n+1} ,

$$u_i^{n+1} = u_i^n - (v\Delta t/\Delta x)(u_i^n - u_{i-1}^n), \quad i = 1, 2, \dots, M \quad (1.22)$$

Equation (1.22) has the important characteristic that it gives u_i^{n+1} *explicitly*; that is, we can solve for the solution at the advanced point in t , $n + 1$, from the solution at the base point n . In other words, explicit numerical integration of Eq. (1.16) is by the *forward FD* of Eq. (1.21), and this procedure is generally termed the *forward Euler method*, which is the most basic form of ODE integration.

While the explicit form of Eq. (1.22) is computationally convenient, it has a possible limitation. If the time step Δt is *above a critical value*, *the calculation becomes unstable*, which is manifest by successive changes in the dependent variable, $\Delta u = u_i^{n+1} - u_i^n$, becoming larger and eventually unbounded as the calculation moves forward in t (for increasing n). In fact, for the solution of Eq. (1.8) by the

method of Eq. (1.22) to remain stable, the dimensionless group $(v\Delta t/\Delta x)$, which is called the *Courant-Friedricks-Lewy or CFL number*, must remain below a critical value – in this case, unity. Note that this *stability limit* places an upper limit on Δt for a given v and Δx ; if one attempts to increase the accuracy of Eq. (1.22) by using a smaller Δx (larger number of grid points in x by increasing M), a smaller value of Δt is required to keep the CFL number below its critical value. Thus, there is a *conflicting requirement of improving accuracy while maintaining stability*.

The stability limit of the explicit Euler method as implemented via the forward FD of Eq. (1.21) can be circumvented by using a *backward FD* for the derivative in t

$$\frac{du_i}{dt} \approx \frac{u_i^n - u_i^{n-1}}{\Delta t} + O(\Delta t) \quad (1.23)$$

so that the FD approximation of Eq. (1.16) becomes

$$\frac{u_i^n - u_i^{n-1}}{\Delta t} = -v \frac{u_i^n - u_{i-1}^n}{\Delta x}$$

or after rearrangement (with $(v\Delta t/\Delta x) = \alpha$),

$$(1 + \alpha)u_i^n - \alpha u_{i-1}^n = u_i^{n-1}, \quad i = 1, 2, \dots, M \quad (1.24)$$

Note that we cannot now solve Eq. (1.24) explicitly for the solution at the advanced point u_i^n in terms of the solution at the base point u_i^{n-1} . Rather, Eq. (1.24) is *implicit* in u_i^n because u_{i-1}^n is also unknown; that is, we must solve Eq. (1.24) written for each grid point $i = 1, 2, \dots, M$ as a simultaneous system of *bidiagonal equations* (bidiagonal because each of Eq. (1.24) has two unknowns so that *simultaneous solution of the full set of approximating algebraic equations is required* to obtain the complete numerical solution $u_1^n, u_2^n, \dots, u_M^n$). Thus, the solution of Eq. (1.24) is termed the *implicit Euler method*.

We could then naturally ask why use Eq. (1.24) when Eq. (1.22) is so much easier to use (explicit calculation of the solution at the next step in t of Eq. (1.22) vs. the implicit calculation of Eq. (1.24)). The answer is that the implicit calculation of Eq. (1.24) is often worthwhile because the *implicit Euler method has no stability limit* (is *unconditionally stable* in comparison with the explicit method, with the stability limit stated in terms of the CFL condition). However, there is a price to pay for the improved stability of the implicit Euler method; that is, we must solve a *system of simultaneous algebraic equations*; Eq. (1.24) is an example. Furthermore, if the original ODE system approximating the PDE is nonlinear, we have to solve a *system of nonlinear algebraic equations*. (Equation (1.24) is linear, so the solution is much easier.) The system of nonlinear equations is typically solved by a *variant of Newton's method* that can become very demanding computationally if the number of ODEs is large (due to the use of a large number of spatial grid points in the MOL approximation of the PDE, especially when we attempt the solution of 2D and 3D PDEs). If you have had some experience with Newton's method, you may appreciate that the *Jacobian matrix* of the nonlinear algebraic system can *become very large and sparse as the number of spatial grid points increases*.

Additionally, although there is no limit for Δt with regard to stability for the implicit method, there is a *limit with regard to accuracy*. In fact, the first-order upwind

approximation of u_x in Eq. (1.8), Eq. (1.20), and the first-order approximation of u_t in Eq. (1.8), Eq. (1.21) or (1.23), taken together limit the accuracy of the resulting FD approximation of Eq. (1.8). One way around this accuracy limitation is to use *higher-order FD approximations for the derivatives in Eq. (1.8)*.

For example, if we consider the second-order approximation of u_x at i

$$u_x \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x} + O(\Delta x^2) \quad (1.25)$$

substitution in Eq. (1.8) gives the MOL approximation of Eq. (1.8)

$$\frac{du_i}{dt} = -v \frac{u_{i+1} - u_{i-1}}{2\Delta x}, \quad 1 \leq i \leq M \quad (1.26)$$

We could then reason that if the integration in t is performed by the explicit Euler method, that is, we use the approximation of Eq. (1.21) for $u_t = du_i/dt$, the resulting numerical solution should be more accurate than the solution from Eq. (1.22). In fact, the MOL approximation based on this idea

$$u_i^{n+1} = u_i^n - \frac{v\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n), \quad i = 1, 2, \dots, M \quad (1.27)$$

is *unconditionally unstable*; this conclusion can be demonstrated by a *von Neumann stability analysis* that we will not cover here. This surprising result demonstrates that *replacing the derivatives in PDEs with higher-order approximations does not necessarily guarantee more accurate solutions, or even stable solutions*.

NUMERICAL DIFFUSION AND OSCILLATION

Even if the implicit Euler method is used for the integration in t of Eq. (1.26) to achieve stability (or a more sophisticated explicit integrator in t is used that automatically adjusts Δt to achieve a prescribed accuracy), we would find that the solution *oscillates* unrealistically. This numerical distortion is one of two generally observed forms of numerical error. The other numerical distortion is *diffusion* that would be manifest in the solution from Eq. (1.22). Briefly, the solution would exhibit excessive smoothing or rounding at points in x where the solution changes rapidly. This overall observation that a *first-order approximation of u_x produces numerical diffusion, while higher-order approximations of u_x produce numerical oscillation* is predicted by the *Godunov order barrier theorem for the Riemann problem* [2]. To explain briefly, the order barrier is first order and *any linear FD approximation above first order will be oscillatory*. Equation (1.8) is an example of the Riemann problem [2] if IC Eq. (1.17a) is discontinuous; for example, $u(x, t = 0) = h(t)$, where $h(t)$ is the *Heaviside unit step function*. The (exact) analytical solution is the IC function $f(x)$ of Eq. (1.17a) moving left to right with velocity v (from Eq. (1.8)) and without distortion, that is, $u(x, t) = f(x - vt)$; however, the numerical solution will oscillate if u_x in Eq. (1.8) is replaced with a linear approximation of second or higher order.

We should also mention one point of terminology for FD approximations. The RHS of Eq. (1.25) is an example of a *centered approximation* since the two points at $i + 1$ and $i - 1$ are centered around the point i . Equation (1.20) is an example of a *noncentered, one-sided, or upwind approximation* since the points i and $i - 1$ are not centered with respect to i .

Finally, to conclude the discussion of first-order hyperbolic PDEs such as Eq. (1.8), since the Godunov theorem indicates that FD approximations above first order will produce numerical oscillations in the solution, the question remains if there are approximations above first order that are nonoscillatory. To answer this question we note first that *the Godunov theorem applies to linear approximations*; for example, Eq. (1.25) is a linear approximation since u on the RHS is to the first power. If, however, we consider nonlinear approximations of u_x , we can in fact develop approximations that are nonoscillatory. The details of such nonlinear approximations are beyond the scope of this discussion, so we will merely mention that they are termed *high-resolution* methods that seek a *total variation diminishing solution*. Such methods, which include *flux limiter* and *weighted essentially nonoscillatory* methods, seek to avoid nonreal oscillations when shocks or discontinuities occur in the solution (such as in the Riemann problem) [3].

So far we have considered only the MOL solution of first-order PDEs, for example, Eq. (1.8). We conclude this discussion of the MOL by considering a second-order PDE, the parabolic Eq. (1.1). To begin, we need an approximation for the second derivative u_{xx} . A commonly used *second-order, central approximation* is (again, derived from the Taylor series, so the term $O(\Delta x^2)$ represents the truncation error)

$$u_{xx} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + O(\Delta x^2) \quad (1.28)$$

Substitution of Eq. (1.28) in Eq. (1.8) gives a system of approximating ODEs

$$\frac{du_i}{dt} = D \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}, \quad i = 1, 2, \dots, M \quad (1.29)$$

Equation (1.29) is then integrated subject to IC (1.2) and BCs (1.3a) and (1.3b). This integration in t can be by the explicit Euler method, the implicit Euler method, or any other higher-order integrator for initial-value ODEs. Generally stability is not as much of a concern as with the previous hyperbolic PDEs (a characteristic of parabolic PDEs that tend to smooth solutions rather than hyperbolic PDEs that tend to propagate nonsmooth conditions). However, stability constraints do exist for explicit methods. For example, for the explicit Euler method with a step Δt in t , the stability constraint is $D\Delta t/\Delta x^2 < \text{constant}$ (so that as Δx is reduced to achieve better spatial accuracy in x , Δt must also be reduced to maintain stability).

Before proceeding with the integration of Eq. (1.29), we must include BCs (1.3a) and (1.3b). The Dirichlet BC at $x = x_0$, Eq. (1.3a), is merely

$$u_1 = u_b \quad (1.30)$$

and therefore the ODE of Eq. (1.29) for $i = 1$ is not required and the ODE for $i = 2$ becomes

$$\frac{du_2}{dt} = D \frac{u_3 - 2u_2 + u_b}{\Delta x^2} \quad (1.31)$$

DIFFERENTIAL ALGEBRAIC EQUATIONS

Equation (1.30) is *algebraic*, and therefore in combination with the ODEs of Eq. (1.29), we have a *differential algebraic (DAE) system*.

At $i = M$, we have Eq. (1.29)

$$\frac{du_M}{dt} = D \frac{u_{M+1} - 2u_M + u_{M-1}}{\Delta x^2} \quad (1.32)$$

Note that u_{M+1} is outside the grid in x ; that is, $M + 1$ is a *fictitious point*. But we must assign a value to u_{M+1} if Eq. (1.32) is to be integrated. Since this requirement occurs at the boundary point $i = M$, we obtain this value by approximating BC (1.3b) using the centered FD approximation of Eq. (1.25)

$$u_x \approx \frac{u_{M+1} - u_{M-1}}{2\Delta x} = 0$$

or

$$u_{M+1} = u_{M-1} \quad (1.33)$$

We can add Eq. (1.33) as an *algebraic equation to our system of equations*, that is, continue to use the DAE format, or we can substitute u_{M+1} from Eq. (1.33) into the Eq. (1.32)

$$\frac{du_M}{dt} = D \frac{u_{M-1} - 2u_M + u_{M-1}}{\Delta x^2} \quad (1.34)$$

and arrive at an ODE system (Eq. (1.29) for $i = 3, \dots, M - 1$, Eq. (1.31) for $i = 2$, and Eq. (1.34) for $i = M$). Both approaches, either an ODE system or a DAE system, have been used in MOL studies. Either way, we now have a complete formulation of the MOL ODE or DAE system, including the BCs at $i = 1, M$ in Eq. (1.29). The integration of these equations then gives the numerical solution $u_1(t), u_2(t), \dots, u_M(t)$. The preceding discussion is based on a relatively basic DAE system, but it indicates that integrators designed for DAE systems can play an important role in MOL analysis.

If the implicit Euler method is applied to Eq. (1.29), we have

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = D \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2}, \quad i = 1, 2, \dots, M$$

or (with $\alpha = D\Delta t/\Delta x^2$),

$$u_{i+1}^{n+1} - (1/\alpha + 2)u_i^{n+1} + u_{i-1}^{n+1} = (1/\alpha)u_i^n, \quad i = 1, 2, \dots, M$$

which is a *tridiagonal* system of algebraic equations (three unknowns in each equation). Since such *banded* systems (the nonzero elements are banded around the main diagonal) are common in the numerical solution of PDE systems, special algorithms have been developed to take advantage of the banded structure, typically by not storing and using the zero elements outside the band. These special algorithms that take advantage of the *structure of the problem equations* can result in major savings in computation time. In the case of tridiagonal equations, the special algorithm is generally called *Thomas' method*. If the coefficient matrix of the algebraic system does not have a well-defined structure, such as bidiagonal or tridiagonal, but consists of mostly zeros with a relatively small number of nonzero elements, which is often the case in the numerical solution of PDEs, the coefficient matrix is said to be *sparse*; special algorithms and associated software for sparse systems have been

developed that can result in very substantial savings in the storage and numerical manipulation of sparse matrices.

Generally when applying the MOL, the integration of the approximating ODE/DAEs (e.g., Eqs. (1.21) and (1.29)) is accomplished by using *library routines for initial-value ODE/DAEs*. In other words, the explicit programming of the ODE/DAE integration (such as the explicit or implicit Euler method) is avoided; rather, an established integrator is used. This has the advantage that (1) the detailed programming of the integration can be avoided, particularly the linear algebra (solution of simultaneous equations) required by an implicit integrator, so that the MOL analysis is substantially simplified, and (2) library routines (usually written by experts) include features that make these routines especially effective (robust) and efficient such as automatic integration step size adjustment and the use of higher-order integration methods (beyond the first-order accuracy of the Euler methods); also, generally, they have been thoroughly tested. Thus, the use of quality ODE/DAE library routines is usually an essential part of MOL analysis. We therefore list at the end of this chapter some public domain sources of such library routines.

HIGHER DIMENSIONS AND DIFFERENT COORDINATE SYSTEMS

To conclude this discussion of the MOL solution of PDEs, we cover two additional points. First, we have considered PDEs in only Cartesian coordinates, and in fact, just one Cartesian coordinate, x . But MOL analysis can in principle be carried out in any coordinate system. Thus, Eq. (1.1) can be generalized to

$$\frac{\partial u}{\partial t} = D \nabla^2 u \quad (1.35)$$

where ∇^2 is the *coordinate independent Laplacian operator* that can then be expressed in terms of a particular coordinate system. For example, in cylindrical coordinates, Eq. (1.35) is

$$\frac{\partial u}{\partial t} = D \left(\frac{\partial^2 u}{\partial r^2} + \frac{1}{r} \frac{\partial u}{\partial r} + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} + \frac{\partial^2 u}{\partial z^2} \right) \quad (1.36)$$

and in spherical coordinates, Eq. (1.35) is

$$\frac{\partial u}{\partial t} = D \left[\frac{\partial^2 u}{\partial r^2} + \frac{2}{r} \frac{\partial u}{\partial r} + \frac{1}{r^2} \left(\frac{\partial^2 u}{\partial \theta^2} + \frac{\cos \theta}{\sin \theta} \frac{\partial u}{\partial \theta} \right) + \frac{1}{r^2 \sin^2 \theta} \frac{\partial^2 u}{\partial \phi^2} \right] \quad (1.37)$$

The challenge then in applying the MOL to PDEs such as Eqs. (1.36) and (1.37) is the algebraic approximation of the RHS ($\nabla^2 u$) using, for example, FDs, *finite elements* or *finite volumes*; all of these approximations have been used in MOL analysis, as well as *Galerkin*, *least squares*, *spectral*, and other methods. A particularly demanding step is *regularization of singularities* such as at $r = 0$ (note the number of divisions by r in the RHS of Eqs. (1.36) and (1.37)) and at $\theta = 0, \pi/2$ (note the divisions by $\sin(\theta)$ in Eq. (1.37)). Thus the application of the MOL typically requires analysis based on the experience and creativity of the analyst (i.e., it is generally not a mechanical procedure from beginning to end).

The complexity of the numerical solution of higher-dimensional PDEs in various coordinate systems prompts the question of why a particular coordinate system

would be selected over others. The mathematical answer is that the judicious choice of a coordinate system *facilitates the implementation of the BCs in the numerical solution*. The answer based on physical considerations is that the *coordinate system is selected to reflect the geometry of the problem system*. For example, if the physical system has the shape of a cylinder, cylindrical coordinates would be used. This choice then facilitates the implementation of the BC at the exterior surface of the physical system (exterior surface of the cylinder). However, this can also lead to complications such as the $r = 0$ singularities in Eq. (1.36) (due to the variable $1/r$ and $1/r^2$ coefficients). The resolution of these complications is generally worth the effort rather than the use of a coordinate system that does not naturally conform to the geometry of the physical system. If the physical system is not shaped in accordance with a particular coordinate system, that is, has an *irregular geometry*, then an approximation to the physical geometry is used, generally termed *body-fitted coordinates*.

***h*- AND *p*-REFINEMENT**

Increasing or decreasing the grid spacing over parts or all of the problem domain is termed *h-refinement*. The name comes from the common convention in the numerical analysis literature of using h as the symbol for the grid spacing.

Modifying the order of the derivative approximation is termed *p-refinement*. The name comes from the convention of using the symbol p for the order of the approximation (e.g., $O(\Delta x)^2$ for a second-order approximation with $p = 2$).

h-refinement seeks to refine the grid spacing using *local truncation error* estimates or other refinement parameters in order to improve the accuracy of the solution. Similarly, *p*-refinement seeks to refine the order of derivative approximations (using the same refinement parameters). Generally there is no unique, general, best combination of *h*- and *p*-refinement and the solution of large problems usually requires some trial and error for a trade-off between accuracy and computational effort; *the goal is to reach a prespecified bound on the global error with a minimal amount of work* [4]. We will not discuss this aspect further here, but refer to [4–6] for further discussion.

ORIGIN OF THE NAME “METHOD OF LINES”

As a concluding point, we might consider the origin of the name *method of lines*. If we consider Eq. (1.29), integration of this ODE system produces the solution $u_2(t), u_3(t), \dots, u_M(t)$. (Note: $u_1(t) = u_b$, a constant, from BC (1.30).) We could then plot these functions in an x - $u(x, t)$ plane as a vertical line at each x ($i = 2, 3, \dots, M$), with the height of each line equal to $u(x_i, t)$. In other words, the plot of the solution would be a set of vertical parallel lines suggesting the name *method of lines* [7].

To illustrate this interpretation, consider the diffusion equation problem of Eq. (1.1) with

- Diffusion coefficient $D = 1$
- Dirichlet BC at the left end, $u(x = -5, t) = 0$

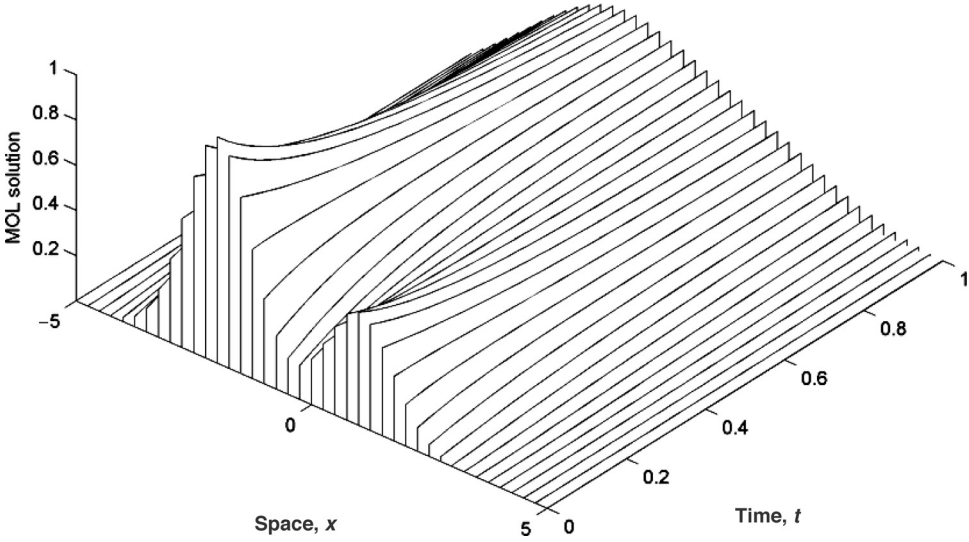


Figure 1.1. MOL solution of Eq. (1.1) illustrating the origin of the *method of lines*

- Neumann BC at the right end, $u_x(x = 5, t) = 0$ (spatial domain $-5 \leq x \leq 5$)
- Time domain $0 \leq t \leq 1$
- Initial condition $u(x, t = 0) = (1/2)e^{-(x-1)^2} + e^{-(x+2)^2}$

The MOL solution for the problem is shown in Figure (1.1). This numerical solution was obtained using Matlab and the MOL library routine `dss044` [7] with the number of grid points $M = 41$ (so that the grid spacing is $[5 - (-5)]/(41 - 1) = 0.25$).

The result of Figure 1.1 matches very well the *infinite-domain analytical solution*

$$u(x, t) = \frac{1}{2\sqrt{4Dt+1}} \left(e^{\frac{3(2x+1)}{4Dt+1}} + 2 \right) e^{-\frac{(x+2)^2}{4Dt+1}} \quad (1.38)$$

This agreement is illustrated in Figure 1.2 where the analytical result has been superimposed on the MOL solution. This comparison illustrates an important distinction between the analytical and numerical (MOL) solutions. The analytical solution is for an infinite domain, $-\infty \leq x \leq \infty$, while the MOL solution is computed on a finite domain (as required by a computer), $-5 \leq x \leq 5$ [1]. The agreement between the analytical and numerical solutions reflects the property that both solutions remain at essentially zero for $u(x = -5, t)$ and $u(x = 5, t)$ for $t \leq 1$ as indicated in Figure 1.2.²

² The exact analytical solution for the finite-domain problem is considerably more complicated than Eq. (1.38) but could be derived by a finite Fourier sine transform ([8], pp. 405–415) or a Green's function ([9], pp. 48, 58).

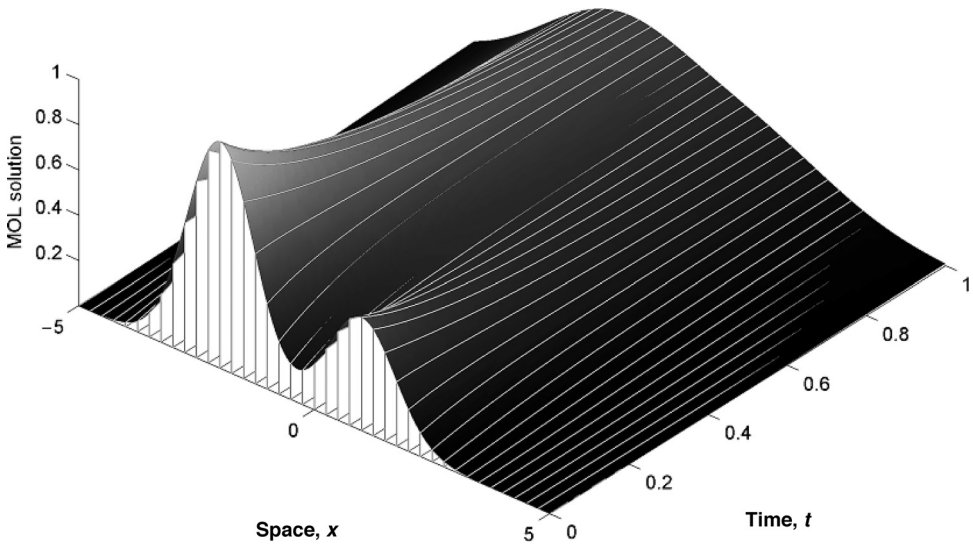


Figure 1.2. Superposition of the MOL solution of Eq. (1.1) and the analytical solution of Eq. (1.38)

SOURCES OF ODE/DAE INTEGRATORS

One of the very useful aspects of MOL is that it enables tried-and-tested ODE/DAE numerical routines to be used, many of which are in the public domain. The following sources are a good starting point for these routines. For example, the LSODE and VODE series of ODE/DAE integrators [2s], DASSL for DAEs [2s], and the SUNDIALS library [5s] are widely used in MOL analysis; test problems for ODE/DAE routines are also available [6s]. Additionally, routines that can be called from MOL codes are available to perform a variety of complementary computations (e.g., functional approximation by interpolation, evaluation of integrals, maximization and minimization in optimization associated with the solution of PDEs) [1s, 3s, 4s].

[1s] <http://www.netlib.org/>

[2s] <http://www.netlib.org/ode/index.html> (emphasis on ODE/DAE software that can be used in MOL analysis)

[3s] <http://gams.nist.gov/>

[4s] <http://www.acm.org/toms/>

[5s] <http://www.llnl.gov/CASC/software.html>

[6s] <http://www.dm.uniba.it/~testset/>

REFERENCES

- [1] Hamdi, S., W. E. Schiesser, and G. W. Griffiths (2007), Method of Lines, *Scholarpedia*, **2**(7):2859; available online at http://www.scholarpedia.org/article/method_of_lines
- [2] Wesseling, P. (2001), *Principles of Computational Fluid Dynamics*, Springer, Berlin
- [3] Shu, C.-W. (1998), Essentially Non-Oscillatory and Weighted Essential Non-Oscillatory Schemes for Hyperbolic Conservation Laws, In: B. Cockburn, C. Johnson, C.-W. Shu,

- and E. Tadmor (Eds.), *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, Lecture Notes in Mathematics, vol. 1697, Springer, Berlin, pp. 325–432
- [4] de Sterck, H., T. A. Manteuffel, S. F. McCormick, J. Nolting, J. Ruge, and L. Tang (2008), Efficiency-Based h - and hp -Refinement Strategies for Finite Element Methods, *Num. Linear Algebr. Appl.*, **15**: 89–114.
- [5] Aftosmis, M. J. and M. J. Berger (2002), Multilevel Error Estimation and Adaptive h -Refinement for Cartesian Meshes with Embedded Boundaries, In: *AIAA Paper 2002-0863, 40th AIAA Aerospace Sciences Meeting and Exhibit*, January 14–17, 2002, Reno, NV
- [6] Dong, S. and G. E. Karniadakis (May 9, 2003), p-Refinement and p-Threads, *Comput. Methods Appl. Mech. Eng.*, **192**(19): 2191–2201
- [7] Schiesser, W. E. (1991), *Numerical Method of Lines Integration of Partial Differential Equations*, Academic Press, San Diego, CA
- [8] Schiesser, W. E. (1994), *Computational Mathematics in Engineering and Applied Science: ODEs, DAEs, and PDEs*, CRC Press, Boca Raton, FL
- [9] Polyanin, A. (2002), *Handbook of Linear Partial Differential Equations for Engineers and Scientists*, Chapman & Hall/CRC, Boca Raton, FL

2

A One-Dimensional, Linear Partial Differential Equation

This partial differential equation (PDE) problem is considered for the following reasons:

1. The PDE has an exact solution that can be used to assess the accuracy of the numerical method of lines (MOL) solution.
2. Both Dirichlet and Neumann boundary conditions (BCs) are included in the analysis.
3. The use of library routines for the finite-difference (FD) approximation of the spatial (boundary-value) derivative is illustrated.
4. The explicit programming of the FD approximations is included for comparison with the use of the library routines.
5. Some basic methods for assessing the accuracy of the MOL solution are presented.

The PDE is the *one-dimensional (1D) heat conduction equation in Cartesian coordinates*:

$$u_t = u_{xx} \quad (2.1)$$

Here we have used subscript notation for partial derivatives, so

$$u_t \leftrightarrow \frac{\partial u}{\partial t}$$
$$u_{xx} \leftrightarrow \frac{\partial^2 u}{\partial x^2}$$

The initial condition (IC) is

$$u(x, t = 0) = \sin(\pi x/2) \quad (2.2)$$

A *Dirichlet* BC is specified at $x = 0$,

$$u(x = 0, t) = 0 \quad (2.3)$$

and a *Neumann* BC is specified at $x = 1$,

$$u_x(x = 1, t) = 0 \quad (2.4)$$

The analytical solution to Eqs. (2.1)–(2.4) is

$$u(x, t) = e^{-(\pi^2/4)t} \sin(\pi x/2) \quad (2.5)$$

A main program in Matlab for the MOL solution of Eqs. (2.1)–(2.4) with the analytical solution, Eq. (2.5), included for comparison with the MOL solution, is given in Listing 2.1.

```
%
% Clear previous files
clear all
clc
%
% Parameters shared with the ODE routine
global ncall ndss
%
% Initial condition
n=21;
for i=1:n
    u0(i)=sin((pi/2.0)*(i-1)/(n-1));
end
%
% Independent variable for ODE integration
t0=0.0;
tf=2.5;
tout=linspace(t0,tf,n);
nout=n;
ncall=0;
%
% ODE integration
mf=1;
reltol=1.0e-04; abstol=1.0e-04;
options=odeset('RelTol',reltol,'AbsTol',abstol);
if(mf==1) % explicit FDs
    [t,u]=ode15s(@pde_1,tout,u0,options); end
if(mf==2) ndss=4; % ndss = 2, 4, 6, 8 or 10 required
    [t,u]=ode15s(@pde_2,tout,u0,options); end
if(mf==3) ndss=44; % ndss = 42, 44, 46, 48 or 50 required
    [t,u]=ode15s(@pde_3,tout,u0,options); end
%
% Store numerical and analytical solutions, errors at x = 1/2
n2=(n-1)/2.0+1;
sine=sin(pi/2.0*0.5);
for i=1:nout
    u_plot(i)=u(i,n2);
    u_anal(i)=exp(-pi^2/4.0*t(i))*sine;
    err_plot(i)=u_plot(i)-u_anal(i);
end
```

```

%
% Display selected output
fprintf('\n mf = %2d  abstol = %8.1e  reltol = %8.1e\n', ...
        mf,abstol,reltol);
fprintf('\n      t      u(0.5,t)  u_anal(0.5,t)
        err u(0.5,t)\n');
for i=1:5:nout
    fprintf('%6.3f%15.6f%15.6f%15.7f\n', ...
            t(i),u_plot(i),u_anal(i),err_plot(i));
end
fprintf('\n ncall = %4d\n',ncall);
%
% Plot numerical solution and errors at x = 1/2
figure(1);
subplot(1,2,1)
plot(t,u_plot); axis tight
title('u(0.5,t) vs t'); xlabel('t'); ylabel('u(0.5,t)')
subplot(1,2,2)
plot(t,err_plot); axis tight
title('Err u(0.5,t) vs t'); xlabel('t');
    ylabel('Err u(0.5,t)');
print -deps pde.eps; print -dps pde.ps; print -dpng pde.png
%
% Plot numerical solution in 3D perspective
figure(2);
colormap('Gray');
C=ones(n);
g=linspace(0,1,n); % For distance x
h1=waterfall(t,g,u',C);
axis('tight');
grid off
xlabel('t, time')
ylabel('x, distance')
zlabel('u(x,t)')
s1=sprintf('Diffusion Equation - MOL Solution');
sTmp=sprintf('u(x,0) = sin(\pi x/2)');
s2=sprintf('Initial condition: %s',sTmp);
title([s1], {s2}],'fontsize',12);
v=[0.8616   -0.5076    0.0000   -0.1770
    0.3712    0.6301    0.6820   -0.8417
    0.3462    0.5876   -0.7313    8.5590
         0         0         0     1.0000];
view(v);
rotate3d on;

```

Listing 2.1. Main program pde_1_main

We can note the following points about the main program given in Listing 2.1:

1. After declaring some parameters `global` so that they can be shared with other routines called via this main program, IC (2.2) is computed over a 21-point grid in x .

```
%
% Clear previous files
clear all
clc
%
% Parameters shared with the ODE routine
global ncall ndss
%
% Initial condition
n=21;
for i=1:n
    u0(i)=sin((pi/2.0)*(i-1)/(n-1));
end
```

2. The independent variable t is defined over the interval $0 \leq t \leq 2.5$; again, a 21-point grid is used.

```
%
% Independent variable for ODE integration
t0=0.0;
tf=2.5;
tout=linspace(t0,tf,n);
nout=n;
ncall=0;
```

3. The 21 ordinary differential equations (ODEs) are then integrated by a call to the Matlab integrator `ode15s`.

```
%
% ODE integration
mf=1;
reltol=1.0e-04; abstol=1.0e-04;
options=odeset('RelTol',reltol,'AbsTol',abstol);
if(mf==1) % explicit FDs
    [t,u]=ode15s(@pde_1,tout,u0,options); end
```

```

if(mf==2) ndss=4; % ndss = 2, 4, 6, 8 or 10 required
[t,u]=ode15s(@pde_2,tout,u0,options); end
if(mf==3) ndss=44; % ndss = 42, 44, 46, 48 or 50 required
[t,u]=ode15s(@pde_3,tout,u0,options); end

```

Three cases are programmed corresponding to $mf=1, 2, 3$, for which three different ODE routines, `pde_1`, `pde_2`, and `pde_3`, are called (these routines are discussed subsequently). The variable `ndss` refers to a library of differentiation routines for use in the MOL solution of PDEs; the use of `ndss` is illustrated in the subsequent discussion. Note that a stiff integrator, `ode15s`, was selected because the 21 ODEs are sufficiently stiff that a nonstiff integrator results in a large number of calls to the ODE routine.

4. Selected numerical results are stored for subsequent tabular and plotted output.
-

```

%
% Store numerical and analytical solutions, errors at  $x = 1/2$ 
n2=(n-1)/2.0+1;
sine=sin(pi/2.0*0.5);
for i=1:nout
    u_plot(i)=u(i,n2);
    u_anal(i)=exp(-pi^2/4.0*t(i))*sine;
    err_plot(i)=u_plot(i)-u_anal(i);
end

```

5. Selected tabular numerical output is displayed.
-

```

%
% Display selected output
fprintf('\n mf = %2d  abstol = %8.1e  reltol = %8.1e\n',...
        mf,abstol,reltol);
fprintf('\n  t   u(0.5,t)  u_anal(0.5,t)  err u(0.5,t)\n');
for i=1:5:nout
    fprintf('%6.3f%15.6f%15.6f%15.7f\n',...
            t(i),u_plot(i),u_anal(i),err_plot(i));
end
fprintf('\n ncall = %4d\n',ncall);

```

The output from this code is given in Table 2.1.

Table 2.1. Output for mf=1 from pde_1_main and pde_1

mf = 1 abstol = 1.0e-004 reltol = 1.0e-004			
t	u(0.5,t)	u_anal(0.5,t)	err u(0.5,t)
0.000	0.707107	0.707107	0.0000000
0.625	0.151387	0.151268	0.0001182
1.250	0.032370	0.032360	0.0000093
1.875	0.006894	0.006923	-0.0000283
2.500	0.001472	0.001481	-0.0000091
ncall = 85			

The output displayed in Table 2.1 indicates that the MOL solution agrees with the analytical solution to at least three significant figures. Also, ode15s calls the derivative routine only 85 times (in contrast with the nonstiff integrator ode45, which requires approximately 5,000–10,000 calls, clearly indicating the advantage of a stiff integrator for this problem).

- The MOL solution and its error (computed from the analytical solution) are plotted.

```
%
% Plot numerical solution and errors at x = 1/2
figure(1);
subplot(1,2,1)
plot(t,u_plot); axis tight
title('u(0.5,t) vs t'); xlabel('t'); ylabel('u(0.5,t)')
subplot(1,2,2)
plot(t,err_plot); axis tight
title('Err u(0.5,t) vs t'); xlabel('t'); ...
    ylabel('Err u(0.5,t)')
print -deps pde.eps; print -dps pde.ps; print -dpng pde.png
```

The plotted error output shown in Figure 2.1 indicates that the error in the MOL solution varied between approximately -3×10^{-5} and 16×10^{-5} , which is not quite within the error range specified in the program

```
reltol=1.0e-04; abstol=1.0e-04;
```

The fact that the error tolerances illustrated in Figure 2.1 were not satisfied does not necessarily mean that ode15s failed to adjust the integration

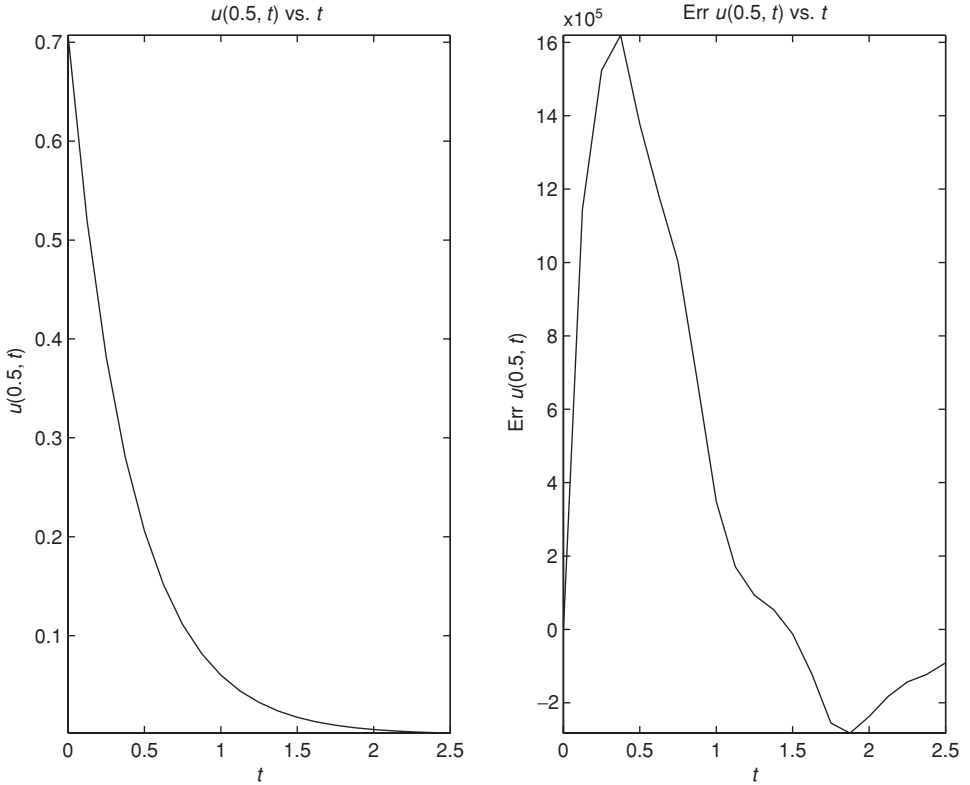


Figure 2.1. Two-dimensional graphical output from `pde_1_main`; `mf=1`

interval to meet these error tolerances. Rather, the error of approximately 1.6×10^{-4} is due to the limited accuracy of the second-order FD approximation of $\partial^2 u / \partial x^2$ programmed in `pde_1`. This conclusion is confirmed when the main program calls `pde_2` (for `mf=2`) or `pde_3` (for `mf=3`), as discussed subsequently; these two routines have FD approximations that are more accurate than in `pde_1`, so the errors fall below the specified tolerances.

This analysis indicates that two sources of errors result from the MOL solution of PDEs such as Eq. (2.1): (1) errors due to the integration in t (by `ode15s`) and (b) errors due to the approximation of the spatial derivatives such as $\partial^2 u / \partial x^2$ programmed in the derivative routine such as `pde_1`. In other words, we have to be attentive to integration errors in the *initial*- and *boundary-value independent variables*.

In summary, a comparison of the numerical and analytical solutions indicates that 21 grid points in x were not sufficient when using the second-order FDs in `pde_1`. However, in general, we will not have an analytical solution such as Eq. (2.5) to determine if the number of spatial grid points is adequate. In this case, some experimentation with the number of grid points, and the observation of the resulting solutions to infer the degree of accuracy or *spatial convergence*, may be required.

7. A 3D plot is also produced.

```
%
% Plot numerical solution in 3D perspective
figure(2);
colormap('Gray');
C=ones(n);
g=linspace(0,1,n); % For distance x
h1=waterfall(t,g,u',C);
axis('tight');
grid off
xlabel('t, time')
ylabel('x, distance')
zlabel('u(x,t)')
s1=sprintf('Diffusion Equation - MOL Solution');
sTmp=sprintf('u(x,0) = sin(\pi x/2)');
s2=sprintf('Initial condition: %s',sTmp);
title([s1, {s2}], 'fontsize', 12);
v=[0.8616    -0.5076     0.0000    -0.1770
    0.3712     0.6301     0.6820    -0.8417
    0.3462     0.5876    -0.7313     8.5590
         0         0         0      1.0000];
view(v);
rotate3d on;
```

The plotted output shown in Figure 2.2 clearly indicates the origin of the *lines in the method of lines* (also discussed in Chapter 1).

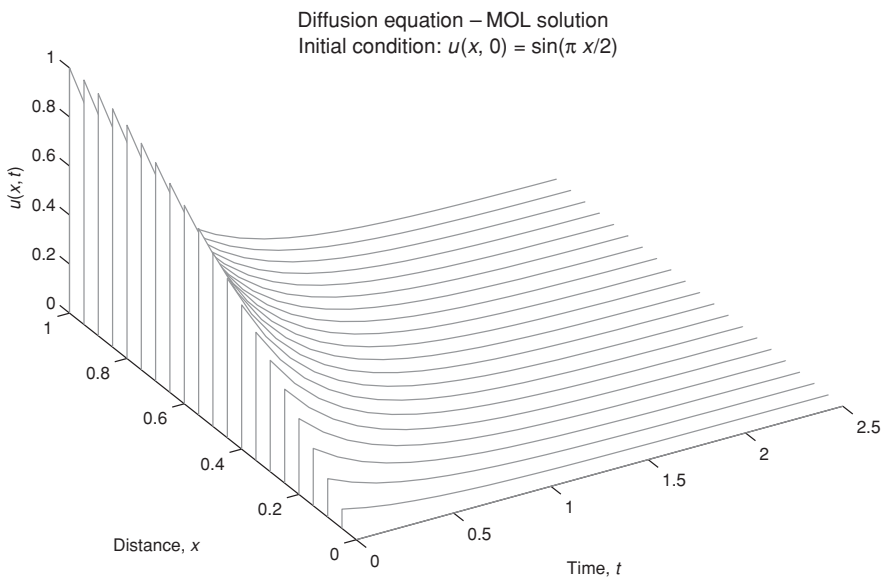


Figure 2.2. Three-dimensional graphical output from `pde_1_main; mf=1`

The programming of the approximating MOL/ODEs is in one of the three routines called by `ode15s`. We now consider each of these routines. For `mf=1`, `ode15s` calls function `pde_1` (see Listing 2.2).

```

function ut=pde_1(t,u)
%
% Problem parameters
global ncall
xl=0.0;
xu=1.0;
%
% PDE
n=length(u);
dx2=((xu-xl)/(n-1))^2;
for i=1:n
    if(i==1)      ut(i)=0.0;
    elseif(i==n)  ut(i)=2.0*(u(i-1)-u(i))/dx2;
    else          ut(i)=(u(i+1)-2.0*u(i)+u(i-1))/dx2;
    end
end
ut=ut';
%
% Increment calls to pde_1
ncall=ncall+1;

```

Listing 2.2. Routine `pde_1`

We can note the following points about `pde_1`:

1. After the call definition of the function, some problem parameters are defined.

```

function ut=pde_1(t,u)
%
% Problem parameters
global ncall
xl=0.0;
xu=1.0;

```

`xl` and `xu` could have also been set in the main program and passed to `pde_1` as global variables. The defining statement at the beginning of `pde_1` indicates

that the independent variable t and dependent variable vector u are inputs to `pde_1`, while the output is the vector of t derivatives, ut ; in other words, all of the n ODE derivatives in t must be defined in `pde_1`.

2. The FD approximation of Eq. (2.1) is then programmed.

```
%
% PDE
n=length(u);
dx2=((xu-xl)/(n-1))^2;
for i=1:n
    if(i==1)      ut(i)=0.0;
    elseif(i==n)  ut(i)=2.0*(u(i-1)-u(i))/dx2;
    else          ut(i)=(u(i+1)-2.0*u(i)+u(i-1))/dx2;
end
end
ut=ut';
```

The number of ODEs (21) is determined by the `length` command `n=length(u)`; so that the programming is general (the number of ODEs can easily be changed in the main program). The square of the FD interval, `dx2`, is then computed.

3. The MOL programming of the 21 ODEs is done in the `for` loop. For BC (2.3), the coding is

```
if(i==1) ut(i)=0.0;
```

since the value of $u(x = 0, t) = 0$ does not change after being set as an IC in the main program (and therefore its time derivative is zero).

4. For BC (2.4), the coding is

```
elseif(i==n) ut(i)=2.0*(u(i-1)-u(i))/dx2;
```

which follows directly from the FD approximation of BC (2.4),

$$u_x \approx \frac{u(i+1) - u(i-1)}{\Delta x} = 0$$

or with $i = n$,

$$u(n+1) = u(n-1)$$

Note that the *fictitious value* $u(n+1)$ can then be replaced in the ODE at $i = n$ by $u(n-1)$.

5. For the remaining interior points, the programming is

```
else ut(i)=(u(i+1)-2.0*u(i)+u(i-1))/dx2;
```

which follows from the FD approximation of the second derivative

$$u_{xx} \approx \frac{(u(i+1) - 2u(i) + u(i-1)))}{\Delta x^2}$$

6. Since the Matlab ODE integrators require a column vector of derivatives, a final transpose of ut is required.

```
ut=ut';
%
% Increment calls to pde_1
ncall=ncall+1;
```

Finally, the number of calls to `pde_1` is incremented so that at the end of the solution, the value of `ncall` displayed by the main program gives an indication of the computational effort required to produce the entire solution. The numerical and graphical output for this case ($mf=1$) was discussed previously.

For $mf=2$, function `pde_2` is called by `ode15s` (see Listing 2.3).

```
function ut=pde_2(t,u)
%
% Problem parameters
global ncall ndss
xl=0.0;
xu=1.0;
%
% BC at x = 0 (Dirichlet)
u(1)=0.0;
%
% Calculate ux
n=length(u);
if (ndss== 2) ux=dss002(xl,xu,n,u); % second order
elseif(ndss== 4) ux=dss004(xl,xu,n,u); % fourth order
```

```

elseif(ndss== 6) ux=dss006(xl,xu,n,u); % sixth order
elseif(ndss== 8) ux=dss008(xl,xu,n,u); % eighth order
elseif(ndss==10) ux=dss010(xl,xu,n,u); % tenth order
end
%
% BC at x = 1 (Neumann)
ux(n)=0.0;
%
% Calculate uxx
if (ndss== 2) uxx=dss002(xl,xu,n,ux); % second order
elseif(ndss== 4) uxx=dss004(xl,xu,n,ux); % fourth order
elseif(ndss== 6) uxx=dss006(xl,xu,n,ux); % sixth order
elseif(ndss== 8) uxx=dss008(xl,xu,n,ux); % eighth order
elseif(ndss==10) uxx=dss010(xl,xu,n,ux); % tenth order
end
%
% PDE
ut=uxx';
ut(1)=0.0;
%
% Increment calls to pde_2
ncall=ncall+1;

```

Listing 2.3. Routine pde_2

We can note the following points about pde_2:

1. The initial statements are the same as in pde_1. Then the Dirichlet BC at $x = 0$ is programmed.

```

%
% BC at x = 0 (Dirichlet)
u(1)=0.0;

```

Actually, the statement $u(1)=0.0$; has no effect since the dependent variables can only be changed through their derivatives, that is, $ut(1)$, in the ODE derivative routine. This code was included just to serve as a reminder of the BC at $x = 0$, which is programmed subsequently.

2. The first-order spatial derivative $\partial u / \partial x = u_x$ is then computed.

```

%
% Calculate ux
n=length(u);

```

```

if (ndss==2)      ux=dss002(xl,xu,n,u); % second order
elseif(ndss== 4) ux=dss004(xl,xu,n,u); % fourth order
elseif(ndss== 6) ux=dss006(xl,xu,n,u); % sixth order
elseif(ndss== 8) ux=dss008(xl,xu,n,u); % eighth order
elseif(ndss==10) ux=dss010(xl,xu,n,u); % tenth order
end

```

Five library routines, dss002 to dss010, are programmed that use second-order to tenth-order FD approximations, respectively. Since ndss=4 is specified in the main program, dss004 is used in the calculation of ux.

3. BC (2.4) is then applied, followed by the calculation of the second-order spatial derivative from the first-order spatial derivative.

```

%
% BC at x = 1 (Neumann)
ux(n)=0.0;
%
% Calculate uxx
if      (ndss== 2) uxx=dss002(xl,xu,n,ux); % second order
elseif(ndss== 4) uxx=dss004(xl,xu,n,ux); % fourth order
elseif(ndss== 6) uxx=dss006(xl,xu,n,ux); % sixth order
elseif(ndss== 8) uxx=dss008(xl,xu,n,ux); % eighth order
elseif(ndss==10) uxx=dss010(xl,xu,n,ux); % tenth order
end

```

Again, dss004 is called, which is the usual procedure (the order of the FD approximation is generally not changed in computing higher-order derivatives from lower-order derivatives, a process termed *stagewise differentiation*).

4. Finally, Eq. (2.1) is programmed and the Dirichlet BC at $x = 0$ (Eq. (2.3)) is applied.

```

%
% PDE
ut=uxx';
ut(1)=0.0;
%
% Increment calls to pde_2
ncall=ncall+1;

```

Note the similarity of the code to the PDE (Eq. (2.1)), and also the transpose required by ode15s.

Table 2.2. Output for mf=2 from pde_1_main and pde_2

mf = 2 abstol = 1.0e-004 reltol = 1.0e-004			
t	u(0.5,t)	u_anal(0.5,t)	err u(0.5,t)
0.000	0.707107	0.707107	0.0000000
0.625	0.151267	0.151268	-0.0000013
1.250	0.032318	0.032360	-0.0000418
1.875	0.006878	0.006923	-0.0000446
2.500	0.001467	0.001481	-0.0000138
ncall = 62			

The numerical output for this case (mf=2) is provided in Table 2.2. The plotted error output given in Figure 2.3 indicates that the error in the MOL solution varied between approximately -5×10^{-5} and 3.2×10^{-5} , which is within the error range specified in the program

```
reltol=1.0e-04; abstol=1.0e-04;
```

Thus, switching from the second-order FDs in pde_1 to fourth-order FDs in pde_2 reduced the *spatial truncation error* so that the MOL solution met the specified error tolerances.

For mf=3, function pde_3 is called by ode15s, as given in Listing 2.4.

```
function ut=pde_3(t,u)
%
% Problem parameters
global ncall ndss
xl=0.0;
xu=1.0;
%
% BC at x = 0
u(1)=0.0;
%
% BC at x = 1
n=length(u);
ux(n)=0.0;
%
% Calculate uxx
nl=1; % Dirichlet
nu=2; % Neumann
```

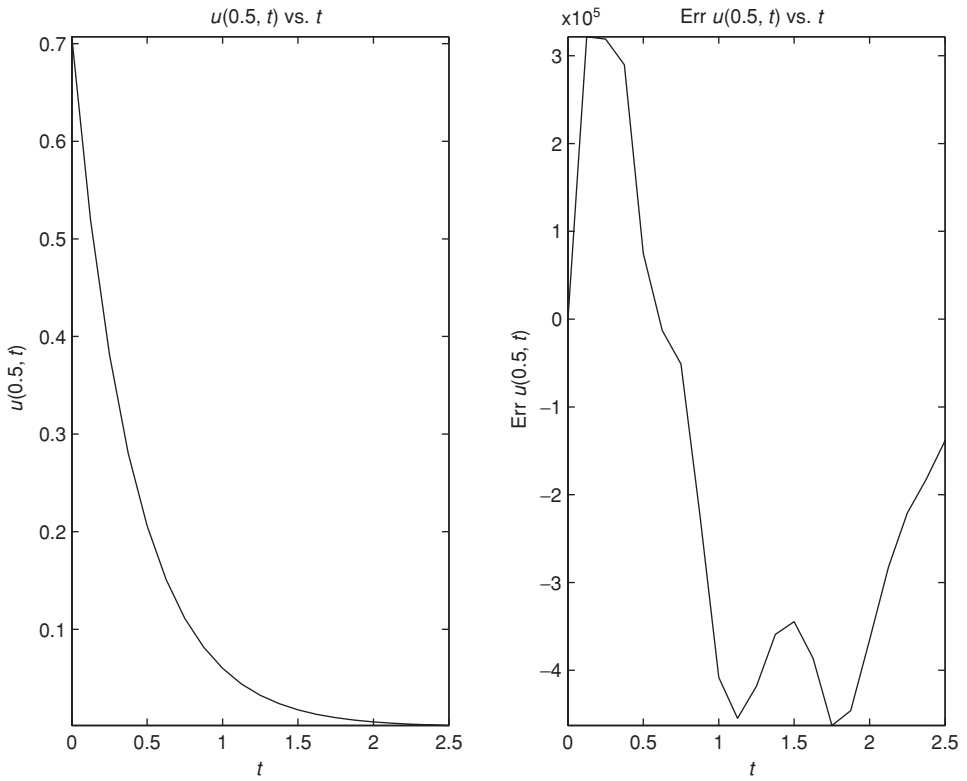


Figure 2.3. Two-dimensional graphical output from `pde_1_main; mf=2`

```

if      (ndss==42) uxx=dss042(xl,xu,n,u,ux,nl,nu);
        % second order
elseif(ndss==44) uxx=dss044(xl,xu,n,u,ux,nl,nu);
        % fourth order
elseif(ndss==46) uxx=dss046(xl,xu,n,u,ux,nl,nu);
        % sixth order
elseif(ndss==48) uxx=dss048(xl,xu,n,u,ux,nl,nu);
        % eighth order
elseif(ndss==50) uxx=dss050(xl,xu,n,u,ux,nl,nu);
        % tenth order
end
%
% PDE
ut=uxx';
ut(1)=0.0;
%
% Increment calls to pde_3
ncall=ncall+1;

```

Listing 2.4. Routine `pde_3`

We can note the following points about `pde_3`:

1. The initial statements are the same as in `pde_1`. Then the Dirichlet BC at $x = 0$ and the Neumann BC at $x = 1$ are programmed.

```
function ut=pde_3(t,u)
%
% Problem parameters
global ncall ndss
xl=0.0;
xu=1.0;
%
% BC at x = 0
u(1)=0.0;
%
% BC at x = 1
n=length(u);
ux(n)=0.0;
```

Again, the statement `u(1)=0.0;` has no effect (since the dependent variables can only be changed through their derivatives, i.e., `ut(1)`, in the ODE derivative routine). This code was included just to serve as a reminder of the BC at $x = 0$, which is programmed subsequently.

2. The second-order spatial derivative $\partial^2 u / \partial x^2 = u_{xx}$ is then computed.

```
%
% Calculate uxx
nl=1; % Dirichlet
nu=2; % Neumann
if (ndss==42)    uxx=dss042(xl,xu,n,u,ux,nl,nu); % second order
elseif(ndss==44) uxx=dss044(xl,xu,n,u,ux,nl,nu); % fourth order
elseif(ndss==46) uxx=dss046(xl,xu,n,u,ux,nl,nu); % sixth order
elseif(ndss==48) uxx=dss048(xl,xu,n,u,ux,nl,nu); % eighth order
elseif(ndss==50) uxx=dss050(xl,xu,n,u,ux,nl,nu); % tenth order
end
```

Five library routines, `dss042` to `dss050`, are programmed that use second-order to tenth-order FD approximations, respectively, for a second derivative. Since `ndss=44` is specified in the main program, `dss044` is used in the calculation of `uxx`. Also, these differentiation routines have two parameters that specify the type of BCs: (a) `nl=1` or `2` specifies a Dirichlet or a

Table 2.3. Output for mf=3 from pde_1_main and pde_3

mf = 3 abstol = 1.0e-004 reltol = 1.0e-004			
t	u(0.5,t)	u_anal(0.5,t)	err u(0.5,t)
0.000	0.707107	0.707107	0.0000000
0.625	0.151267	0.151268	-0.0000017
1.250	0.032318	0.032360	-0.0000420
1.875	0.006878	0.006923	-0.0000447
2.500	0.001467	0.001481	-0.0000138
ncall = 62			

Neumann BC, respectively, at the lower boundary value of $x = xl(= 0)$; in this case, BC (2.3) is Dirichlet, so $nl=1$; and (b) $nu=1$ or 2 specifies a Dirichlet or a Neumann BC, respectively, at the upper boundary value of $x = xu(= 1)$; in this case, BC (2.4) is Neumann, so $nu=2$.

3. Finally, Eq. (2.1) is programmed and the Dirichlet BC at $x = 0$ (Eq. (2.3)) is applied.

```
%
% PDE
ut=uxx';
ut(1)=0.0;
%
% Increment calls to pde_3
ncall=ncall+1;
```

Again, the transpose is required by ode15s.

The numerical output for this case (mf=3) is given in Table 2.3. The plotted error output shown in Figure 2.4 indicates that the error in the MOL solution varied between approximately -4.8×10^{-5} and 3.2×10^{-5} , which is within the error range specified in the program

```
reltol=1.0e-04; abstol=1.0e-04;
```

We conclude the example given in Figure 2.4 with the following observation: As the solution approaches steady state, $t \rightarrow \infty$, $u_t \rightarrow 0$, and from Eq. (2.1), $u_{xx} \rightarrow 0$.