THE APPLICATION OF THE MULTIGRID ALGORITHM TO THE SOLUTION OF STIFF ORDINARY DIFFERENTIAL EQUATIONS RESULTING FROM PARTIAL DIFFERENTIAL EQUATIONS (VOLUME 1)

Nabendra Parumasur

1992

The Application of the Multigrid Algorithm to the Solution of Stiff Ordinary Differential Equations Resulting from Partial Differential Equations

by

Nabendra Parumasur

Submitted in partial fulfilment of the

requirements for the degree of

Masters of Science,

in the

Department of Mathematics and Applied Mathematics ,

University of Natal (Durban)

1992

To my sister Sanjana

PREFACE

The research contained in this thesis[1] was carried out in the Department of Mathematics and Applied Mathematics, University of Natal, Durban, from April 1991 to December 1992, under the supervision of Professor J.R. Mika. These studies represent original work by the author and have not been submitted in any form to another University. Where use was made of the work of others it has been duly acknowledged in the text.

[1]This thesis was typeset by LaTeX , the TeX macro system of the American Mathematical Society.

ACKNOWLEDGEMENTS

I wish to thank my supervisor, Professor J.R. Mika, for his marvellous support and guidance rendered throughout this study. Many thanks are also due to Mr. J.M. Kozakiewicz for introductions to the multigrid solver (MGS). Finally, I am thankful for the following financial assistance I received during this study :

- FRD Master's Student Bursary - 1991
- DAAD Student Bursary - 1991
- FRD Studentship, funded from the Core Programme grant of Prof. JR Mika - 1992

ABSTRACT

We wish to apply the newly developed multigrid method [14] to the solution of ODEs resulting from the semi-discretization of time dependent PDEs by the method of lines. In particular, we consider the general form of two important PDE equations occuring in practice, viz. the nonlinear diffusion equation and the telegraph equation. Furthermore, we briefly examine a practical area, viz. atmospheric physics where we feel this method might be of significance. In order to offer the method to a wider range of PC users we present a computer program, called PDEMGS. The purpose of this program is to relieve the user of much of the expensive and time consuming effort involved in the solution of nonlinear PDEs. A wide variety of examples are given to demonstrate the usefulness of the multigrid method and the versatility of PDEMGS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Numerical Solution of ODEs and PDEs

The numerical solution of ordinary differential equations (ODEs) is a very wide and accomplished field of applied mathematics. In particular, the solution of the initial value problem (IVP) for the system of ODEs

$$\frac{dy}{dt} = f(t, y); \;\; y(0) = y_0; \tag{1.1}$$

has received much attention in the literature with regard to formulation of new methods and development of software [8, 21]. Stiff ODE systems[1] of

[1]Stiff ODE systems are formally defined in the next chapter, see page 9.

the form (1.1) have been recognized as being particularly common in applications and at the same time presents a formidable numerical challenge. However, in this modern computer era it is common knowledge that, with the advent of parallel computers, such systems may be adequately handled by means of newer types of explicit ODE methods (e.g. new Runge-Kutta methods). Such an opinion, has been voiced for instance, at a recent conference for computational ordinary differential equations [2]. On the other hand, with the increasing popularity of personal computers (PCs) and the excessive hardware costs involved with parallelism there still seems to be room for the development of special numerical methods capable of treating the IVP (1.1) when the ODE system is large and stiff. One such recent development is a method proposed by Kozakiewicz and Mika [14], called a *multigrid method*[2] [3]. Their method is designed to solve IVPs of the form (1.1) with a reduced computational effort, in particular, when 1.1 represent a stiff system of ODEs. Another possibility is the case when the time constants

[2]Incidentally, there is also a well established "multigrid method" which applies directly to PDEs [13]. The present multigrid method should not be confused with this "multigrid method".

[3]A formal definition of the multigrid method will be given in chapter 3, see page 41.

in (1.1) do not differ from each other much but are very large. In this case small time steps need to be used and if the time integration interval is long this could result in excessive computing times.

In direct competition with ODEs, the numerical solution of partial differential equations (PDEs) has also prompted a great deal of interest in numerical methods and software development over the years. Even in the present day a common trend for solving nonlinear PDEs is to use the *method of lines* (MOL) [2]. Basically, the MOL is a numerical technique for converting PDEs into a system of ODEs via discretization in the space-like independent variables. With this technique some classes of PDEs lead directly to large stiff systems of ODEs in the time-like independent variable. Recently, Mika and Scribani [18] reported preliminary results of the multigrid method applied to such systems of ODEs resulting from parabolic equations.

1.2 Object of Study

One purpose of this study is to check the feasibility of the multigrid method for the solution of physically realistic PDE problems using the MOL described above. For this purpose we consider the general form of two important

PDE problems that commonly model physical processes, viz. the nonlinear diffusion equation and the telegraph equation. Furthermore, we select a problem from the field of atmospherical physics to illustrate the applicablity of the method to problems arising in various fields of application.

Another goal of this study is to avail the method to a broader spectrum of PC users interested in solving large systems of PDEs occuring in scientific and engineering applications. In this regard, we provide a piece of computer software, called PDEMGS, which will serve as an interface allowing the newly developed multigrid method to be applied to PDEs. PDEMGS relieves the user of much of the duplicative work that is commonly encountered when discretizing PDEs by the MOL. Basically, PDEMGS requires the user to simply define his PDE problem and supply a spatial mesh (uniform) which is then used for the discretization. The software interface automatically forms and evaluates an approximating ODE system to the PDE system which can then be solved using the multigrid method.

This study is structured in the following way : Chapter 2 describes stiff ODE systems and methods for solving stiff ODE systems. We concentrate mainly on *linear multistep* methods and *Runge-Kutta* methods since these methods are most commonly used in codes for stiff ODE systems. Chapter

3 reviews the main results of the multigrid method and includes an improved multigrid algorithm for implementation on a PC. We also propose an *improved* multigrid algorithm for implementation on a PC. The solution of stiff ODEs resulting from PDEs is discussed in chapter 4. Chapter 5 contains an extensive investigation into the applicability of the multigrid method to PDEs. A variety of non-trivial numerical examples designed to indicate the versatility of the method and the ease of use of the software interface PDEMGS is presented here. Finally, Chapter 6 concludes this study with a general discussion of the material presented in this thesis.

Chapter 2

Stiff ODE Systems

2.1 Introduction

Stiff ordinary differential equations (ODEs) arise in many application areas, e.g. chemical kinetics, control theory, electronic circuit theory, biomathematics, etc. [1, 19]. The intent of this chapter is to give a brief review of stiff ODE systems.

We begin by considering the first order initial value problem in the following form

$$\dot{y} = f(t, y) \; ; \; t_0 \leq t \leq t_{final} \; ; \tag{2.1}$$

$$y(t_0) \;=\; y_0\,; \tag{2.2}$$

where

t is the time variable,

$y = [y^1, y^2, ..., y^N]^T$ is the solution vector,

N is the number of scalar first order ODEs,

$\dot{} = d/dt$ denotes differentiation with respect to t,

f is an N-vector function of y and t,

t_0 is the given initial value of time,

t_{final} is the final value of the interval of integration,

y_0 is the initial value N-vector.

As an example of a stiff ODE system of the form (2.1)-(2.2) we consider a system from chemical kinetics. Robertson's problem [12] is given by the following three nonlinear equations describing the reaction rates of three processes

$$\begin{aligned} \dot{y}^1 &= -0.04y^1 + 10^4 y^2 y^3\,; \\ \dot{y}^2 &= 0.04y^1 - 10^4 y^2 y^3 - 3 \times 10^7 y^2 y^2\,; \\ \dot{y}^3 &= 3 \times 10^7 y^2 y^2\,; \end{aligned} \tag{2.3}$$

$$y^1(0) = 1 \ ;$$

$$y^2(0) = 0 \ ; \tag{2.4}$$

$$y^3(0) = 0.$$

This reaction is interesting because the reaction coefficients (the constants on the right hand side of (2.3)) are widely varying (the smallest being of order 10^{-2} and the largest of order 10^7). Moreover, it can be trivially shown that the steady state solution for this problem is $y^1 \to 0$, $y^2 \to 0$, $y^3 \to 1$. By inspection of (2.3) we see that the dominant equation at equilibrium is $\dot{y}^2 = -10^4 y^2$, whose solution $y^2 = c_0 e^{-10^4 t}$, represents a strongly damped exponential regardless of the initial value. This situation is typical of stiff ODE systems.

Before proceeding to define what is meant by a stiff ODE system and how to solve such a system we would like to preview a few concepts for solving the ODE system (2.1)-(2.2), in general. Firstly most numerical methods for solving (2.1)-(2.2) (and the ones discussed in section 2.3) require us to compute a sequence of values $y_0, y_1, ..., y_n, ...$ which approximate the solution $y(t)$ at the discrete t values $t_0, t_1, ..., t_n, ...$ This process is called *discretization.* Since an approximation is being computed at each step errors are incurred.

The two main types of errors are *local* error and *global* error [5]. In general, we need to know if these errors have a large or small effect on the solution. To this end we define the concept of *absolute stability* [7] using the test problem $\dot{y} = \lambda y$, where λ is a complex constant[1]:

Definition 2.1 : The region of absolute stability is defined as the set of values of h and λ for which a perturbation in a single value y_n will produce a change in subsequent values which does not increase from step to step.

Now we are ready to define what is meant by stiffness and a stiff ODE system in general.

2.2 Stiffness and Stiff ODE Systems

Although it is difficult to give a precise mathematical definition to the concept of *stiffness*, it can be roughly said that a stiff ODE system is one having a general solution containing both very fast and very slow components. For example, consider the following prototypical stiff ODE given by

$$\dot{y} = -1000[y - e^{-t}] - [e^{-t}] \; ; \;\; 0 \leq t \leq t_{final} \; ; \tag{2.5}$$

[1]The constant λ corresponds either to $\frac{\partial f}{\partial y}$ for a scalar equation or the eigenvalues (which may be complex) of the Jacobian matrix for a system of equations

with the initial condition

$$y(0) = 0. \tag{2.6}$$

The exact solution to this problem is

$$y(t) = e^{-t} - e^{-1000t}\ ; \tag{2.7}$$

from which it is seen that the solution consists of a rapidly varying component e^{-1000t} and a slowly varying one e^{-t}. In general, the rapidly varying solution components are referred to as the *transient* solution components or *stiff* solution components and the slowly varying solution components are referred to as the *non-transient* solution components or *smooth* solution components. It is clear from (2.7) that the transient component e^{-1000t} will almost damp out at the end of a very small time interval (0,0.01) (see Figure 2.1). Usually the time interval for which this happens is called the *transient interval.*

If we use the fourth order Runge-Kutta method to solve (2.5) then we would have to use a small time-step in the transient interval in order to represent the rapidly decaying transient components accurately. In fact, for absolute stability we require $\lambda h \in (-2.78, 0)$, (see Lambert [15]), and since $\lambda = -1000$ this implies that $h < 0.00278$. Beyond the transient interval the value of the solution is essentially the slow component, and one might

expect to take a larger time-step. However, this is not the case, since the the presence of the transient component (although fully decayed) forces the use of excessively small time-steps. Thus the small time-step must be used throughout the interval, and if t_{final} is large this could be extremely costly. Had it been possible to change the initial value problem (2.5)-(2.6) for $t > 0.01$ so that the solution did not contain the transient, a step-size up to $h = 2.78$ would have been possible, although a smaller value would be required for accurate results. This leads to one definition of stiffness, viz. *stiffness occurs when the step-size is restricted by numerical stability rather than by accuracy*. Consider the general linear system with constant coefficients

$$\dot{y}(t) = Ay(t) + \Psi(t)\,; \tag{2.8}$$

where A is an $N \times N$ matrix whose eigenvalues λ_j, $j = 1, 2, ..., N$ are assumed to be distinct, and $\Psi(t)$ is a given vector function.

A more formal definition of stiffness given by Lambert [15] is that the system (2.8) is said to be *stiff* if :

$$\Re(\lambda_j) \;<\; 0 \text{ for } j = 1, 2, ..., N\,; \tag{2.9}$$

$$\underset{j}{\text{Max}}\,|\Re(\lambda_j)| \;\;/\;\; \underset{j}{\text{Min}}\,|\Re(\lambda_j)| = S >> 1. \tag{2.10}$$

The number S is called the *stiffness ratio*.

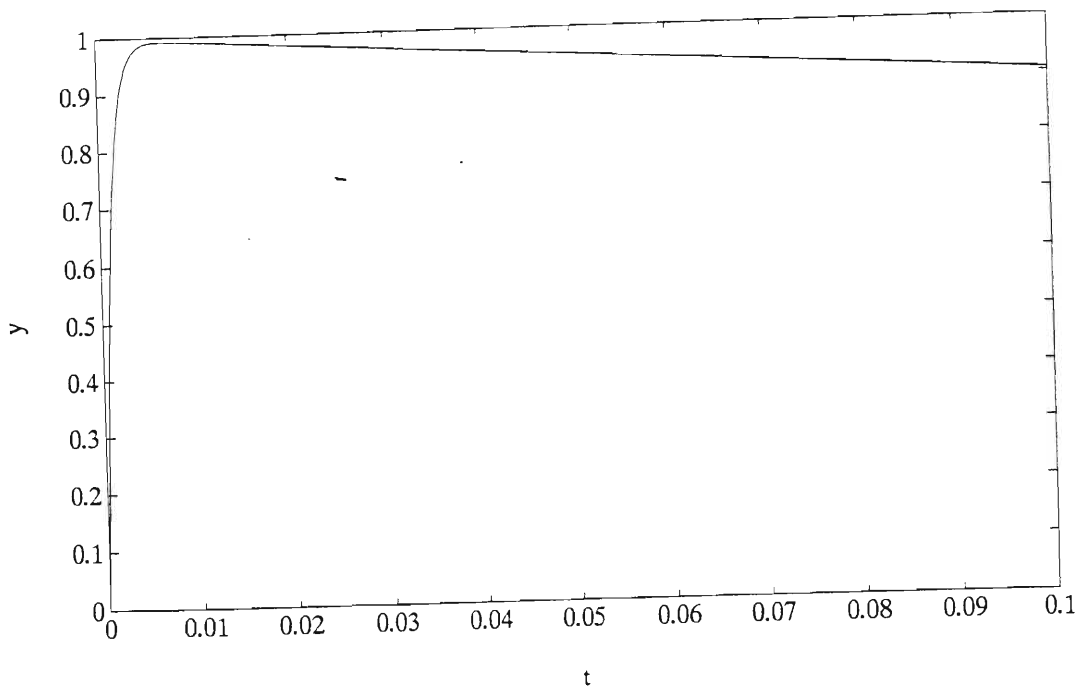


Figure 2.1: Graph of $y(t) = e^{-t} - e^{-1000t}$

Generalizing this definition we say that a nonlinear system (2.1)-(2.2) is stiff on the interval $0 \leq t \leq t_{final}$ if the eigenvalues $\lambda(t)$ of the Jacobian matrix satisfy (2.9) and (2.10) for all $0 \leq t \leq t_{final}$.

The definition (2.9)-(2.10) is perhaps too restrictive in requiring that all the eigenvalues should have negative real parts. In fact, stiffness in ODE systems really occurs because different terms in the solution vector behave on different scales, so that an ODE system is stiff if the Jacobian matrix has at least one eigenvalue whose real part is negative and large in magnitude, in comparison with the others. Since the independent variable is usually t,

an ODE system is stiff if its solution is slowly varying with respect to the most rapidly decaying component. This is essentially the definition given by Shampine and Gear [20].

To illustrate this point, we note that equation (2.5) has one eigenvalue $\frac{\partial \dot{y}}{\partial y} = -1000$. By equation (2.7) the solution beyond the transient interval is essentially e^{-t}. It is indeed slowly varying with respect to e^{-1000t}. By contrast, if we replace the forcing function by, say, $\sin(100t)$, the solution no longer varies slowly with respect to the transient, and the problem is not stiff according to the preceeding definition. The previous point is closely associated with a common misunderstanding of stiffness. Problems which have undamped high frequency oscillations in the solution, either due to forcing functions or to eigenvalues with large imaginary parts are generally not called stiff. One reason being that highly oscillatory problems require numerical methods which differ radically from those of stiff problems and as such deserve a special treatment.

In practice, it is usually difficult to determine if the system (2.8) is stiff or not. In fact, even a knowledge of the eigenvalues of the Jacobian matrix is not sufficient justification to ascertain if the system (2.8) is stiff or not. For

example, consider (2.8) with $\Psi(t) = 0$ and the matrix $A(t)$ defined by [6]

$$A(t) = \begin{pmatrix} -1 - 9\cos^2 6t + 6\sin 12t & 12\cos^2 6t + \frac{9}{2}\sin 12t \\ -12\sin^2 6t + \frac{9}{2}\sin 12t & -1 - 9\sin^2 6t - 6\sin 12t \end{pmatrix}. \quad (2.11)$$

The matrix $A(t)$ has constant eigenvalues -1 and -10, but the solution to the system (2.8) is

$$y = C_1 e^{2t} \begin{pmatrix} \cos 6t + 2\sin 6t \\ 2\cos 6t - \sin 6t \end{pmatrix} + C_2 e^{-13t} \begin{pmatrix} \sin 6t - 2\cos 6t \\ 2\sin 6t + \cos 6t \end{pmatrix}; \quad (2.12)$$

where C_1 and C_2 are arbitrary constants. Clearly the exponentials e^{-t} and e^{-10t} are not present in the solution.

In general, stiffness should be suspected if a fixed step-size method gives very poor results - or even diverges - for "reasonable" step-sizes , or if a solver with variable step-size cannot meet the accuracy requirement or is excessively slow (takes excessively small time-steps).

2.3 Solving Stiff ODE Systems

Some familiarity with numerical methods for solving stiff ODE systems is needed for applying the newly developed MGM [14] to the system (2.1) and for understanding the mechanics of available ODE software for solving such systems. For this reason we give here a brief description of two important

ODE methods, viz. *Linear multistep* methods and *Runge-Kutta* methods, which are most frequently used in ODE solvers. A more extensive survey can be found in Gupta, Sacks-Davis, and Tischer [8] and Seward, Fairweather, and Johnston [21].

2.3.1 Linear Multistep Methods

The class of *linear multistep methods* is very wide and varied and contains some of the most useful methods for solving stiff and non-stiff systems of ODEs of the form (2.1)-(2.2). A linear multistep method for solving the vector system of equations (2.1)-(2.2) can be defined by the formula

$$y_{n+1} = \sum_{i=1}^{K_1} \alpha_i y_{n+1-i} + h \sum_{i=0}^{K_2} \beta_i \dot{y}_{n+1-i} \; ; \tag{2.13}$$

where $\dot{y}_j = f(t_j, y_j)$ and h is a constant step-size in t. The coefficients α_i and β_i and the non-negative integer constants K_1 and K_2 are fixed for a given method. The number K=max(K_1, K_2) is called the step-number, i.e. the number of previous values, and (2.13) is referred to as a K-step method. In general, the most significant distinction among linear multistep formulas is between *explicit* and *implicit* methods. A method given by (2.13) is explicit if $\beta_0 = 0$ ($\dot{y}_{n+1}$ is absent) and is implicit otherwise. Equation (2.13) can also

be written in a form which uses varying step-sizes $h_{j+1} = t_{j+1} - t_j$, i.e.

$$y_{n+1} = \sum_{i=1}^{K_1} \alpha_{ni} y_{n+1-i} + h_{n+1} \sum_{i=0}^{K_2} \beta_{ni} \dot{y}_{n+1-i} \; ; \tag{2.14}$$

where the α and β coefficients now depend on the changing step-size.

The simplest examples of linear multistep methods are the *Euler* (forward Euler) method

$$y_{n+1} = y_n + h\dot{y}_n \; ; \tag{2.15}$$

and the *backward Euler* method

$$y_{n+1} = y_n + h\dot{y}_{n+1}. \tag{2.16}$$

These are one-step methods ($K = 1$) but are nevertheless included in the class of linear multistep methods as special cases, for convenience.

The class of methods most commonly used for stiff problems is the *Backward Differentiation Formulas* (BDFs). These methods follow from (2.13) (or (2.14)) by setting $K_2 = 0$ and $K_1 = q$. Thus the BDF of order q is given by either

$$y_{n+1} = \sum_{i=1}^{q} \alpha_i y_{n+1-i} + h\beta_0 \dot{y}_{n+1} \; ; \tag{2.17}$$

or the variable-step analogue. The case $q = 1$ is the backward Euler method.

On the other hand, the most commonly used non-stiff methods are the *Adams* methods. These methods are characterized by having only one term,

y_n, in the first sum in (2.13) or (2.14). Thus, the *explicit Adams* method of order q is given by either

$$y_{n+1} = y_n + h \sum_{i=1}^{q-1} \beta_i \dot{y}_{n+1-i} \; ; \tag{2.18}$$

or the variable-step analogue of this formula, and the *implicit Adams* method of order q is given by either

$$y_{n+1} = y_n + h \sum_{i=0}^{q-1} \beta_i \dot{y}_{n+1-i} \; ; \tag{2.19}$$

or its variable-step analogue. The familiar *Trapezoidal Rule* is obtained by setting $q = 2$ in (2.19), i.e.

$$y_{n+1} = y_n + (h/2)(\dot{y}_{n+1} + \dot{y}_n) \; ; \tag{2.20}$$

The term *order* is well defined for linear multistep methods. For (2.13), it is the largest integer q for which

$$y(t_{n+1}) - \sum_{i=1}^{K_1} \alpha_i y(t_{n+1-i}) - h \sum_{i=0}^{K_2} \beta_i \dot{y}(t_{n+1-i}) = O(h^{q+1}) \; ; \tag{2.21}$$

as $h \to 0$, when $y(t)$ is an arbitrary smooth function. It can be equivalently defined as the largest integer q for which the *local error* $y_{n+1} - y(t_{n+1}) = O(h^{q+1})$ when (2.13) is used to take one step with all past values exact ($y_{n+1-i} = y(t_{n+1-i})$ for $i \geq 1$).

For an implicit method, an algebraic system of the form

$$y_{n+1} = h\beta_0 f(t_{n+1}, y_{n+1}) + \sum_{i\geq 1} (\alpha_i y_{n+1-i} + h\beta_i \dot{y}_{n+1-i}) = h\beta_0 f(t_{n+1}, y_{n+1}) + a_n \; ; \tag{2.22}$$

where a_n is a known quantity, has to be solved for y_{n+1} at each step. The choice of methods for doing this may have a significant effect on the efficiency of the resulting algorithm or solver. As f is in general nonlinear, an iterative procedure of some type is ususlly used. The simplest such procedure is *functional* (or *fixed point*) iteration,

$$y_{n+1}^{(m+1)} = h\beta_0 f(t_{n+1}, y_{n+1}^{(m)}) + a_n \; ; \;\; (m = 0, 1, ...) \; ; \tag{2.23}$$

where $y_{n+1}^{(0)}$ is taken as an initial approximation to y_{n+1} obtained by an explicit method, e.g. Euler's method, and m denotes the iteration number. This works reasonably well for nonstiff problems, but for stiff problems it converges only when h is smaller than or comparable to the smallest time constant in the system, and such a restriction on h is unacceptable because of the excessive computer run time.

For stiff problems, the choices most often made for solving (2.22) are based on *Newton's* iteration method. For the problem in the form

$$F(y_{n+1}) \equiv y_{n+1} - h\beta_0 f(t_{n+1}, y_{n+1}) - a_n = 0 \; ; \tag{2.24}$$

Newton iteration takes the form

$$y_{n+1}^{(m+1)} = y_{n+1}^{(m)} - [F_y(y_{n+1}^{(m)})]^{-1} F(y_{n+1}^{(m)}) \tag{2.25}$$

or

$$[I - h\beta_0 f_y(y_{n+1}, y_{n+1}^{(m)})](y_{n+1}^{(m+1)} - y_{n+1}^{(m)}) = -F(y_{n+1}^{(m)}) \, . \tag{2.26}$$

Newton's method is in general a powerful method for solving nonlinear equations but has some considerable costs associated with it. For example, the Jacobian matrix f_y needs to be computed and stored at each step and also the solution of the $(N \times N)$ system for the correction $\Delta y = \bar{y}^{(m+1)} - \bar{y}^{(m)}$ is required. Both costs can be reduced significantly by modifying the iteration and paying close attention to the matrix structure. Firstly, since f_y usually varies slowly, little is lost in the rate of convergence by using a constant Jacobian matrix for more than one iteration in a single step. By the same reasoning, f_y can be kept fixed for several steps, provided a test can be made to decide when to recompute it. Newton's iteration procedure modified in this way is referred to as *modified* Newton iteration. Secondly, for a given problem, f_y may have a sparse matrix structure that could be exploited in order to reduce the costs of computing and storing it. Once a value of the

matrix

$$F_y = P = I - h\beta_0 f_y \; ; \tag{2.27}$$

has been computed (or approximated), suitable preprocessing of it (such as LU decomposition) can be done, depending on the structure assumed, so that the subsequent solutions of linear systems

$$P\Delta y = -F \; ; \tag{2.28}$$

are as inexpensive as possible.

2.3.2 Runge-Kutta Methods

The class of Runge-Kutta methods is also wide and varied. However, we give here only a brief summary of these methods since they are used less frequently in current codes for stiff ODEs.

Runge-Kutta methods are one-step methods, but involve intermediate stages within a step. They can be either explicit or implicit. The general r-stage *explicit Runge-Kutta* formula (RKF) for solving (2.1) can be written as

$$k_1 \;=\; hf(t_n, y_n) \; ; \tag{2.29}$$

$$k_i = hf(t_n + c_i h, y_n + \sum_{j=1}^{i-1} a_{ij} k_j) \; ; \quad (i = 2, 3, ..., r) \; ; \tag{2.30}$$

$$y_{n+1} = y_n + \sum_{i=1}^{r} b_i k_i \; ; \tag{2.31}$$

where the a_{ij}, b_i and c_i are constants satisfying $c_i = \sum_{j=1}^{i-1} a_{ij}$. For example, setting $r = 4$; $c_2 = c_3 = \frac{1}{2}$, $c_4 = 1$; $b_1 = b_4 = \frac{1}{6}, b_2 = b_3 = \frac{1}{3}$; $a_{21} = a_{32} = \frac{1}{2}, a_{43} = 1$; and all other constants equal to zero in (2.29)-(2.31) yields the explicit *fourth order Runge-Kutta* formula (RK4)

$$k_1 = hf(t_n, y_n) \; ; \tag{2.32}$$

$$k_2 = hf(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_1) \; ; \tag{2.33}$$

$$k_3 = hf(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_2) \; ; \tag{2.34}$$

$$k_4 = hf(t_n + h, y_n + k_3) \; ; \tag{2.35}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{2.36}$$

By means of rather lengthy calculations [7] one can determine the coefficient values in (2.29)-(2.31) which yield higher order formulas ($q \leq r$). On the other hand, it is often convenient to embed a method of order q within a method of order $q - 1$, thus making dynamic error control possible, based on the difference between the two y_{n+1} values .

For stiff problems, explicit RKF are not suitable, which means implicit

methods have to be used. The general r-stage *implicit* RKF can be written as

$$k_i = hf(t_n + c_i h, y_n + \sum_{j=1}^{r} a_{ij} k_j) \; ; \; (i = 1, 2, ..., r) \; ; \tag{2.37}$$

$$y_{n+1} = y_n + \sum_{i=1}^{r} b_i k_i. \tag{2.38}$$

Upon inspection of (2.37) we see that, since there are N ODEs in the system (2.1), we have to solve an algebraic system in rN unknowns. Many special cases have been considered to reduce this algebraic problem. We summarize a few cases below [9]:

(i) *Semi-implicit* RKF - in this case the matrix (a_{ij}) is lower triangular and so each k_i involves solving a system of N equations.

(ii) *Diagonally implicit* RKF - if Newton's method is used to solve (2.37) then a matrix of the form $I - ha_{ii}f_y$ appears. A further reduction in algebraic effort can be accomplished by taking all the a_{ii} equal- in this case the method is called diagonally implicit.

(iii) *Singly implicit* RKF - these formulas are characterized by the matrix (a_{ij}) having a single r-fold eigenvalue, thus permitting a linear transformation to an algebraic system that resembles that of the diagonally implicit case.

(iv) *Rosenbrock methods* (ROW) - here terms involving the Jacobian matrix

are added to (2.37), i.e.

$$k_i = hf_i(t_n + c_i h, y_n + \sum_{j=1}^{r} a_{ij} k_j) + hJ \sum_{j=1}^{i} d_{ij} k_i \; ; \tag{2.39}$$

were $J = \dfrac{\partial f}{\partial y}$ represents the Jacobian matrix or some approximation to it. The coefficients d_{ij} are chosen to optimize order and stability properties and setting all the d_{ii} equal reduces the required matrix computation to a minimum.

(v)*Mono-implicit* RKF - in this case the y argument of f in (2.30) contains a term involving a single unknown y_{n+1} - hence the name mono-implicit.

2.4 Stability of Stiff ODE Systems

When solving the system (2.8) numerically the step-size h must be chosen according to two criteria : firstly the transient components must be accurately represented and, secondly, $h\lambda_j$ must be within the region of absolute stability for all j such that $\Re(\lambda_j) < 0$. This latter condition may pose some difficulties when solving stiff systems. In view of this we review some results that call for a method to possess some 'adequate' region of absolute stability [15]

Definition 2.2 : A numerical method is said to be A-stable if its region of absolute stability contains the entire left half of the complex plane

$$\mathcal{C} = \{\lambda h : \Re(\lambda h) < 0\}$$

(see Figure 2.2 (a)).

Ideally one would like to use an A-stable method to solve stiff ODEs since no matter how large $\max_j |\Re(\lambda_j)|$, is, no stability restriction on h can result. Unfortunately A-stability is a very stringent requirement to ask of a numerical method as the following thereom shows [15]

Theorem 2.1 : (i) An explicit linear multistep method cannot be A-stable.

(ii) The order of an A-stable implicit linear multistep method cannot exceed two.

(iii) The most accurate A-stable linear multistep method is the trapezoidal rule.

In view of this less demanding stability definitions have been postulated. We list three such definitions here [15].

Definition 2.3 : A numerical method is said to be $A(\alpha)$-stable if its region of absolute stability contains the infinite wedge

$$W_\alpha = \{\lambda h : -\alpha < \pi - \arg(\lambda h) < \alpha\} \text{ for } \alpha \in (0, \pi/2);$$

(see Figure 2.2 (c)) and it is said to be $A(0)$-stable if it is $A(\alpha)$-stable for some sufficiently small $\alpha \in (0, \pi/2)$.

Definition 2.4 : A numerical method is said to be *stiffly*-stable if it is absolutely stable in the region $\{\lambda h : \Re(\lambda h) \leq -a\}$ and is accurate in the region $\{ \lambda h : -a < \Re(\lambda h) < b \, ; \; -c < \Im(\lambda h) \leq c \}$ where a, b, c are positive constants (see Figure 2.2 (b)).

Definition 2.5 : A numerical method is said to be A_0-stable if its region of absolute stability contains the whole negative real axis (see Figure 2.2 (d)).

While A-stability is a very stringent stability requirement, an even stronger stability condition can be defined for one-step methods. Consider the trapezoidal rule applied to the scalar test equation $\dot{y} = \lambda y$, λ a complex constant with $\Re(\lambda) < 0$. Then

$$\begin{aligned} r_1(\lambda h) &= \frac{y_{n+1}}{y_n} \\ &= \frac{1 + \frac{\lambda h}{2}}{1 - \frac{\lambda h}{2}} \rightarrow -1 \text{ as } \Re(\lambda h) \rightarrow -\infty. \end{aligned}$$

Thus any decaying components of the solution will do so slowly, in an oscillatory manner, for a large step-size. In contrast, the backward Euler method has

$$r_1(\lambda h) = \frac{1}{1 - \lambda h} \rightarrow 0 \text{ as } \Re(\lambda h) \rightarrow -\infty;$$

so decaying components will be rapidly damped out in a monotonic way. The above demonstration leads to the following definition [15]:

Definition 2.6 : An A-stable one-step numerical method is said to be L-stable, *strongly* A-stable, or *stiffly* A-stable if $r_1(\lambda h) \to 0$ as $\Re(\lambda h) \to -\infty$, when applied to the scalar test equation $\dot{y} = \lambda y$, λ being a complex constant with $\Re(\lambda) < 0$.

From the above it follows that

L-stability $\Rightarrow$ A-stability $\Rightarrow$ *Stiff*-stability $\Rightarrow$ $A(\alpha)$-stability $\Rightarrow$ A_0-stability.

2.5 Numerical Example

Consider the system [15]

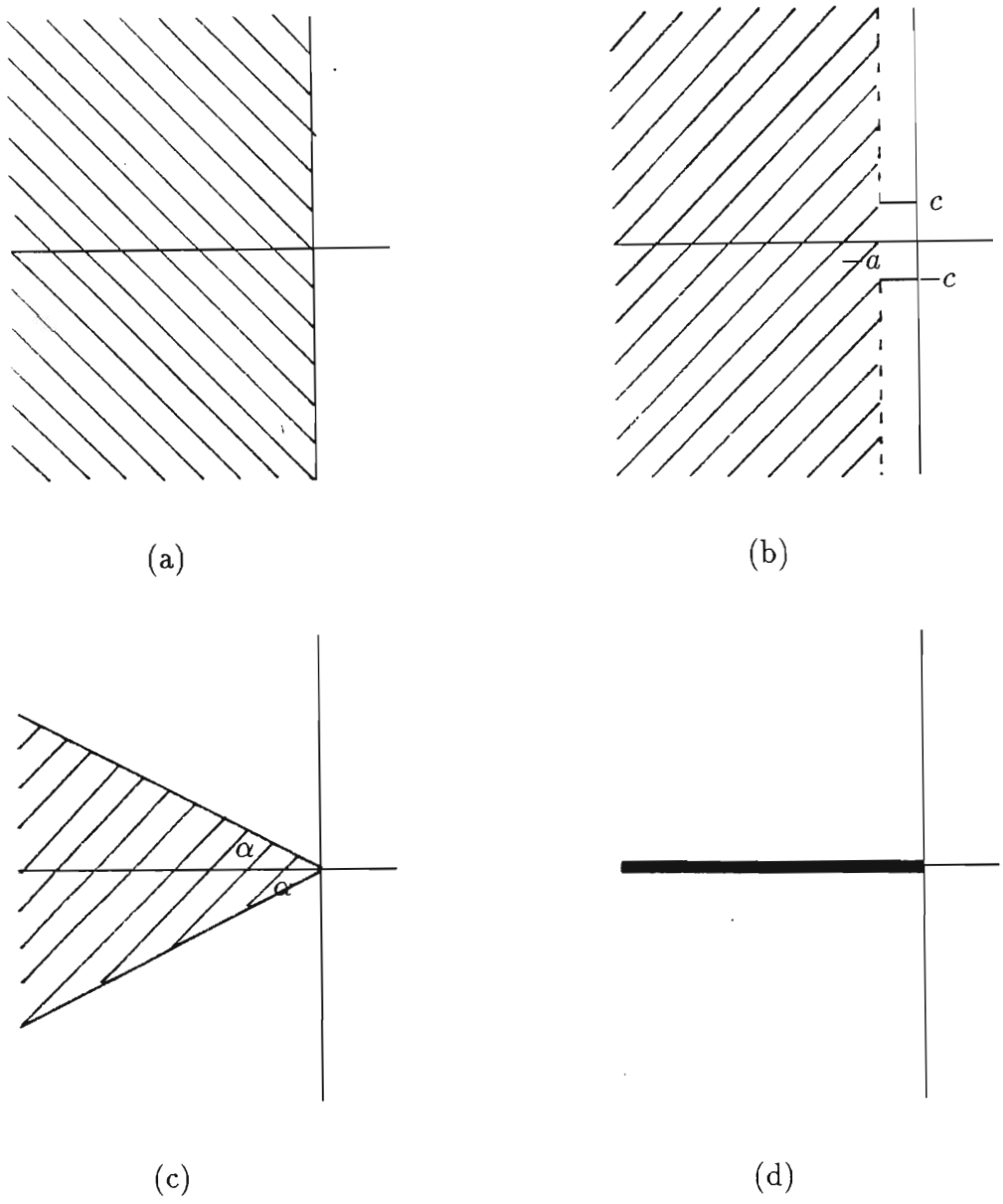
$$\dot{y} = Ay(t), \;\; y(0) = [1, 0, -1]^T, \;\; 0 \leq t \leq 1; \tag{2.40}$$

where

$$A = \begin{bmatrix} -21 & 19 & -20 \\ 19 & -21 & 20 \\ 40 & -40 & -40 \end{bmatrix}$$

The exact solution of the above system is

$$y^1(t) \;\; = \;\; \frac{1}{2}e^{-2t} + \frac{1}{2}e^{-40t}(\cos 40t + \sin 40t) \; ; \tag{2.41}$$



(a) (b)

(c) (d)

Figure 2.2: If the region of absolute stability includes the shaded region then the method is (a) A-stable, (b) *stiffly*-stable, (c) $A(\alpha)$-stable, (d) A_0-stable.

$$y^2(t) = \frac{1}{2}e^{-2t} - \frac{1}{2}e^{-40t}(\cos 40t + \sin 40t)\ ; \tag{2.42}$$

$$y^3(t) = -e^{-40t}(\cos 40t - \sin 40t). \tag{2.43}$$

where $y(t) = [y^1(t), y^2(t), y^3(t)]^T$

We solved this problem on a PC/AT using double precision arithmetic and the following solvers

(i) TRAP - stiff ODE solver which implements the trapezoidal rule (2.20) using a generalized Newton's iteration procedure. A starting approximation for the iteration was obtained using the explicit Euler method (2.15). The number of iterations per time-step was set equal to 10 ($m = 10$ in (2.25)). An exact Jacobian matrix was supplied in (2.27) and the resulting linear system (2.28) was solved using LU decomposition.

(ii) RK4 - non-stiff solver which implements the explicit fourth order Runge-Kutta formulas (2.32)-(2.36).

The numerical results giving the solution components y^1, y^2, y^3, at $t = 1s$, are given in Table 2.1. In the first instance, using h=0.1 we see that RK4 failed to solve this problem. TRAP, on the other hand, produced a solution, although not very accurate, for this case. Since the trapezoidal rule is A-stable this suggests that the choice of $h = 0.1$ causes $h\lambda$ to lie outside

the region of absolute stablity for RK4, and this is indeed the case. The eigenvalues λ of the Jacobian matrix A for this problem are -2, $-40 + 40i$, $-40 - 40i$. Recall that the region of absolute stability for RK4 is given by the open interval (-2.78,0) in the complex-plane. Hence for $h\lambda$ to lie within this interval for all three values of λ, h must satisfy $h < 0.0695$. In the second case, using $h = 0.01$, created no problems and both solvers produced reasonably accurate solutions. It may be worthwhile to note that in this case RK4 performs better than TRAP. This should be expected since TRAP is 2nd-order and RK4 is 4-th order.

Finally, we note that for this problem the stiffness ratio S (defined by (2.10)) equals 20 so that this represents a mildly stiff problem. Stiffness ratios of up to 50 000 are not uncommon in practice [4].

SOLVER	STEP-SIZE	y^1	y^2	y^3
EXACT SOLUTION		6.7667E-2	6.7667E-2	0.0000
RK4	0.1	-2.6717E13	-2.6717E1	1.9733E14
	0.01	6.7667E-2	6.7667E-2	0.0000
TRAP	0.1	6.7306E-2	6.7279E-2	0.0000
	0.01	6.7663E-2	6.7663E-2	0.0000

Table 2.1: Numerical Results for Example 2.5.

Chapter 3

Multigrid Method for Stiff ODE Systems

3.1 Introduction

In many practical applications large systems of ODEs need to be solved. These systems are generally recognized as being stiff since they contain components with highly different time constants. We have seen in the previous chapter that the conventional way of handling such systems is to use a stiff ODE solver based on e.g. BDF or Runge-Kutta methods.

The common trend nowadays for handling such ODE systems is to use

methods which are suitable for parallel computation on large main frame computers (e.g. splitting of ODE system into several sub-systems and integrating each sub-system separately over a certain time interval). Such a view has emerged, for example, from the recent conferences on Computational Ordinary Differential Equations [1, 2].

However, due to an increasing popularity for personal computers and the excessive costs of large main frame computers, it still seems highly advantageous to develop special numerical techniques that are capable of exploiting certain features of the system being solved without resorting to large main frame computers. For example, when simulating the time dependent behaviour of nuclear reactors, an extremely large system of ODEs results. To alleviate the situation the so called *quasistatic* method was developed and used successfully in nuclear reactor calculations in the early sixties. Originally the method was based on purely heuristic grounds until its mathematical foundations were given by Mika [17].

Recently, Kozakiewicz and Mika [14] proposed a *multigrid method*, for the solution of ODEs, based on the ideas used in developing the quasistatic method. In this chapter we review some of the results of the multigrid method and in the process propose an improved algorithm for implementing

the method on a PC.

3.2 Review of the Multigrid Method

Consider the singularly perturbed initial value system of ODEs in the following form

$$\epsilon \frac{dx}{dt} = f(x,t); \quad t \geq 0; \tag{3.1}$$

$$x(0) = q; \tag{3.2}$$

where $x(t)$, q and f are n dimensional vectors and ϵ is a small positive parameter. Firstly, we note that when integrating (3.1)-(3.2) the time step would be severely restricted by numerical stability, due to the presence of the small constant ϵ. So when we need to integrate (3.1) over a long time interval, being forced to choose very small time steps means that the integration process is excessively slow. However physical systems modelled by equations of the form (3.1) might have the property that the shape of the solution vector $x(t)$ changes rapidly with respect to t only in a very short time interval, called the transient interval. Past the transient interval the solution $x(t)$ changes very slowly with respect to t, i.e. the solution vector $x(t)$ assumes

a relatively smooth shape for the rest of the time interval. In such cases the solution vector $x(t)$ may be approximately expressed as the product

$$x(t) = \varphi(t)v(t); \tag{3.3}$$

where $\varphi(t)$ is a scalar function which changes very rapidly with respect to t and $v(t)$ is a vector function which can be made to vary slowly with respect to t by imposing certain conditions on it. We will call the function $\varphi(t)$ the *amplitude function* and the vector $v(t)$ the *shape vector* of the solution vector $x(t)$.

In the next section we will consider a more general case in which the solution vector $x(t)$ may be separated into several sub-vectors so that (3.3) turns into a sum of products of several amplitude functions and shape vectors. We will discuss the conditions that have to be imposed on the shape vectors v and derive the necessary equations for the amplitude functions and shape vectors.

3.2.1 Amplitude and Shape Equations

When solving certain systems of ODEs the properties of the system may suggest a partition of the solution vector into several sub-vectors. We will

be interested in three particular cases :

(i) When, solving stiff ODE systems, it may be possible to combine separately the fast and slow components into one or more groups (see example in section 2.1)

(ii) When, reducing a second order system of ODEs into a first order system of ODEs, the solution vector can be expressed in a partitioned form (see section 5.6).

(iii) The solution vector of a discretized version of a system of n time dependent PDEs can be represented in a partitioned form consisting of n subvectors (see section 5.2).

When it is possible to partition the solution vector into several subvectors, we write the IVP for a system of ODEs in the following form:

$$\frac{dx}{dt} = f(x,t), \quad x(0) = q, \quad 0 < t \leq t_0, \quad t_0 > 0; \tag{3.4}$$

we shall introduce the following notation:

$$\begin{aligned} x(t) &= (x_1, x_2, ..., x_m)^T; \\ & \\ x_k &= (x_k^1, x_k^2, ..., x_k^{n_k})^T; \; k = 1, 2, ..., m; \end{aligned} \tag{3.5}$$

so that x is a function from $[0, t_0]$ into R^n, where $n = \sum_{k=1}^{m} n_k$. Similarly $f : D \times [0, t_0] \to R^n$ where

$$D = \{a \in R^n : \|a - q\| \leq d\}; \tag{3.6}$$

with $\|.\|$ any of the equivalent norms in R^n,$d > 0$ and q is the initial vector with the same structure as that of x given in (3.5).

We have the following well known existence theorem which we will state without proof.

Theorem 3.1. [14] The IVP (3.4) has a unique solution on the interval $[0, t_1]$ where $t_1 = \min\{t_0, \frac{d}{M}\}$ if

(i) f is continous with respect to all arguments and Lipschitz continous with respect to x on $D \times [0, t_0]$;

(ii) $\sup\{\|f(x,t)\|;\ x \in D;\ t \in [0, t_0]\} = M < \infty$.

In a manner similar to (3.3) we express each of the sub-vectors x_k in the form

$$x_k(t) = \varphi_k(t) v_k(t); \tag{3.7}$$

where φ_k are the amplitude functions and $v_k = (v_k^1, v_k^2, ..., v_k^{n_k})$ are the shape vectors. We now require that the shape vectors v_k depend on time as weakly as possible. To express that, we will make the basic assumption that the

projections of v_k onto a one-dimensional sub-space of R^{n_k} are independent of t:

Let $P_k : R^{n_k} \rightarrow R$ be a projection operator defined by the usual scalar product as

$$P_k a = \langle p_k, a \rangle \; ; \; a, p_k \in R^{n_k}; \tag{3.8}$$

then we require that

$$(P_k v_k)(t) = 1; \;\; t \in [0, t_1]. \tag{3.9}$$

From equation (3.7) we get

$$\varphi_k(t) = (P_k x_k)(t); \;\; t \in [0, t_1]. \tag{3.10}$$

The choice of the projection operator P_k usually depends on the properties of the original system (3.4) being solved and in most cases the simple averaging operator might be adequate.

Theorem 3.2. [14] The form (3.7) is unique on an interval $[0, t_2]$, where $0 < t_2 \leq t_1$, if

$$\varphi_k(t) \geq \mu_k > 0; \;\; t \in [0, t_2]; \;\; k = 1, 2, ..., n_k. \tag{3.11}$$

Proof

By (3.11), x_k is not a zero vector. Suppose that

$$x_k(t) = \bar{\varphi}_k(t)\bar{v}_k(t) = \varphi_k(t)v_k(t); \quad t \in [0, t_2]. \tag{3.12}$$

Applying the projection operator P_k to this equation and using equation (3.9) gives

$$\bar{\varphi}_k(t) = \varphi_k(t) > 0; \quad t \in [0, t_2]. \tag{3.13}$$

Then from (3.12)

$$v_k(t) = \bar{v}_k(t) \neq 0; \quad t \in [0, t_2]. \tag{3.14}$$

Since $\varphi_k(t)$ and $v_k(t)$ are both nonzero, then (3.13) and (3.14) prove uniqueness.□

Consider the system (3.4) in the extended form

$$\frac{dx_k}{dt} = f_k(x_1, x_2, ..., x_m; t); \quad x_k(0) = q_k; \quad k = 1, 2, ..., m. \tag{3.15}$$

Substituting x_k in the form given by equation (3.7) into this equation yields

$$\frac{d\varphi_k}{dt}v_k + \varphi_k\frac{dv_k}{dt} = f_k(\varphi_1 v_1, \varphi_2 v_2, ..., \varphi_m v_m; t). \tag{3.16}$$

Now applying the projection operators P_k $(k = 1, 2, ..., m)$ to both sides of equation (3.16) and noting that

$$P_k\frac{dv_k}{dt} = \frac{d}{dt}(P_k v_k) = \frac{d}{dt}(1) = 0;$$

yields the following system of scalar equations for the amplitude functions

$$\frac{d\varphi_k}{dt} = \gamma_k(\varphi_1, \varphi_2, ..., \varphi_m; v_1, v_2, ..., v_m; t); \quad k = 1, 2, ..., m; \tag{3.17}$$

where

$$\gamma_k(\varphi_1, \varphi_2, ..., \varphi_m; v_1, v_2, ..., v_m; t) = (P_k f_k)(\varphi_1 v_1, \varphi_2 v_2, ..., \varphi_m v_m; t).$$

As a natural initial condition for (3.17) we take

$$\varphi_k(0) = \alpha_k; \quad k = 1, 2, ..., m; \tag{3.18}$$

where $\alpha_k = P_k q_k$. To satisfy the uniqueness condition (3.11) for some $t_2 > 0$ we have to assume that

$$\varphi_k(0) = \alpha_k \neq 0; \quad k = 1, 2, ..., m. \tag{3.19}$$

If the condition (3.11) holds then we may eliminate $\frac{d\varphi_k}{dt}$ from (3.16) to obtain a system of equations for the shape vectors

$$\frac{dv_k}{dt} = \frac{1}{\varphi_k}(f_k - \gamma_k v_k); \quad k = 1, 2, ..., m; \tag{3.20}$$

with initial conditions

$$v_k(0) = \frac{1}{\alpha_k} q_k; \quad k = 1, 2, ..., m. \tag{3.21}$$

In a manner similar to (3.4), we may write the system of equations (3.17)-(3.18) and (3.20)-(3.21) in vector form as

$$\frac{dz}{dt} = h(z,t); \;\; z(0) = r; \tag{3.22}$$

where

$$\begin{aligned} z(t) &= (\varphi_1, \varphi_2, ..., \varphi_m, v_1, v_2, ..., v_m) \\ v_k &= (v_k^1, ..., v_k^{n_k}), \;\; k = 1, 2, ..., m \end{aligned}$$

so that $z : [0, t_2] \rightarrow R^{n+m}$ where $n = \sum_{k=1}^{m} n_k$.

Similarly $h : D_1 \times [0, t_2] \rightarrow R^{n+m}$ where

$$D_1 = \{b \in R^{n+m} : \|b - r\| \leq d_1\}, \tag{3.23}$$

where $d_1 > 0$, and $\|.\|$ is a norm in R^{n+m} related to the norm chosen in R^n. Finally r is the initial vector with the same structure as that of z, and is given by

$$r = (\alpha_1, \alpha_2, ..., \alpha_m, \frac{1}{\alpha_2} q_1, \frac{1}{\alpha_2} q_2, ..., \frac{1}{\alpha_m} q_m). \tag{3.24}$$

Theorem 3.3. [14] The system of ODEs (3.22) has a unique solution on an interval $[0, t_3]$ where $0 < t_3 \leq t_2$ with t_2 defined by (3.11).

Proof

We will show that the system (3.22) satisfies the conditions (i) and (ii) of theorem 3.1. The first condition follows immediately since the projection operators P_k , applied to the original system of ODEs (3.4), do not change the smoothness properties of the functions contained in the system.

On the other hand, dividing by φ_k in (3.20) may change the constant M defined in condition (ii), thereby resulting in a new constant M_1 defined by

$$M_1 = \sup\{\|h(z,t)\|;\ z \in D_1;\ t \in [0,t_2]\} < \infty$$

Thus, on the basis of condition (i) and (ii), the system of ODEs (3.22) will have a unique solution on an interval $[0,t_3]$, where $0 < t_3 \leq t_2$ and t_2 is defined by (3.11). □

Finally we state the following theorem without proof:

Theorem 3.4. [14] The solutions of (3.4) and (3.22) are equivalent to each other on an interval $[0,t_3]$, with t_3 defined as in the preceeding theorem

The above procedure for reducing the original system of ODEs (3.4) into equivalent systems for the amplitude equations (3.17)-(3.18) and the shape vectors (3.20)-(3.21), might be called a *multigrid method* [14]. The advantage of this multigrid method is that one can use different step-sizes for solving the amplitude and shape equations. Obviously the multigrid method is applica-

ble only if the system (3.20)-(3.21) is much easier to solve than the original system (3.4), otherwise it would be senseless to solve m additional equations (3.17).

A numerical procedure for solving the amplitude and shape equations is given in the next section.

3.3 Multigrid Numerical Procedure

In the previous section it was shown that over a certain time interval the original system of ODEs (3.15) is equivalent to the system for the amplitude functions

$$\frac{d\varphi_k}{dt} = \gamma_k; \quad \varphi_k(0) = \alpha_k; \quad k = 1, 2, ..., m; \tag{3.25}$$

and the system for the shape vectors

$$\frac{dv_k}{dt} = \frac{1}{\varphi_k}(f_k - \gamma_k v_k); \quad v_k(0) = \frac{1}{\alpha_k} q_k; \quad k = 1, 2, ..., m; \tag{3.26}$$

where the functions φ_k and the vectors v_k are related to the solution vector x_k by the relationship (3.7).

Equation (3.25) is a system of scalar equations whose solutions φ_k are rapidly changing functions of time. One aim of using the multigrid method

to solve (3.15) is (hopefully) to concentrate all the "nasty" features of the system into the amplitude equations, when these nasty features exist. If, for example, (3.25) is stiff then it may be necessary to use a stiff solver to integrate (3.25). On the other hand, (3.26) is a system of vector equations whose solution components v_k change very slowly with respect to time. To solve (3.26) it is admissible to use a simple explicit method with relatively large step-size.

The stiff and non-stiff solvers are combined into a single multigrid solver, (called MGS [14]), for solving the original system of ODEs in the form (3.15).

An algorithm for implementing the multigrid method on a PC was given by Kozakiewicz and Mika [14]. Here we propose an improved[1] version of this algorithm.

3.3.1 Improved Multigrid Algorithm

(i) Begin the first cycle at $t = 0$, (I = 0), (see Figure 3.1) by solving the system for the amplitude equation (3.25) over a fixed number L of steps of size h_i, using the initial values φ_k^0 and v_k^0. The functions γ_k in the amplitude

[1]The algorithm has been improved by introducing inner iterations in each cycle of the multigrid algorithm and using improved starting values in each stage in a cycle.

equations are calculated with v_k fixed at their initial values.

(ii) Solve the system for the shape vectors (3.26) over a single step having size $H = \sum_{i=1}^{L} h_i$ using as initial values v_k^0 and a linear approximation $\varphi_k^{av} = (\varphi_k^0 + \varphi_k^1)/2$, which is simply the average values of φ_k at the end points $t = 0$ and $t = H$ (see figure 3.1). L is prescribed by the user.

(iii) Repeat the calculations starting at $t = 0$, (I = 1) using the average values $v_k^{av} = (v_k^0 + v_k^1)/2$ and the value φ_k^0 as starting values for the first stage and the values $(\varphi_k^{av} = \varphi_k^1 + \varphi_k^2)/2$ and v_k^0 as starting values for the second stage.

(iv) At the end of the second stage we have improved values φ_k^2 and v_k^2 which serve as initial values for the next cycle.

3.4 Structure and Use of MGS

The structure of MGS consists of a main program, a driver routine for setting up and solving the amplitude and shape equations and a routine which defines the original ODE system. The typical overall structure of MGS is illustrated in Figure 3.2. We give a brief description of each of the principal routines used in MGS:

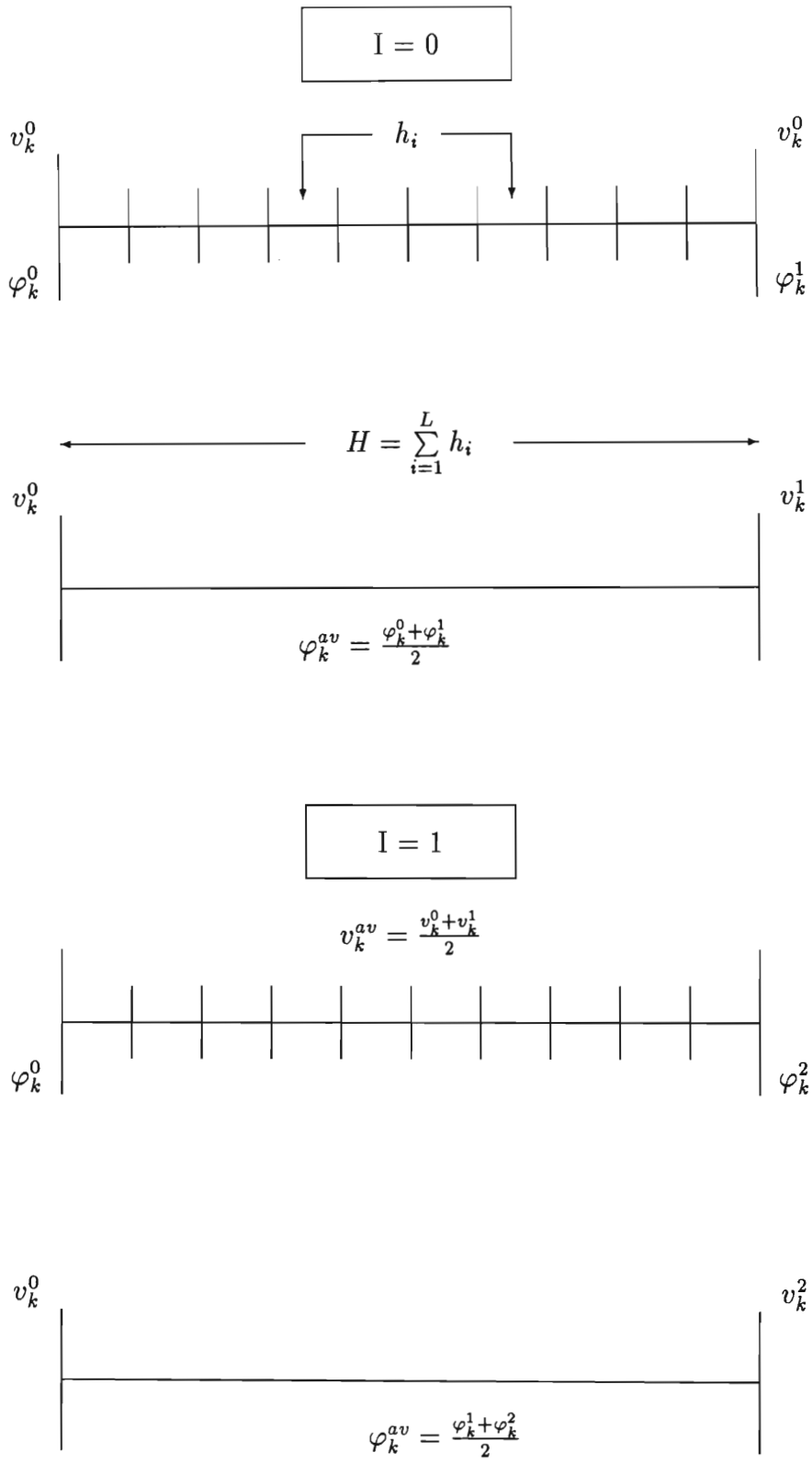


Figure 3.1: Inner iteration for a multigrid cycle: I=iteration number.

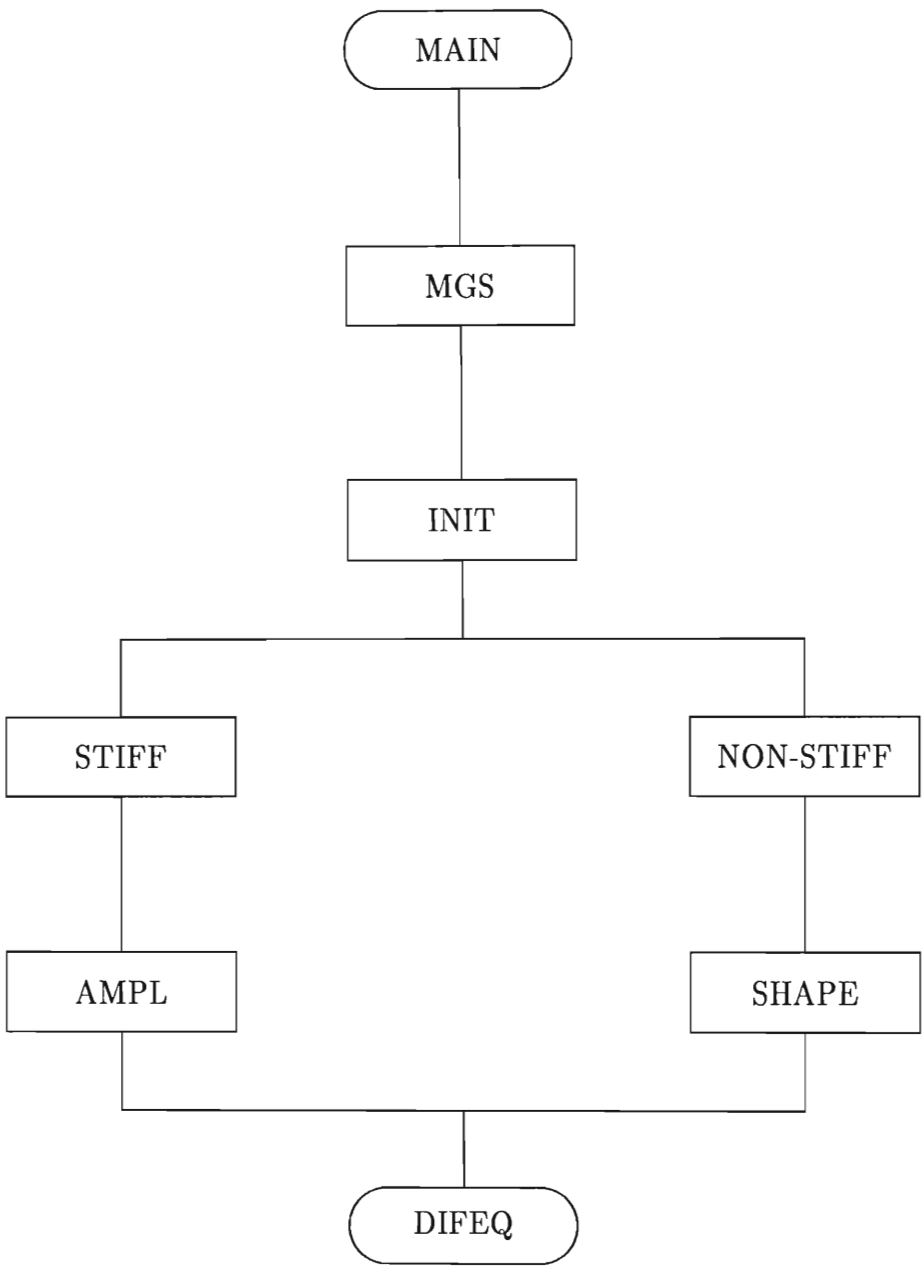


Figure 3.2: Structure of MGS: Oval shapes indicate user-defined routines.

MAIN : User-defined main program that is responsible for setting parameters and allocating storage for variables used in MGS. The parameters may include the initial time, number of amplitude equations, number of shape equations, number of time-steps in amplitude equations for one step in shape equations and the user-defined parameters of the stiff ODE solver (e.g. initial time-step, error tolerance, etc.). The main program also serves to call the driver routine MGS and sets up the initial conditions of the original system of ODEs (3.4). Finally the main program calculates the computing time for execution of MGS and provides for output times and printing of results.

MGS : Driver routine for solver. On the first call to the solver this routine performs initialization tasks such as evaluation of the projection operators and calling of INIT. After initialization is complete, MGS makes repeated calls to the core integrator routine of the stiff ODE solver in order to advance the time variable of the amplitude equations a single step h_i at a time until the user-defined L (number of time steps in amplitude equations for one step in shape equations) is reached. Next, MGS makes one single call to the non-stiff ODE solver in order to advance the time variable of the shape equations by a single step $H = \sum_{i=1}^{L} h_i$.

INIT : Routine containing initial conditions of the amplitude and shape equa-

tions.

STIFF : Stiff ODE solver for integrating amplitude equations. Solver should implement methods that have good stability properties (see section 2.3).

Non-stiff : Non-stiff ODE solver for integrating shape equations. A solver based on a simple explicit method should suffice.

AMPL : Routine for evaluating the right-hand side of the amplitude equations.

SHAPE : Routine for evaluating the right-hand side of the shape equations.

DIFEQ : User-defined routine containing the function f of the original system of ODEs represented in the form (3.15).

In order to use MGS, the user needs to supply two routines, viz. MAIN and DIFEQ. The subsequent calculations of the right-hand sides and initial conditions of (3.25) and (3.26) are done automatically by MGS.

3.5 Numerical Example

We consider the following system of $2n$ coupled linear ODEs for which the extended form (3.15) is [14]

$$\frac{dx_1^j}{dt} = (A+j)x_1^j + x_1^{j+1} + 0.1\left(x_2^{j+1}\right); \quad x_1^j(0) = 1; \tag{3.27}$$

$$\frac{dx_2^j}{dt} = x_2^{j+1} + 0.1\left(x_1^{j+1}\right); \qquad x_2^j(0) = 1; \tag{3.28}$$

$j = 1, 2, ..., n$. We use here the convention that $x_1^{n+1} = x_1^1$ and $x_2^{n+1} = x_2^1$. The above system (3.27)-(3.28) becomes stiff if A is large in magnitude. The sub-vector x_1 contains the fast components and the sub-vector x_2 contains the slow components of the solution vector. In our calculations we choose $A = 1000$ and $n = 30$.

We solved this problem on a PC/AT (speed-10MHz) using double precision arithmetic and the following multigrid solvers:

(i) MGS1 : Same as MGS [14], except that in this case the classic fourth order Runge-Kutta method (RK4) with fixed step-size is used to integrate the amplitude equations and the shape equations are treated using the simple second order Runge-Kutta method[2] (RK2) with fixed step-size. The number of time-steps for one step in shape vectors was set as $L = 10$, and a step-size of $H = 1 \times 10^{-4}$ was used for the integration of the amplitude equations.

(iii) MGS2 : Same as (i), except that in this case we implement the improved multigrid algorithm for solving the amplitude and shape equations, discussed in section 3.3.1.

[2]Refers to Heun's method [5]

The numerical results of MGS1 and MGS2 are compared with Implicit GS in Table 3.1[3]. The table gives the 1st and 10th solution component of the solution sub-vectors x_1 and x_2 in (3.27)-(3.28),respectively, at $t = 0.015s$. It is noted that, MGS2 is more accurate than MGS1 when compared with Implicit GS. Although MGS2 is slower than MGS1 it is still considerably faster than Implicit GS. The implementation of the improved multigrid algorithm using the Implicit GS for treating the amplitude equations is presently under consideration.

The above discussion shows that the multigrid method seems to be advantageous for the solution of large systems of ODEs because a large saving in computer time may be achieved. One such area where this method may be viable is the solution of large stiff systems of ODEs resulting from PDEs. The solution of such systems is considered in the next chapter.

[3]Appendix 1 contains complete results for this example.

SOLVER	ERROR TOLERANCE OR STEP-SIZE	COMPUTING TIME (s)	x_1^{10} ($\times 10^3$)	x_2^{10} ($\times 10^{-1}$)
IMPLICIT GS	$\varepsilon = 1 \times 10^{-4}$	326	3373	3887
MGS1	$H = 1 \times 10^{-4}$	17	3392	3890
MGS2	$H = 1 \times 10^{-4}$	35	3373	3882

Table 3.1: Numerical Results for Example 3.5.

Chapter 4

Solution of Stiff ODE Systems Resulting from PDEs

4.1 Introduction

In order to solve scientific and engineering problems on a personal computer it is often convenient to use a readily available software package. In the field of ordinary differential equations very reliable and robust software is available for solving the initial value problem

$$\frac{dy}{dt} = f(t, y); \quad y(t_0) = y_0; \tag{4.1}$$

where y_0 is a vector of initial values.

One such software package is the GEAR[1] solver (GS). This solver is capable of solving both stiff and non-stiff equations of the type (4.1) and has reliable and efficient algorithms for dynamically changing step-size and method order to maintain stability and user specified accuracy. Other popular software packages for solving (4.1) are too numerous to mention and can be found in the book *Stiff Computation* [1].

The current state of the art for ODEs is in direct contrast to that for the field of partial differential equations[2] (PDEs) where, with the possible exception of elliptic PDEs, there exists very limited packaged software that is capable of efficiently solving very wide classes of nontrivial and difficult problems. One reason for this is that due to the variety of PDE equations, it is much more difficult to produce general purpose software packages for PDEs. In view of this, it is not surprising that the common trend in most of the current software packages for solving initial boundary value PDEs is to

[1]This is the 1974 version of GEAR developed by A.C.Hindmarsh of Lawrence Livermore Laboratory. The solver has been modified, so that the integration proceeds to the next desired output time, by M.Borysiewicz of the Institute of Atomic Energy, Swierk, Poland.

[2]Here we consider only 2nd order PDEs

use a numerical technique that produces a system of ODEs of the form (4.1), which may be subsequently solved by a state of the art ODE integrator.

This technique is called the *method of lines* or *semi-discretization method* and will be discussed in the next section.

4.2 Method of Lines (MOL)

Basically the MOL can be described as follows :

If one has a time dependent PDE, then it can be transformed into a corresponding system of ODEs by discretizing the space variable(s).

To demonstrate the method, we consider the one-dimensional heat equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}; \quad 0 \leq x \leq X; \quad t > 0; \tag{4.2}$$

where $u(x,t)$ satisfies the initial condition

$$u(x,0) = g(x); \quad 0 \leq x \leq X; \tag{4.3}$$

and has known boundary values[3] at $x = 0$ and X for $t > 0$.

[3]The boundary conditions may be functions of time, but here we consider only time independent boundary conditions.

If the spatial derivative in equation (4.2) is replaced at (x,t) by the central difference approximation

$$\frac{1}{h^2}\{u(x-h,t)-2u(x,t)+u(x+h,t)\}; \tag{4.4}$$

and x is considered as a constant, equation (4.2) can be written as the ODE

$$\frac{du(t)}{dt}=\frac{1}{h^2}\{u(x-h,t)-2u(x,t)+u(x+h,t)\}+O(h^2). \tag{4.5}$$

The MOL consists of subdividing the interval $0 \leq x \leq X$ into $N+1$ equal sub-intervals by the grid lines $x_i = ih,\ i = 0,1,...,N+1$, where $(N+1)h = X$, (see Figure 4.1), and applying (4.5) at each mesh point $x_i = ih,\ i = 1,2,...,N$, along time level t. It then follows that the values of $v_i(t)$ approximating $u_i(t)$ will be the exact solution values of the system of N ODEs

$$\begin{aligned}
\frac{dv_1(t)}{dt} &= \frac{1}{h^2}(v_0-2v_1+v_2);\\
\frac{dv_2(t)}{dt} &= \frac{1}{h^2}(v_1-2v_2+v_3);\\
&\vdots\\
\frac{dv_N(t)}{dt} &= \frac{1}{h^2}(v_{N-1}-2v_N+v_{N+1});
\end{aligned}$$

where v_0 and v_{N+1} are known boundary values.

The above system of ODEs can be written in a matrix form as

$$\frac{d}{dt}\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N-1} \\ v_N \end{bmatrix} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N-1} \\ v_N \end{bmatrix} + \frac{1}{h^2}\begin{bmatrix} v_0 \\ 0 \\ \vdots \\ 0 \\ v_{N+1} \end{bmatrix};$$

i.e. as

$$\frac{dv(t)}{dt} = Av(t) + b \tag{4.6}$$

where $v(t) = [v_1, v_2, ...v_N]^T$, b is a column vector of zeros and known boundary values, and A is a matrix of dimension N given by

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \tag{4.7}$$

The above description shows that the MOL consists of two stages, viz.

(i) semi-discretization, and

(ii) the solution of the resulting ODEs.

Several observations about this method are given below.

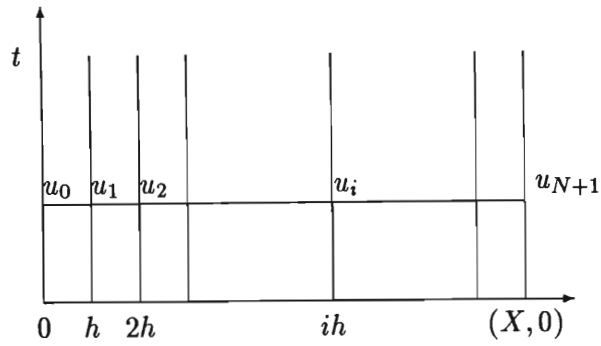


Figure 4.1: Method of lines: semi-discretization process

1. The method is most directly applicable to a problem of the *form*

$$u_t = F(x, u, u_x, u_{xx});$$

$$u(x, 0) = given;$$

and $u(x,t)$ specified at the boundary points. Here x stands for the space variables, for example, (x, y) in a problem with two space variables, $u_t = \dfrac{\partial u}{\partial t}$ and t represents the time variable.

2. One must *balance the errors*: Truncatioin error is determined by two independent sources:

(i) the discretization of the space variable $O(h^2)$ in equation (4.2)

(ii) the solution of the resulting system of ODEs.

Thus the error associated to solving the ODEs should be approximately equal to that in discretizing the space variables.

3. The problem can be *nonlinear*: Since the ODE solution techniques can be applied equally well to nonlinear problems, it is not necessary for the right hand side of (4.6) to be linear.

The method applies to an equation like

$$\frac{\partial u}{\partial t} = (u^2 + t)\frac{\partial^2 u}{\partial x^2} + \left(\frac{\partial u}{\partial x}\right)^2 - \sin(u + tx).$$

4.The right hand side is *banded*: In this linear problem the matrix A in equation (4.6) is tri-diagonal. Even if the system is nonlinear, there is a special banded structure to the problem that can be exploited. In this case it may be advantageous to use some sparse matrix techniques in order to resolve storage difficulties. Some ODE software, for example LSODES [10], has special provisions for "banded" right hand sides just to facilitate its use with the MOL . However, one should be careful when using such codes, since the costs involved in programming to exploit sparsity may be unacceptable.

5. Any *discretization method* can be used for the space variables: The application of the MOL to the PDE problem (4.2) is characterized by the form of the algebraic approximation of the spatial derivative. Generally, these approximations can be classified as direct and indirect. The direct methods involve direct replacement of the PDE spatial derivatives with algebraic ap-

proximations, e.g. finite differences, while the indirect methods involve some operations performed on the PDE residuals, e.g. the various methods based on weighted residuals, such as Galerkin's, and collocation on finite elements. The first approach leads to systems of explicit ODEs of the form :

$$\frac{dy}{dt} = f(y,t); \tag{4.8}$$

while the second approach leads to systems of implicit ODEs of the form

$$A\frac{dy}{dt} = f(y,t), \tag{4.9}$$

where A is a coupling matrix that may be a function of y. In this work we used the method of finite differences because it is the simplest, and moreover, because the ODE integrator, viz. GS, which we are using, is directly applicable to the ODE system (4.8).

6. *Dimensionality*: If the problem under consideration is one-dimensional then, in many cases, 50 lines can be used without excessive computing time, provided an efficient ODE solver is available. On the other hand, as the dimensionality of the problem increases, the number of ODEs increases exponentially. For example, if M PDEs are discretized in a D-dimensional space with N grid points in each direction, then the number of ODEs is approximately MN^D, which can be quite large (consider $M = 2$, $D = 3$,

$N = 30$, then the number of ODEs approximately equals 54 000!). This latter case is so large that it is unlikely that the ODEs can be solved without taking advantage of sparsity, and using a powerful computer.

7. The system of ODEs is frequently *stiff*: For the tri-diagonal matrix A, defined by (4.7), the eigenvalues λ_s can be calculated by the formula [23]

$$\lambda_s = -\left(\frac{4}{h^2}\right)\left(\sin^2\frac{s\pi}{2(N+1)}\right); \quad s = 1, 2, ..., N. \tag{4.10}$$

Hence for large N, the largest eigenvalue can be approximated by

$$\begin{aligned} \lambda_{\max} &= -\left(\frac{4}{h^2}\right)\sin^2\left(\frac{N}{2(N+1)}\pi\right) \\ &\to -\left(\frac{4}{h^2}\right) \quad \text{as } N \to \infty; \end{aligned}$$

while the smallest eigenvalue can be estimated by

$$\begin{aligned} \lambda_{\min} &= -\left(\frac{4}{h^2}\right)\sin^2\left(\frac{\pi}{2(N+1)}\right) \\ &\approx -\left(\frac{4}{h^2}\right)\left(\frac{\pi^2}{4N^2}\right). \end{aligned}$$

Then the stiffness ratio S (see section 2.2) for the system of ODEs (4.6) is measured by

$$\begin{aligned} S &= \frac{\max \mu_s}{\min \mu_s} \\ &\approx \frac{4N^2}{\pi^2}. \end{aligned}$$

Thus the value of N is crucial in determining whether the PDE given by equation (4.2) can be efficiently solved using existing ODE techniques, since for large N the system of ODEs (4.6) resulting from (4.2) is *stiff*. In this case the *stiff methods* for solving stiff ODEs, discussed in section 2.3, have to be used.

8. *Stiff and Non-stiff Integration*: Since the system of ODEs resulting from the original PDE system must be numerically integrated with respect to t, it follows that the choice of an initial-value integrator is necessary. Most ODE integrators implement time integration methods which are suitable for solving either stiff or non-stiff equations .

Typically stiff integrators implement methods that have good stability properties. For example, the GS implements Gears's stiffly stable formulas of up to fifth order. These methods are generalizations of the usual backward difference formulas.

The methods implemented in stiff integrators usually employ use a modified Newton's method for solving their nonlinear equations. This iterative scheme requires the Jacobian matrix and therefore involves additional computational costs.

On the other hand, the most popular non-stiff integrators implement

Adams' methods. For example, the GS, which has both stiff and non-stiff methods built into the same program, uses Adams-Bashforth-Moulton predictor-corrector schemes of up to order twelve, for the non-stiff integration.

The advantage of non-stiff integrators is that their solution process does not require the solution of large matrix systems, since fixed point iteration is used to solve the nonlinear equations. Their main disadvantage is that the stability regions for the higher order methods, are small and hence often require very small time steps to maintain stability and accuracy. Non-stiff integrators are not always adequate for even simple PDEs, since the problem can become stiff as the number of mesh points increases.

However, non-stiff integration should not be precluded on the basis of the stability requirement. In fact, it is wise to try explicit integration of the ODE system first. It is easy to program and is very effective in many MOL applications.

Only if the computing times are excessive, i.e. the integration time steps are small, or the eigenvalues are widely separated, should implicit integration be used. In other words, using a stiff integrator for a non-stiff problem can be wasteful in computer time because of the linear algebra of implicit

integration, which may be performed unnecessarily.

9. *Adaptive Grids*: The adaptive grid technique is a numerical technique which is used to compute accurate numerical solutions to PDE systems which exhibit steep moving fronts. For example, such fronts appear in solutions of systems of some first-order PDEs. Numerical methods which approximate the spatial derivatives of such PDE systems using a uniformly spatial grid are bound to fail. A possible solution to this problem has been to implement moving grids which automatically adapt to the local needs of the problem, e.g. concentrate spatial grid points where the solution is changing rapidly in space (see section 4.3).

10. *Software*: Over the years much attention has been focused on developing software packages for PDEs, that implement the MOL . Some of the more popular software packages include [1] PDEONE, PDETWO, PDECOL, DPDE, DSS/2, DYLA and FORSIM. These codes provide the user with the option of performing stages (i) and (ii) mentioned (on page 56) in the MOL separately or automatically. However, it may not be possible to extract the integrators from codes that perform both stages (i) and (ii) together.

4.3 Numerical Example

To illustrate the main points made above we take a PDE with travelling wave solutions. Burger's equation [16] for $u = u(x,t)$ is

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - u\frac{\partial u}{\partial x}. \tag{4.11}$$

An exact solution for Burger's equation is given by

$$u(x,t) = f(x,t) = \frac{f_1(x,t)}{f_2(x,t)}; \tag{4.12}$$

where

$$\begin{aligned} f_1 &= 0.1e^{-A} + 0.5e^{-B} + e^{-C}; \\ f_2 &= e^{-A} + e^{-B} + e^{-C}; \\ A(x,t) &= \left(\frac{0.05}{\nu}\right)(x - 0.5 + 4.95t); \\ B(x,t) &= \left(\frac{0.25}{\nu}\right)(x - 0.5 + 0.75t); \\ C(x,t) &= \left(\frac{0.5}{\nu}\right)(x - 0.375). \end{aligned}$$

The initial conditions and Dirichlet boundary conditions are taken directly from (4.12), i.e.

$$u(x,0) = f(x,0); \tag{4.13}$$

$$u(0,t) = f(0,t); \tag{4.14}$$

$$u(1,t) = f(1,t). \tag{4.15}$$

Burger's equation is an ideal test problem since :

- It is nonlinear.
- The exact solution of the PDE is known.
- It can be thought of as a hyperbolic problem with negligible diffusion for small ν [3].
- It is sometimes used in boundary layer calculations for the flow of viscous fluids [3].

As mentioned above the simplest method of spatial discretization is to discretize along the x-axis using a uniform mesh of $N+2$ grid points and to replace all spatial derivatives in (4.11) by (say) centered finite difference approximations. Thus if we take

$$h = \frac{1}{N+1}; \tag{4.16}$$

$$u_i(t) = f(ih,t); \quad i = 1,2,...,N+1; \tag{4.17}$$

then the system of ODEs for the MOL solution to (4.11) is

$$\dot{u}_i = \frac{\nu}{h^2}(u_{i+1} - 2u_i + u_{i-1}) - \frac{u_i}{2h}(u_{i+1} - u_{i-1}); \; i = 1,2,...,N; \tag{4.18}$$

$$u_i(0) = f(ih, 0); \tag{4.19}$$

$$u_1(t) = f(0, t); \tag{4.20}$$

$$u_N(t) = f(1, t); \tag{4.21}$$

where (4.19)-(4.21) are taken directly from (4.12).

We note by inspection that the system of ODEs (4.18) has a tri-diagonal Jacobian matrix, the sub-diagonal elements of which are

$$\frac{\partial \dot{u}_i}{\partial u_{i-1}} = \frac{\nu}{h^2} + \frac{u_i}{2h}; \tag{4.22}$$

while the diagonal elements are

$$\frac{\partial \dot{u}_i}{\partial u_i} = -\frac{2\nu}{h^2} - \left(\frac{u_{i+1} - u_{i-1}}{2h}\right); \tag{4.23}$$

and the super-diagonal elements are

$$\frac{\partial \dot{u}_i}{\partial u_{i+1}} = \frac{\nu}{h^2} - \frac{u_i}{2h}; \tag{4.24}$$

for $i = 1, 2, ..., N$.

Another method of spatial discretization for (4.11) was described by Chin, Hedstrom, Karlson [3]. They used a simplified Galerkin method to reduce Burger's equation to a linearly implicit system of ODEs of the form (4.9). The system for (4.11) is then of the form

$$A\dot{u} = g(t, u); \tag{4.25}$$

where A is an $N \times N$ tri-diagonal matrix.

If $u_i(t)$ is the numerical solution of (4.11) at x_i then

$$g_i = -\frac{1}{4h}\left[u_{i+1}^2 - u_{i-1}^2\right] + \frac{\nu}{h^2}\left[u_{i+1} - 2u_i + u_{i-1}\right]; \quad i = 1, 2, ..., N; \qquad (4.26)$$

The initial and boundary conditions can be taken from (4.19)-(4.21). The Jacobian matrix of g can be obtained from (4.26) as before, and is given by

$$\frac{\partial g_i}{\partial u_{i-1}} = \frac{u_{i-1}}{2h} + \frac{\nu}{h^2}; \qquad (4.27)$$

$$\frac{\partial g_i}{\partial u_i} = -\frac{2\nu}{h^2}; \qquad (4.28)$$

$$\frac{\partial g_i}{\partial u_{i+1}} = -\frac{u_{i+1}}{2h} + \frac{\nu}{h^2}; \qquad (4.29)$$

for $i = 1, 2, ..., N$.

We solved this problem on a PC/AT (speed-10MHz) using double precision, and the following solvers :

(i) Explicit GS : Non-stiff solver which implements the implicit Adams' method and uses fixed point iteration to solve the associated nonlinear equations.

(ii) Implicit GS : Stiff solver which implements the Gear stiffly stable methods and uses a generalized Newton's method to solve the corresponding nonlinear equations. The Jacobian matrix is computed automatically using finite differences.

(iii) Implicit GS1 : Same as (ii) except that in this case the Jacobian matrix is replaced by a diagonal matrix based on a directional derivative.

(iv) Implicit GS2 : Same as (ii) except that the solver requires the user to supply a separate subroutine for the Jacobian matrix.

(v) RK4 : Explicit fourth order Runge-Kutta solver with fixed step-size h.

The first four solvers above are built into the GS[4]. We only consider the first method of spatial discretization , viz. finite differences, since the GS is applicable to the ODE system (4.18)-(4.21).

For this problem we used a uniform mesh of 50 grid points and the value of ν was taken equal to 0.003. The initial time step and the time integration error tolerance in GS were set equal to 1×10^{-8} and 1×10^{-5}, respectively. The step-size in RK4 was taken equal to $h = 1 \times 10^{-5}$.

Table 4.1[5] [6] gives the $20th$ and $30th$ solution components at $t = 0.1s$, and Table 4.2 gives the error in these solution components. From Table 4.2

[4]In all, there are eight options available in GS. In addition to the above 4 options the other combinations are implicit Adams' method with Newton iteration, and Gear's stiffly stable methods with fixed point iteration.

[5]Results for this problem using MGS will be presented in the next chapter.

[6]Appendix 1 contains a complete set of results for this problem.

it is seen that the Implicit GS1 performs the best for this problem. It is the fastest and most accurate of all the solvers. It may also be noted that the Explicit GS is approximately 4 times faster than Implicit GS2 and 7 times faster than Implicit GS, and the solutions obtained are almost identical. This confirms the point made previously that explicit integration should not be precluded on the basis of the stability requirement.

Another important point worth noting is that it is wasteful, from the viewpoint of computing time, to make the time integration error tolerance for this problem smaller than 1×10^{-5}. In fact, most of the error in the solution is due to the spatial discretization. This can be verified by noting that refining the mesh from 50 to 100 points reduces the error in the solution, at $x = 0.4$ and $x = 0.6$, by approximately a factor of 1.25 (see Table 4.3). On the other hand, reducing the time integration error tolerance to 1×10^{-10} affects the solution very slightly but the computing time is almost doubled (see Table 4.3). Finally, we note that RK4 is extremely slow for this problem, and in general its use is not recommended for the solution of ODEs resulting from PDEs.

Now let us consider Burger's equation (4.11) with $\nu = 1$. The reason for considering this case is to show that the ODE system resulting from (4.11)

SOLVER	COMPUTING TIME (s)	$u^{20}(\times 10^{-6})$	$u^{30}(\times 10^{-6})$
EXACT SOLUTION	—	502633	106379
EXPLICIT GS	11	499106	103362
IMPLICIT GS	75	499106	103362
IMPLICIT GS1	10	499106	103362
IMPLICIT GS2	46	499106	103361
RK4	1787	499106	103362

Table 4.1: 20th and 30th solution components at $t = 0.1s$.

SOLVER	COMPUTING TIME (S)	ERROR IN $u^{20}(\times 10^{-7})$	ERROR IN $u^{30}(\times 10^{-7})$
EXPLICIT GS	11	35259	30167
IMPLICIT GS	75	35261	30162
IMPLICIT GS1	10	35259	30162
IMPLICIT GS2	46	35261	30164
RK4	1787	35279	30172

Table 4.2: Error in 20th and 30th solution components at $t = 0.1s$.

NO. OF SPATIAL GRID POINTS	COMPUTING TIME	ERROR AT $x = 0.4$ $(\times 10^{-8})$	ERROR AT $x = 0.6$ $(\times 10^{-8})$
50	11	352591	301677
100	20	280584	242697

Table 4.3: Error in u at $x = 0.4$ and $x = 0.6$ using 50 grid points.

TIME INTEGRATION ERROR TOLERANCE	COMPUTING TIME	ERROR AT $x = 0.4$ $(\times 10^{-8})$	ERROR AT $x = 0.6$ $(\times 10^{-8})$
1×10^{-5}	11	352591	301677
1×10^{-10}	19	352620	301633

Table 4.4: Error in u at $x = 0.4$ and $x = 0.6$ using 100 grid points.

can become stiff as the number of spatial grid points becomes large. Using the same parameters in the Explicit GS and Implicit GS1, as before, we solved this problem using uniform meshes of 50 and 100 grid points. Table 4.5 and Table 4.5 give the solution vector at $x = 0.4$ and $x = 0.6$, for $t = 0.1$, using 50 and 100 grid points, respectively. In both instances we see that the Explicit GS is slower than the Implicit GS1 and in particular for 100 grid points the Explicit GS is considerably slow. This clearly indicates that the problem is stiff since fixed point iteration is expensive.

Finally, we note that since Burger's equation exhibits steep moving fronts the solution process might be enhanced by using the adaptive grid technique. Schiesser [22] has applied the adaptive grid technique to Burger's equation. The solution obtained is shown in Figure 4.2. The solid curves represent the exact solution and the points are the computed MOL solution. The behaviour of the adaptive grid for different values of t is shown at the base of the figure and the changing number ODEs (due to the changing number of spatial grid points during the solution process) is shown in Figure 4.3.

SOLVER	COMPUTING TIME s	SOLUTION AT $x = 0.4$ $(\times 10^{-6})$	SOLUTION AT $x = 0.6$ $(\times 10^{-6})$
EXACT SOLUTION	—	534987	521288
EXPLICIT GS	211	532375	518681
IMPLICIT GS1	44	532299	518604

Table 4.5: Solution at $t = 0.1s$ using 50 grid points.

SOLVER	COMPUTING TIME s	SOLUTION AT $x = 0.4$ $(\times 10^{-6})$	SOLUTION AT $x = 0.6$ $(\times 10^{-6})$
EXACT SOLUTION	—	534571	521012
EXPLICIT GS	1644	531944	518410
IMPLICIT GS1	282	531905	518364

Table 4.6: Solution at $t = 0.1s$ using 100 grid points.

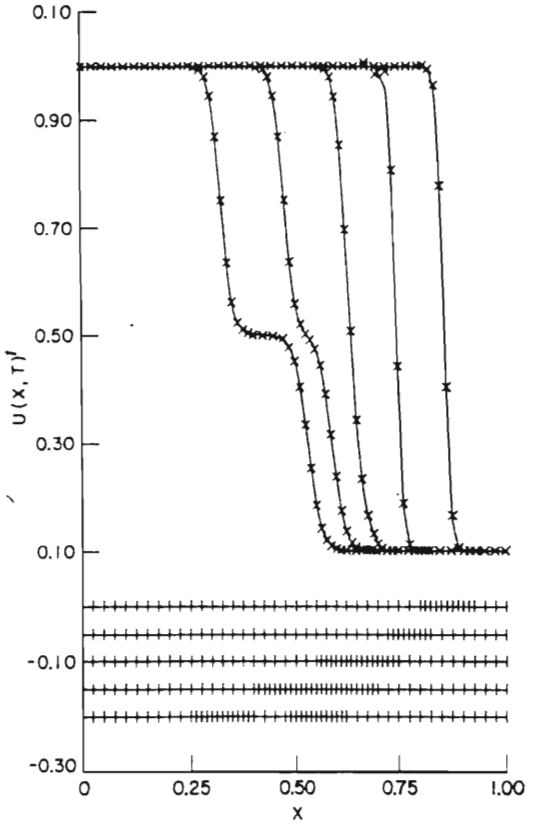


Figure 4.2: Adaptive grid solution of Burger's equation

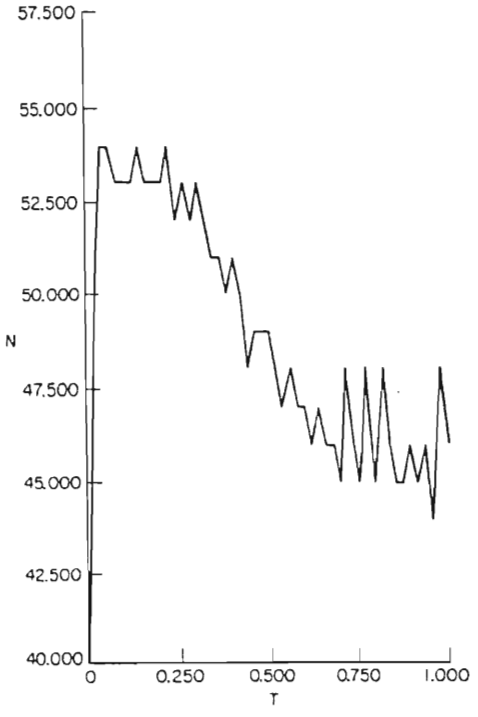


Figure 4.3: Number of ODEs against t for the Burger's equation solution

Chapter 5

Multigrid Solver for PDE Systems

5.1 Introduction

The primary objective in this chapter is to present a numerical technique which will serve as an interface allowing the newly developed *Multigrid Solver*, discussed in chapter 2, to be applied to time dependent PDEs. For this purpose we use a discretized form of a PDE obtained by the method of lines, using finite difference approximations.

We intend to show that the approach is quite robust in that it is capable of

solving a wide class of difficult PDE problems with reduced computational effort. In particular, we consider the general form of two important PDE equations occuring in practice, viz. the nonlinear diffusion equation and the telegraph equation. Furthermore, we briefly examine a practical area, viz. atmospherical physics, where we feel this approach might be of significance. We feel, however, that the approach is applicable to an even wider class of physically realistic problems.

Finally, we stress that the approach seems to be particularly useful for solving large systems of PDEs, occuring in scientific and engineering applications, on a *personal computer*. Therefore to make the technique accessible to a wider range of users we implemented the technique in a computer program which we call PDEMGS.

5.2 The Nonlinear Diffusion Equation

Consider the general system of N nonlinear PDEs, of at most second order, on the interval $[a, b]$ in the following form

$$\frac{\partial u}{\partial t} = f(t, \rho, u, u_\rho, u_{\rho\rho}); \quad t \geq 0; \tag{5.1}$$

where

$$\begin{aligned} u &= (u_1, u_2, ..., u_N); \\ u_\rho &= \left(\frac{\partial u_1}{\partial \rho}, \frac{\partial u_2}{\partial \rho}, ..., \frac{\partial u_N}{\partial \rho}\right); \\ u_{\rho\rho} &= \left(\frac{\partial^2 u_1}{\partial \rho^2}, \frac{\partial^2 u_2}{\partial \rho^2}, ..., \frac{\partial^2 u_N}{\partial \rho^2}\right). \end{aligned}$$

Each function u_k depends on the spatial variable ρ and time t. Depending on the arguments of the vector function f, any particular equation in (5.1) may be an ODE, a first order PDE, or a second order PDE. In the latter two cases *boundary conditions* need to be specified at a and b. These boundary conditions are assumed to be of the form

$$\alpha_k u_k + \beta_k \frac{\partial u_k}{\partial \rho} = \gamma_k. \tag{5.2}$$

Such boundary conditions can be classified into three types : Dirichlet ($\beta_k = 0$); Neumann ($\alpha_k = 0$); and mixed ($\alpha_k \neq 0, \beta_k \neq 0$). We make the basic assumption that the *boundary conditions are consistent with the initial condition*. If the initial condition is given in the form

$$u_k(0, \rho) = q_k(\rho); \tag{5.3}$$

then q_k must satisfy the boundary condition (5.2).

It is important to note that for the equations (5.1)-(5.3) the solution may either not exist or be not unique. In the sequel we deal only with mathematically meaningful initial-boundary value problems for PDEs such that the solution exists and is unique.

5.2.1 Using MGS to Solve the Nonlinear Diffusion Equation

We show that the system of PDEs (5.1)-(5.3) can be reduced to the form

$$\frac{dx_k}{dt} = f_k(x_1, x_2, ..., x_N; t); \quad x_k(0) = q_k; \quad k = 1, 2, ..., N. \tag{5.4}$$

required by MGS. The basic technique which we use to achieve this is the numerical method of lines discussed in chapter 3.

Firstly, we assume that $\rho \in [0, 1]$ for simplicity, and divide the interval $[0, 1]$ into $m - 1$ equal sub-intervals of size h. This imposes on the interval $[0, 1]$ a mesh of m equally spaced points. We associate with each mesh point the functions ${x_k}^i(t)$, $(k = 1, 2, ...N)$ that represent approximations of the true solution taken at the points $\rho^i = \dfrac{i-1}{m-1}$. We define

$$u_k^i(t) = u_k\left(\frac{i-1}{m-1}, t\right). \tag{5.5}$$

To obtain the system of ODEs in the form (5.4) in terms of x_k^i we consider the kth PDE in the system (5.1) at $\rho = \rho^i$ where $1 < i < m$. For this we need the values $u, u_\rho, u_{\rho\rho}$ which are not known exactly. We approximate these values by means of 3-point central-difference approximations involving the functions x_k^i. More specifically, for $\rho = \rho^i$ we substitute the approximations

$$u_j(t, \rho^i) \approx x_j{}^i; \tag{5.6}$$

$$\frac{\partial u_j}{\partial \rho}(t, \rho^i) \approx \frac{x_j^{i+1} - x_j^{i-1}}{2h}; \tag{5.7}$$

$$\frac{\partial^2 u_j}{\partial \rho^2}(t, \rho^i) \approx \frac{x_j^{i+1} - 2x_j^i + x_j^{i-1}}{h^2}; \tag{5.8}$$

$i = 2, 3, ..., m-1$; $j = 1, 2, ...N$ in the kth PDE in (5.1).

Next we consider the boundary points, $\rho = 0$ and $\rho = 1$. It will suffice to consider the difference approximations for the left boundary point $\rho = 0$ since that for the right boundary point $\rho = 1$ is completely analogous.

If for the kth PDE at $\rho = 0$ we have that $\beta_k = 0$ in (5.2) then clearly the value of x_k^1 is given by $x_k^1 = \dfrac{\gamma_k}{\alpha_k}$ and no ODE is required. However to preserve the basic structure of the ODE system (5.4) we set

$$\frac{dx_k^1}{dt} = 0. \tag{5.9}$$

On the other hand, suppose that for the kth PDE at $\rho = 0$, we have that $\beta_k \neq 0$ in (5.2). Then the value of the solution at $\rho = 0$ is not given explicitly.

In this case we combine (5.1) and (5.2) at $\rho = 0$, to derive an appropriate ODE for the left boundary point.

Proceeding as for the interior points, we use at $\rho = 0$ the approximations

$$u_j(t,0) \approx x_j^1; \tag{5.10}$$

$$\frac{\partial u_j}{\partial \rho}(t,0) \approx \begin{cases} \frac{x_j^2 - x_j^1}{2h} & \text{if } \beta_j = 0, \quad j \neq k \\ \frac{\gamma_j - \alpha_j x_j^1}{\beta_j} & \text{if } \beta_j \neq 0 \end{cases} \tag{5.11}$$

$$\frac{\partial^2 u_j}{\partial \rho^2}(t,0) \approx \begin{cases} \frac{2\left(x_j^2 - x_j^1\right)}{h^2} & \text{if } \beta_j = 0, \quad j \neq k \\ \frac{2}{h}\left[\frac{x_j^2 - x_j^1}{h} - \frac{\left(\gamma_j - \alpha_j x_j^1\right)}{\beta_j}\right] & \text{if } \beta_j \neq 0 \end{cases} \tag{5.12}$$

for $j = 1, 2, ...N$.

Clearly upon substitution of (5.6)-(5.8) and the boundary conditions into (5.1) we obtain a semi-discrete system of $N * m$ ODEs of the form

$$\begin{aligned} \frac{dx_k^1}{dt} &= \tilde{f}_k^1(t; x^1, x^2) \\ \frac{dx_k^i}{dt} &= \tilde{f}_k^i(t; x^{i-1}, x^i, x^{i+1}) \quad i = 2, ..., m-1 \\ \frac{dx_k^m}{dt} &= \tilde{f}_k^m(t; x^{m-1}, x^m) \end{aligned}$$

for $k = 1, 2, ...N$; where $x^i = (x_1^i, x_2^i, ..., x_N^i)$. The above system can be expressed in vector form as

$$\frac{dx_k}{dt} = \tilde{f}_k(t; x_1, x_2, ..., x_N); \tag{5.13}$$

$k = 1, 2, ..., N$. The appropriate initial conditions for (5.13) are easily ob-

tained from (5.3), i.e.

$$x_k^i(0) = q_k^i;$$

or in vector form

$$x_k(0) = q_k; \quad k = 1, 2, ..., N. \tag{5.14}$$

Combining equations (5.13) and (5.14) gives the desired form (5.4) required by MGS.

The next step is to implement the above technique in a computer program which we call PDEMGS. Basically, PDEMGS requires the user to provide the number of spatial grid points, the vector function f and the boundary conditions for an allowable problem (5.1)-(5.3). PDEMGS then uses the difference approximations (5.6)-(5.12) to define and evaluate the right-hand side of the approximating ODE system (5.4). Moreover, this ODE system is in a form which is immediately useable with MGS.

In the next section we discuss the structure and use of PDEMGS.

5.3 Structure and Use of PDEMGS

According to the discussion presented in section 3.4, to solve the system (5.1)-(5.3) by MGS the user is required to provide a subroutine DIFEQ which

contains the approximating system of ODEs to the original system of PDEs (see Figure 3.2). This involves reducing the original system of PDEs (5.1)-(5.3) to the reduced form (5.4) discussed in the previous section.

The purpose of PDEMGS is to relieve the user of this time consuming and tedious task by automatically forming and evaluating the right-hand side of (5.4). Therefore according to the discussion presented in section 3.4, to use PDEMGS and MGS to solve the PDE system (5.1), we must construct a main program which sets parameters and allocates storage for MGS, sets the initial conditions (5.3), specifies the m points of the spatial mesh ($m \geq 3$), initializes and calls MGS, and provides for evaluation of the calculation times and printing of the results. Furthermore, a minor adjustment needs to be made in the main program, in order to incorperate PDEMGS into MGS, viz. the user needs to initialize the array $X(m)$, containing the m spatial points of the spatial mesh, and the step-size h. These variables should be specified in a common block, say, MESH.

The remaining task is to define the approximating ODE system (5.4) for MGS. This is done by noting that the PDE system (5.1)-(5.3) is completely defined if one specifies the interval $[a, b]$; the initial time, t_0; the vector functions $f = (f_k)$, $\alpha = (\alpha_k)$, $\beta = (\beta_k)$, $\gamma = (\gamma_k)$; and the initial conditions,

$q_k(x)$ $(k = 1, 2, ..., N)$.

In order to use PDEMGS to solve the PDE system (5.1)-(5.3) it is only necessary for the user to supply two basic subroutines containing the vector f and the boundary conditions (i.e. the vectors α, β, γ). These user-defined subroutines will be called F and BNDRY, respectively.

Next we consider the structure of PDEMGS and give a brief description of the subroutines which it uses. In the following description T and X are scalar quantities representing the current time and spatial variable, respectively; U is a vector containing the solution vector and UX and UXX are vector quantities containing the approximations to the 1^{st} and 2^{nd} derivatives, respectively. A brief description of each of the subroutines used in PDEMGS follows.

PDEMGS : Interface routine, between MGS and the user-defined routines, F and BNDRY, that is responsible for converting the original system of PDEs (5.1)-(5.3) into a form (5.4) which may be integrated using MGS. It replaces the routine DIFEQ shown in Figure 5.1. The new structure is illustrated in Figure 5.2. Clearly MGS will have to be trivially modified so that it properly calls PDEMGS instead of DIFEQ. This modification is made in the routines AMPL and SHAPE (see Figure 3.2). In other words, instead of

calling DIFEQ these routines call PDEMGS.

On a call to PDEMGS, it performs the semi-discretization process for converting the original system of PDEs into a system of ODEs of the form $DU = F(T, X, U)$, which may be integrated using MGS.

The input parameters are as follows :

$NPDE$ = number of PDEs

$MPTS$ = number of spatial grid points

T = current value of time

U = an $N * m$ array containing the computed solution at time T.

The output parameters are :

$DU = NPDE * MPTS$ array containing the right-hand side of the resulting system of ODEs $F(T, X, U)$ obtained by discretizing the given PDE system.

The user is required to insert a dimension statement of the following form DIMENSION ALPHA(**), BETA(**), GAMMA(**), U(**), UX(**), UXX(**),

where the symbol ** represents the actual dimension of the PDE system being solved.

To perform the semi-discretization process PDEMGS adopts the following procedure:

1. Updates the left boundary values by calling the user-defined routine BNDRY.

2. Forms approximations to $\frac{\partial u}{\partial x}$ at the left boundary.

3. Evaluates $\frac{\partial^2 u}{\partial x^2}$ at the left boundary.

4. Evaluates the right-hand side of the PDEs at the left boundary by calling the user-defined routine F.

5. Updates the right boundary values by calling BNDRY.

6. Performs main loop that form the ODEs at the grid points by the following steps :

6.1 Evaluates $\frac{\partial u}{\partial x}$ at the ith grid point;

6.2 Evaluates $\frac{\partial^2 u}{\partial x^2}$ at the ith grid point;

6.3 Evaluates right-hand side of PDEs at the ith grid point by calling F.

7. Forms approximations to $\frac{\partial u}{\partial x}$ at the right boundary.

8. Evaluates $\frac{\partial^2 u}{\partial x^2}$ at the right boundary.

9. Evaluates right-hand side of PDEs at the right boundary by calling F.

10. Returns to the calling program.

BNDRY : User-defined routine specifying α_k, β_k and γ_k in (5.2). If X is the left (right) boundary point $X(1)$ ($X(m)$) define ALPHA(K), BETA(K) and GAMMA(K) at the left (right) boundary point and return.

F : User-defined routine containing the right-hand side vector f of (5.1).

When solving PDE problems using MGS and the interface routine PDEMGS we will refer to the solver simply as PDEMGS. In order to demonstrate the potential usefulness of PDEMGS we present a few numerical examples. The value of L, number of time-steps in amplitude equations for one step in shape equations in PDEMGS, was taken equal to 10 in all cases. All computations were performed on a PC/AT (speed-10MHz) using double precision arithmetic.

5.3.1 Numerical Examples

A

We consider the numerical solution of the equation

$$2\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + u - (4x+2)e^{x+t} \tag{5.15}$$

on the x interval $[-1,1]$ and for $0 < t \leq 0.1$. We assume initial and boundary conditions of the form

$$u(x,0) = (x^2-1)e^x; \tag{5.16}$$

$$u(-1,t) = u(1,t) = 0. \tag{5.17}$$

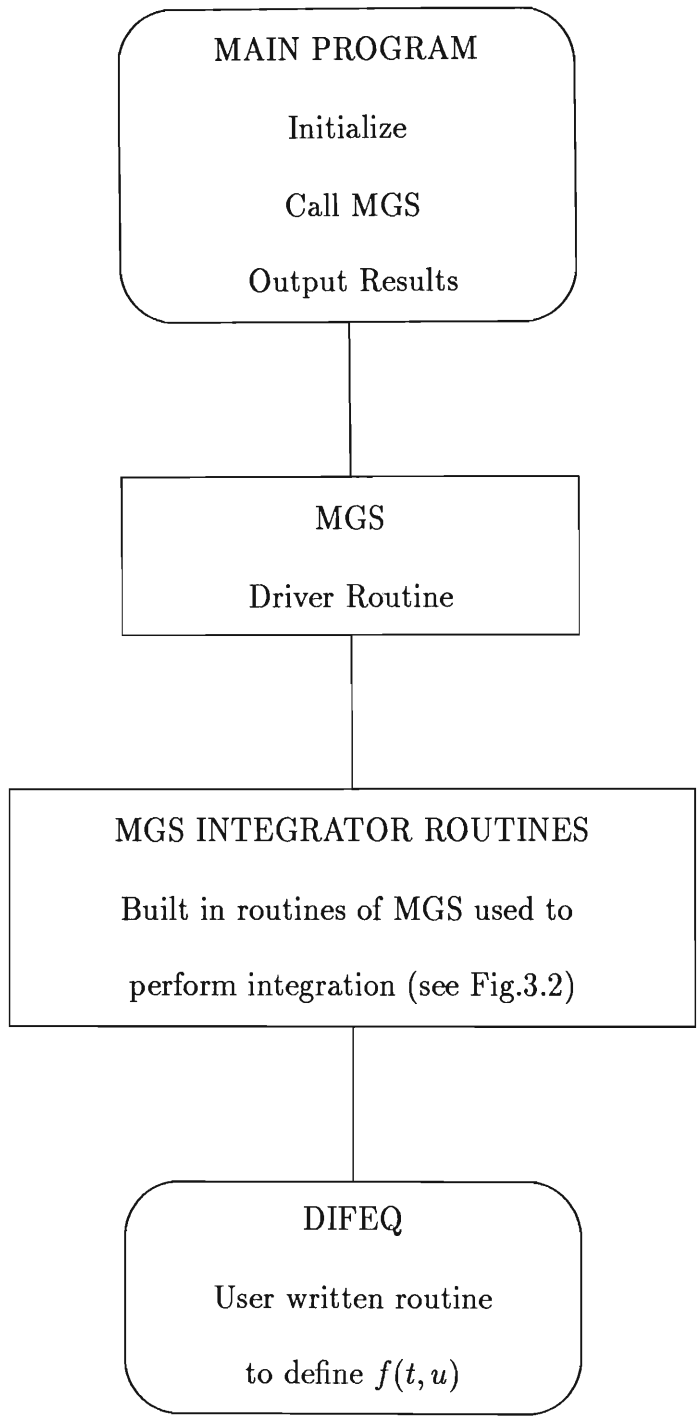


Figure 5.1: Structure of MGS. Oval shapes indicate user-defined routines.

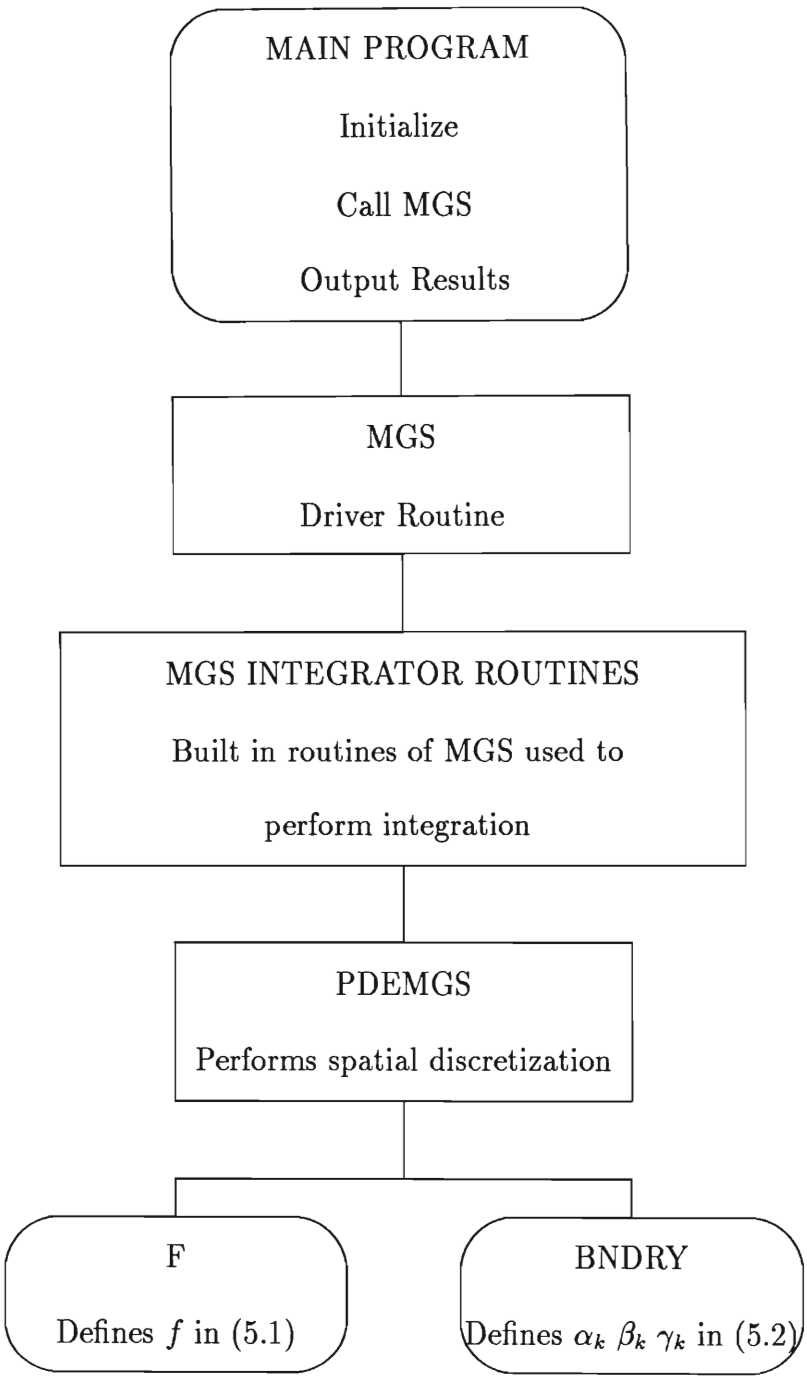


Figure 5.2: Structure of PDEMGS. Oval shapes indicate user-defined routines.

Note that the initial conditions are consistent with the boundary conditions as required by PDEMGS. The exact solution for this problem is easily seen to be

$$u(x,t) = (x^2 - 1)e^{x+t}. \tag{5.18}$$

To solve this problem we used the following solvers

(1) Explicit GS (discretization done by hand)

(2) Implicit GS (discretization done by hand)

(3) PDEMGS (discretization done automatically)

(4) RK4 (discretization done by hand).

We used uniform meshes of 60 and 120 points and the time integration error tolerance (EPS), in Explicit GS, Implicit GS and PDEMGS, was set at 1×10^{-4} and 1×10^{-5}. The initial time-step ($DTMIN$) was set equal to 1×10^{-15} in the first three solvers above. Step-sizes of 1×10^{-4} and 1×10^{-5} were used in RK4.

In the first instance, using a uniform mesh of 60 points we obtained the numerical results shown in Table 5.1[1]. In the table the 10th and 40th components of the solution vector u are shown at $t = 0.1$. From the table it is

[1]See Appendix 1

seen that with $EPS = 1 \times 10^{-5}$ the Implicit GS was slightly faster than Explicit GS and dramatically faster (approximately twenty-eight times faster) than RK4 solver with fixed step-size $h = 1 \times 10^{-5}$. This is clearly indicative of the stiffness inherent in the problem. On the other hand, PDEMGS was approximately ten times faster than the Implicit GS and was still sufficiently accurate. Such a difference in computing time might be crucial when solving PDEs where a very fine mesh refinement is required (see chapter 3).

Moreover, in the second instance an even more interesting observation was made using PDEMGS. In this case using 120 grid points means that we have to solve a system of 120 ODEs. Hence this represents an even stiffer system due to the $\dfrac{\partial^2 u}{\partial x^2}$ term. The numerical results giving the 10th and the 40th components of the solution vector u at $t = 0.1$ are shown in Table 5.2. The Implicit GS failed to solve this problem ! It is suspected that the problem lies with the Newton iteration procedure for solving the nonlinear equations arising from implicit integration. The solver seems to have failed to solve the nonlinear system (2.28) resulting for this problem. The Explicit GS and RK4, produced a solution but were slow, in particular, RK4 with $h = 1 \times 10^{-5}$ was excessively slow. PDEMGS, on the other hand, with $EPS = 1 \times 10^{-5}$ gave a sufficiently accurate solution and only took $19s$.

SOLVER	ERROR TOLERANCE OR STEP-SIZE	COMPUTING TIME (s)	u^{10} ($\times 10^{-5}$)	u^{40} ($\times 10^{-4}$)
EXACT SOLUTION	—	—	-28523	-13669
EXPLICIT GS	$EPS = 1 \times 10^{-4}$	87	-28523	-13668
	$= 1 \times 10^{-5}$	107	-28522	-13668
IMPLICIT GS	$EPS = 1 \times 10^{-4}$	91	-28522	-13668
	$= 1 \times 10^{-5}$	102	-28521	-13668
PDEMGS	$EPS = 1 \times 10^{-4}$	9	-28521	-13667
	$= 1 \times 10^{-5}$	10	-28519	-13666
RK4	$H = 1 \times 10^{-4}$	283	-28521	-13668
	$= 1 \times 10^{-5}$	2838	-28522	-13668

Table 5.1: Numerical Results for Example 5.3.1(A), using 60 points.

SOLVER	ERROR TOLERANCE OR STEP-SIZE	COMPUTING TIME (s)	u^{10} ($\times 10^{-5}$)	u^{40} ($\times 10^{-5}$)
EXACT SOLUTION	—	—	-13226	-69011
EXPLICIT GS	$EPS = 1 \times 10^{-4}$	773	-13225	-69011
	$= 1 \times 10^{-5}$	747	-13225	-69011
IMPLICIT GS	$EPS = 1 \times 10^{-4}$	****	FAILED	****
	$= 1 \times 10^{-5}$	****	FAILED	****
PDEMGS	$EPS = 1 \times 10^{-4}$	18	-13225	-69007
	$= 1 \times 10^{-5}$	19	-13225	-69007
RK4	$H = 1 \times 10^{-4}$	572	-13225	-69010
	$= 1 \times 10^{-5}$	5723	-13225	-69011

Table 5.2: Numerical Results for Example 5.3.1(A), using 120 points.

B

We attempted to solve Burger's equation

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x}; \tag{5.19}$$

presented in section 4.3, using PDEMGS. The initial and boundary conditions were taken as in equations (4.13)-(4.15). As noted in section 4.3, in order to solve this problem, to any reasonable degree of accuracy, it is necessay to use a very fine mesh refinement if a uniform mesh is being used for the spatial discretization.

We solved this problem in exactly the same way as the previous one, using a uniformly spaced mesh of 400 points. We did not attempt to obtain a solution using RK4, considering the excessive computer run time in example 4.3 when only 50 grid points were used. The value of ν, in equation (5.19), was taken equal to 0.003. The parameters EPS and $DTMIN$ were set equal to 1×10^{-7} and 1×10^{-15}, respectively, in Explicit GS, Implicit GS and PDEMGS. The numerical results giving the 50th and 300th components of the solution vector u, at $t = 0.1$ are shown in Table 5.3.

Once again Implicit GS failed to solve this problem for the same reason as suspected in the previous example. The Explicit GS produced a solution

SOLVER	COMPUTING TIME (s)	$u^{50}(\times 10^{-5})$	$u^{300}(\times 10^{-5})$
EXACT SOLUTION	—	100000	10000
EXPLICIT GS	1008	99939	10000
IMPLICIT GS	****	FAILED	****
PDEMGS	28	99932	10000

Table 5.3: Numerical Results for Example 5.3.1(B).

but was very slow. PDEMGS, on the other hand, produced a reasonably accurate solution in under half a minute. Since Burger's equation is frequently encountered in practice, this shows that PDEMGS might be useful for practical applications. In section 5.4 we consider a particular practical application of PDEMGS.

In general the ODE system resulting from a PDE system becomes stiff as the number of spatial grid points increases (see section 4.2). Hence Explicit GS may not be suitable for these problems and moreover the fixed step RK4 method is completely ruled out. In the remaining problems in this section we solve using only Implicit GS and PDEMGS.

In the next example we consider a system of two nonlinear PDEs.

C

Consider the system of PDEs [16]

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left((v-1)\frac{\partial u}{\partial x}\right) + [16xt - 2t - 16(v-1)](u-1) + 10xe^{-4x}; \quad (5.20)$$

$$\frac{\partial v}{\partial t} = \frac{\partial^2 v}{\partial x^2} + \frac{\partial u}{\partial x} + 4u - 4 + x^2 - 2t - 10te^{-4x}; \quad (5.21)$$

on the x interval $[0,1]$, with boundary conditions

$$u(t,0) = v(t,0) = 1.0 \qquad \text{at } x = 0; \quad (5.22)$$

$$3u + \frac{\partial u}{\partial x} = 3; \quad 5\frac{\partial v}{\partial x} = e^u(u-1) \qquad \text{at } x = 1 \quad (5.23)$$

and initial conditions

$$u(0,x) = v(0,x) = 1. \quad (5.24)$$

The exact solution for this problem is

$$u(t,x) = 1 + 10xte^{-4x}; \quad (5.25)$$

$$v(t,x) = 1 + x^2t. \quad (5.26)$$

Note that the initial conditions are consistent with the boundary as required by PDEMGS. We used a uniform mesh of 30 grid points and set $EPS =$

1×10^{-6} and $DTMIN = 1 \times 10^{-15}$ in Implicit GS and PDEMGS. The numerical results giving the 10th and 20th components of the u and v solution vectors, at $t = 0.01$, are shown in Table 5.4 and Table 5.5, respectively. From the table it is seen that PDEMGS is approximately 10 times faster than Implicit GS and the solutions obtained are almost identical.

D

Consider the following PDE

$$2\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + u - (4x+2)e^{x+t} + 2\cos t; \tag{5.27}$$

on the interval $[0,1]$ and for $0 < t \leq 0.1$.

The boundary and initial conditions for this problem are taken as

$$u(x,0) = (x^2-1)e^x; \tag{5.28}$$

$$u(-1,t) = u(1,t) = 0. \tag{5.29}$$

The exact solution for this problem is

$$u(x,t) = (x^2-1)e^{x+t} + \sin t. \tag{5.30}$$

We used a uniform mesh of 60 points and set $EPS = 1 \times 10^{-4}$ and $DTMIN = 1 \times 10^{-15}$ in Implicit GS and PDEMGS.

SOLVER	COMPUTING TIME (s)	$u^{10}(\times 10^{-5})$	$u^{20}(\times 10^{-5})$
EXACT SOLUTION	—	100807	100428
IMPLICIT GS	180	100896	100476
PDEMGS	11	100896	100476

Table 5.4: Numerical Results for Example 5.3.1(C) (u vector).

SOLVER	COMPUTING TIME (s)	$v^{10}(\times 10^{-5})$	$v^{20}(\times 10^{-5})$
EXACT SOLUTION	—	100086	100386
IMPLICIT GS	180	100119	100475
PDEMGS	11	100076	100433

Table 5.5: Numerical Results for Example 5.3.1(C) (v vector).

SOLVER	COMPUTING TIME (s)	$u^{20}(\times 10^{-4})$	$u^{50}(\times 10^{-3})$
EXACT SOLUTION	—	-5762	-1105
IMPLICIT GS	161	-5773	-1118
PDEMGS	11	-5353	-1023

Table 5.6: Numerical Results for Example 5.3.1(D).

The numerical results giving the 20th and 50th components of the solution vector u are shown in Table 5.6. From the table it is seen that PDEMGS is still faster than Implicit GS, in terms of computing speed, but in this case PDEMGS is not very accurate. The reason for this is discussed in section 5.5.

In the next example we consider a case where the initial condition is not consistent with the boundary conditions.

E

We consider the same problem as that in example 5.3.1 (A) but in this case the initial conditions and boundary conditions are not consistent. Here we

assume initial and boundary conditions of the form

$$u(x,0) = (x^2+1)e^x; \tag{5.31}$$

$$u(-1,t) = u(1,t) = 0; \tag{5.32}$$

The exact solution for this problem is still

$$u(x,t) = (x^2-1)e^{x+t}. \tag{5.33}$$

Proceeding in the same way as before, using a uniform mesh of 60 points, we obtained the results in Table 5.7. The table gives the 20th and 40th components of the solution vector u at $t = 0.1$. From the table we see that PDEMGS performed badly for this problem. The solution obtained was very inaccurate and not much gain in computing speed was obtained. The reason for this is that the initial conditions are not consistent with the boundary conditions and the basic assumptions of PDEMGS fail.

5.4 Physical Significance of PDEMGS

The occurence of stiff ODEs resulting from PDEs is frequently encountered in practice. To illustrate the physical significance of PDEMGS we select a problem from the field of atmospheric physics.

SOLVER	COMPUTING TIME (s)	$u^{20}(\times 10^{-4})$	$u^{40}(\times 10^{-3})$
EXACT SOLUTION	—	8722	1683
IMPLICIT GS	297	8393	1496
PDEMGS	247	6770	1064

Table 5.7: Numerical Results for Example 5.3.1(E).

Description of an atmosphere involves transport phenomena with chemical reactions. Thus stiffness can occur because the time constants for chemical reactions might be smaller than those for particle transport. The transport equations themselves can be stiff if the distances involved are large, and the chemical kinetic equations are almost always stiff.

The pollution caused by supersonic transport (SST) in the stratosphere has been studied by MacCracken [11]. The SST exhaust can disrupt the chemical balance of the stratosphere thus leading to a net decrease in ozone $[O_3]$. This would in turn lead to an increase in ultraviolet radiation on the surface of the earth.

5.4.1 Numerical Example

A simple one-dimensional model which describes the interactions of the chemical species O_2, O_3, NO, and NO_2 along with normal diffusion processes is given by the system of equations below

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + a_1 - a_2 u + a_3 v + a_4 y - a_5 uv - a_6 uy; \quad (5.34)$$

$$\frac{\partial v}{\partial t} = D\frac{\partial^2 v}{\partial x^2} + b_1 u - b_2 v + b_3 uv - b_4 vw; \quad (5.35)$$

$$\frac{\partial w}{\partial t} = D\frac{\partial^2 w}{\partial x^2} - c_1 w + c_2 y + c_3 uy - c_4 vw + 800 + S; \quad (5.36)$$

$$\frac{\partial y}{\partial t} = D\frac{\partial^2 y}{\partial x^2} - d_1 y + d_2 vw - d_3 uy + 800; \quad (5.37)$$

where

$$D = 10^{-4};$$

$$a_1 = 4 \times 10^5, \quad a_2 = 272.443800016, \quad a_3 = 10^{-4},$$

$$a_4 = 0.007, \quad a_5 = 3.67 \times 10^{-16}, \quad a_6 = 4.13 \times 10^{-12},$$

$$b_1 = 272.4438, \quad b_2 = 1.00016 \times 10^{-4}, \quad b_3 = 3.67 \times 10^{-15} \quad b_4 = 3.57 \times 10^{-15},$$

$$c_1 = 1.6 \times 10^{-8}, \quad c_2 = 0.007, \quad c_3 = 4.1283 \times 10^{-12} \quad c_4 = 3.57 \times 10^{-15},$$

$$d_1 = 7.000016 \times 10^{-3}, \quad d_2 = 3.57 \times 10^{-15}, \quad d_3 = 4.128 \times 10^{-12}$$

and the source producing NO is given by

$$S = \begin{cases} 3250 & \text{if } .475 \leq x \leq .575 \\ \\ 360 & \text{otherwise.} \end{cases} \tag{5.38}$$

The reflecting boundary conditions

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial x} = \frac{\partial w}{\partial x} = \frac{\partial y}{\partial x} = 0; \tag{5.39}$$

are assumed to hold at each end of the normalized spatial domain $[0, 1]$.

The variables u, v, w and y, respectively, represent the chemical species O_2, O_3, NO, NO_2 and are assumed to have the constant initial values

$$u(0, x) = 1.306028 \times 10^6, \tag{5.40}$$

$$v(0, x) = 1.076508 \times 10^{12}, \tag{5.41}$$

$$w(0, x) = 6.457715 \times 10^{10}, \tag{5.42}$$

$$y(0, x) = 3.542285 \times 10^{10}. \tag{5.43}$$

The problem (5.34)-(5.37) is difficult because the chemical reactions make the problem stiff and it is usually necessary to obtain solutions for very large real times. For this problem we used a uniform mesh of 31 points. We solved using Explicit GS, Implicit GS, PDEMGS and RK4. We set $EPS = 1 \times 10^{-5}$ and $DTMIN = 1 \times 10^{-10}$ in the first three solvers and $H = 1 \times 10^{-4}$ in

SOLVER	COMPUTING TIME (s)	$w^{10}(\times 10^6)$	$w^{20}(\times 10^6)$
EXPLICIT GS	389	64577	64577
PDEMGS	14	64576	64576
RK4	3587	64577	64577

Table 5.8: Numerical Results for Example 5.4.1.

RK4. Once again the Implicit GS failed to solve this problem, probably for the same reason as suspected in 5.3.1 (A). Table 5.8 gives the 10th and 20th components of the third solution sub-vector (NO) at $t = 1s$. From the table it is seen that PDEMGS is very much faster than Explicit GS and dramatically faster than RK4 and the solutions obtained are almost identical.

5.5 Limitations and General Remarks Concerning PDEMGS

Obviously a single program such as PDEMGS will not be able to solve all PDE problems because of certain restrictions that it impose on the class of problems it solves. These restrictions are listed below.

Firstly, the class of of problems that PDEMGS solves obviously depends on the choice of projection operators P_k used in MGS (see section 3.2.1). In this study we choose as projection operators the simple averaging operator and this may not be suitable for all problems.

Secondly, PDEMGS requires that the initial conditions be consistent with the boundary conditions as demonstrated in example 5.3.1 (F).

Thirdly, the assumption that the solution vector $x(t)$ be written as a product of the amplitude functions $\varphi(t)$ and shape vectors $v(t)$, as suggested in section 3.2.1, may not serve a purpose. In other words, the shape vector of the solution vector $x(t)$ may not have the property that it changes very rapidly with respect to time in only a very short time interval, assuming a relatively smooth shape, thereafter. Example 5.3.1 (E) is a case of where this happens.

When using a solver like PDEMGS it is customary to encounter certain criteria that might affect the computing time. For example, one such criterion is the time integration error tolerance which is set by the user. Obviously, it will cost more if a greater accuracy is desirable. As a general rule, it is neither necessary nor desirable to use a time integration error tolerance which is very much smaller than the error which is being produced by the spatial

discretization. However, if one wants to be sure that all the errors in the approximate solution are due to the spatial discretization, a relatively small time integration error tolerance should be used. The type of integration error control may vary for different integrators (for example, absolute, relative, mixed, etc.). We note that GEAR (see chapter 3), implemented in PDEMGS, uses a relative error bound.

In general, when solving problems using PDEMGS a very small initial time-step is used. One might wonder if this might have a significant effect on the computing time. In fact, GEAR usually requires a very small starting time-step and then automatically adjusts the step-size to an appropriate size without much sacrifice in computing time. Usually, it is preferable to start with a time-step which is too small rather than one which is too large, since a time-step which is too large may force the integrator to execute its error recovery mechanisms thus resulting in additional computational costs.

While GEAR is a very reliable ODE integrator, the user should note that, if modified so that the integration proceeds to the next desired output time (see chapter 4), it may lose its stability if forced, by the user, to change step-size too frequently.

Finally, as mentioned in section 5.2.1, we use the "dummy" ODE $\frac{dx_k^1}{dt} = 0$ for the boundary points where the solution value x_k^1 is known a priori from the boundary condition. This is done to maintain a convenient regular structure for the resulting ODE system (5.4), as required by MGS. Using this ad hoc device has the effect of eliminating the direct influence of this particular solution value from the time-step or error controls in GEAR. This is precisely what is desired, since known boundary values must be allowed to assume their true values (they have no discertization error). These values only indirectly affect the time-step choice or error control, through their influence on the behaviour of neighbouring solution values.

5.6 The Telegraph Equation

We consider the singularly perturbed telegraph equation in the following form

$$\varepsilon \frac{\partial^2 q}{\partial t^2} + A\frac{\partial q}{\partial t} = \mu \frac{\partial^2 q}{\partial \rho^2} - \gamma(t)q(\rho, t) + r(q, t); \tag{5.44}$$

where the unknown function q depends on a spatial variable ρ and time t, A is a constant matrix, μ is an arbitrary constant, γ an arbitrary function of t and r is a function of ρ and t.

Following a similar procedure to that presented in section 5.2.1 we express (5.44) in the multigrid form (5.4). For simplicity we assume that the spatial variable ρ changes from 0 to 1 and divide the interval $[0,1]$ into $n-1$ equal sub-intervals of length h. Then if we define

$$q\left(\frac{i-1}{n-1}, t\right) = x_i(t); \quad i = 1, 2, ..., n; \tag{5.45}$$

we can write (5.44) as the second order system of ODEs

$$\varepsilon \frac{d^2x}{dt^2} + A\frac{dx}{dt} + f(x,t) = 0; \tag{5.46}$$

where $f(x,t)$ is given by

$$f(x,t) = -\mu Dx + \gamma(t)x - r(q,t). \tag{5.47}$$

and the matrix D, representing the central difference approximation to the spatial derivative in (5.44), has the form

$$(Dx)_i = \frac{1}{h^2}\left(x_{i-1} - 2x_i + x_{i+1}\right); \; i = 2, 3, ..., n-1. \tag{5.48}$$

The form of $(Dx)_1$ and $(Dx)_n$ depends on the boundary conditions as in section 5.2.1.

If we set $\dfrac{dx}{dt} = z$ in (5.46) then we can write the second order system of ODEs (5.46) as the equivalent first order system

$$\frac{dx}{dt} = z; \tag{5.49}$$

$$\varepsilon \frac{dz}{dt} = -Az - f(x,t). \tag{5.50}$$

Here the sub-vector x represents the slow variable and the sub-vector z the fast variable. Replacing x by x_1 and z by x_2 we see that the system of equations given in (5.49)-(5.50) has the same form as that given in (5.13) i.e.

$$\frac{dx_k}{dt} = \tilde{f}_k(t, x_1, x_2); \quad k = 1, 2. \tag{5.51}$$

where the functions $\tilde{f}_k$ have the obvious definitions implied by the right-hand side of (5.49)-(5.50) .

The initial conditions for (5.51) are obtained from the initial conditions of (5.44) in a similar fashion to that described in section 5.2.1. Moreover, the ODE system (5.51) is in a form which is immediately useable with MGS.

5.6.1 Numerical Examples

A

We consider the solution of the following second order PDE in time

$$\frac{\partial^2 q}{\partial t^2} = \frac{\partial^2 q}{\partial \rho^2} + q - 2e^t e^\rho \cos\rho; \tag{5.52}$$

on the ρ interval $0 \leq \rho \leq \pi$ and for $0 < t \leq 0.1$ obtained by setting

$$\varepsilon = \mu = 1; \quad A = 0; \quad \gamma = -1; \quad r = -2e^t e^\rho \cos\rho; \tag{5.53}$$

in equation (5.44)

The boundary conditions and initial conditions for (5.52) are taken as

$$u(0,t) = u(\pi,t) = 0; \tag{5.54}$$

$$u(\rho,0) = e^{\rho}\sin\rho. \tag{5.55}$$

The exact solution for (5.52) is given by

$$u(\rho,t) = e^{t}e^{\rho}\sin\rho. \tag{5.56}$$

We used a uniform mesh of 30 grid points and set $EPS = 1 \times 10^{-5}$ and $DTMIN = 1 \times 10^{-15}$ in Implicit GS and MGS. The numerical results giving the 5th and 25th components of the solution vector q, at $t = 0.1$, are shown in Table 5.9. We observed that Implicit GS gave completely incorrect results for this problem. PDEMGS, on the other hand, gave a more acceptable solution with reduced computational effort.

This problem illustrates a further advantage of PDEMGS namely that for certain problems MGS might be slightly more reliable than the standard solvers.

SOLVER	COMPUTING TIME (s)	$q^5(\times 10^{-4})$	$q^{25}(\times 10^{-3})$
EXACT SOLUTION	—	7157	7670
IMPLICIT GS	145	8553	23112
MGS	11	7157	7550

Table 5.9: Numerical Results for Example 5.6.1(A).

B

In this example we consider the solution of the following PDE:

$$\frac{\partial^2 q}{\partial t^2} + \frac{\partial q}{\partial t} = \frac{\partial^2 q}{\partial \rho^2} - q + \pi \cos \pi t, \tag{5.57}$$

on the ρ interval $-1 \leq \rho \leq 1$ and for $0 < t \leq 1$, obtained by setting

$$\varepsilon = \mu = 1, \quad A = I, \quad \gamma = 1, \quad r = \pi \cos \pi t \tag{5.58}$$

in (5.44).

As boundary conditions for this problem we take

$$\frac{\partial q}{\partial \rho}(-1, t) = \frac{\partial q}{\partial \rho}(1, t) = 0; \tag{5.59}$$

SOLVER	COMPUTING TIME (s)	$q^5(\times 10^{-5})$	$q^{25}(\times 10^{-5})$
IMPLICIT GS	158	274623	274623
MGS	6	274623	274623

Table 5.10: Numerical Results for Example 5.6.1(B).

and as initial conditions we take

$$q(\rho, 0) = (1, ..., 1); \tag{5.60}$$

$$\frac{\partial q}{\partial t}(\rho, 0) = (\pi, ..., \pi). \tag{5.61}$$

We solved this problem in exactly the same way as the previous problem with all the required parameters in Implicit GS and MGS unchanged. The numerical results giving the 5th and 25th components of the solution vector q at $t = 1.0s$ are shown in Table 5.10. From the table it is seen that the results obtained for this problem using MGS is exactly the same as Implicit GS but MGS is considerably faster than Implicit GS.

Chapter 6

Conclusion

We have seen that the MOL is a well established technique for the numerical solution of PDEs. When this technique is applied to a PDE a stiff system of ODEs usually results. The recent approach for solving such systems on large mainframe computers has been to take advantage of parallel computation (e.g. partitioning of a system into stiff and non-stiff sub-systems). On the other hand, the common PC user still relies heavily on a stiff ODE solver based on, typically, Backward Differentiation Formulas (BDFs) for treating such systems. Since BDF is an implicit method an algebraic system needs to be solved at each time step. It is generally recognized that for physically realistic PDE systems (e.g. the system in section 5.4) the problems are

so difficult or so large that a direct application of the BDF codes can be prohibitive in both cost and storage requirement, or may not be applicable at all (see section 5.4). In such cases it may be necessary to use specific ODE solution methods which are capable of exploiting the special features of the system to be solved.

From the discussions and results presented in this thesis it is probably safe to conclude that the multigrid method seems to be one such viable solution method. The method has been applied successfully to single equations, systems of equations, linear and nonlinear equations, parabolic and hyperbolic equations, linear and nonlinear boundary conditions, or a combination of these. We have also indicated the feasibility of using the multigrid method in practical applications by considering Burger's equation, occuring in many real life situations, and solving a difficult PDE system in chemical kinetics (see section 5.4). Moreover, most of the preceeding problems (except the ones in section 5.6) were solved with a single program, PDEMGS, resulting in both reduction in computing time and human effort.

We feel that this should be sufficient justification to indicate the potential usefulness of the multigrid method for solving ODEs resulting from PDEs, and the versatility of the software interface PDEMGS.

Finally, we would like to mention that the modified multigrid method, presented in chapter 3, where innner iterations are introduced in each cycle seems to be a promising way of implementing the multigrid algorithm. The code MGS2 implements this algorithm using RK4 for the integration of the amplitude equations and RK2 for the solution of the shape equations. It was found that MGS2 was more accurate than MGS1, which is the code for implementing the original multigrid method [14] using the same solvers as MGS2 for the integration of the amplitude and shape equations. Although MGS2 was slightly slower than MGS1 it was still considerably faster than the standard GS.

Bibliography

[1] Aiken, R.C., Ed., *Stiff Computation*, Oxford University Press, New York, 1985.

[2] Cash, J.R., and Gladwell, I., Ed., *Computational Ordinary Differential Equations*, Clarendon Press, Oxford, 1992.

[3] Chin, R.C.Y., Hedstrom, G.W., and Karlson, E., A simplified Galerkin Method for Hyperbolic Equations, *Math. Comp.*, **33**, 647, 1979.

[4] Curtis, A.R., *Solution of Large, Stiff Initial Value Problems - The State of the Art*, Numerical Software - Needs and Availability, Ed. Jacobs, D., Academic Press, p257.

[5] Dahlquist, G., Bjorck, A., *Numerical Methods*, Prentice-Hall, New York, 1974.

[6] Dekker, K., and Verwer, J.G., *Stability of Runge-Kutta Methods for Stiff Nonlinear Differential Equations*, North-Holland, New York, 1984

[7] Gear, C.W., *Numerical Initial Value problems in Ordinary Differential Equations*, Prentice-Hall, Engelwood Cliffs, New Jersey, 1971.

[8] Gupta, G.K., Sacks-Davis, R., and Tischer, P.E., A Review of Recent Developments in Solving ODEs, *ACM Computing Surveys*, **17**, 5, 1985.

[9] Hairer, E. Nørsett, S.P. Wanner, G. *Solving Ordinary Differential Equations*, Z Voll., Springer-Verlag, 1987,1991.

[10] Hindmarsh, A.C., ODEPACK, a systemized Collection of ODE Solvers, Scientific Computing, Ed. Stepleman, R.S., *et.al.*, North-Holland, Amsredam, 1983, p.55.

[11] Hindmarsh, A.C., Applications to atmospheric models at LLNL, *Stiff Computation*, Ed. Aiken, R.C., Oxford University Press, New York, 1985.

[12] Hindmarsh, A.C., and Byrne, G.D., Application of EPISODE: An Experimental Package for the Integration of Systems of Ordinary Differen-

tial Equations, *Numer. Methods for Differential Systems*, Eds. Lapidus, L., and Schiesser, W.E., Academic Press, p.147, 1976.

[13] Jespersen, Dennis, C., Multigrid Methods for Partial Differential Equations, in *Studies in Mathematics Vol.24*, Ed. Gene H. Golub.

[14] Kozakiewicz, J.M., and Mika, J.R., Multigrid Method for Numerical Solution of Ordinary Differential Equations, *Zastosowania Matematyki*, in print.

[15] Lambert, J.D., *Computational Methods in Ordinary Differential Equations*, John Wiley and Sons, 1973.

[16] Madsen, N.K., and Sincovec, R.F., General Software for Partial Differential Equations, *Numer. Methods for Differential Systems*, Eds., Lapidus, L., and Schiesser, W.E., Academic Press, p.229, 1976.

[17] Mika, J.R., Mathematical foundations of the quasistatic approximation in reactor physics, *Annals of Nuclear Energy*, **9**, 585, 1982.

[18] Mika, J.R., and Scribani, I., Multigrid line method for partial differential equations, Presented at the South African Mathematical Society Congress, Mmabatho, October 1990.

[19] Miranker, W.L., *Numerical Methods for Stiff Equations*, Reidel, Boston, 1981.

[20] Shampine, L.F., and Gear, C.W., A User's View of Solving Stiff Differential Equations, *SIAM Review*, **21**, 1, 1979.

[21] Seward, W.L., Fairweather, G. and Johnston, R.L., A Survey of Higher-order Methods for the Numerical Integration of Semidiscrete Parabolic Problems, *IMA J. of Numer. Analysis*, **4**, 375, 1984.

[22] Schiesser, W.E., Some characteristics of ODE problems generated by the method of lines, *Stiff Computation*, Ed., Aiken, R.C., Oxford University Press, p.139, 1985.

[23] Varga, R.S., *Matrix Iterative Analysis*, Prentice-Hall, Engelwood Cliffs, New Jersey, 1962.

THE APPLICATION OF THE MULTIGRID ALGORITHM TO THE SOLUTION OF STIFF ORDINARY DIFFERENTIAL EQUATIONS RESULTING FROM PARTIAL DIFFERENTIAL EQUATIONS (VOLUME 2)

Nabendra Parumasur

1992

Contents

Appendix A

Numerical Results

Appendix 1 contains additional tables for examples 3.5, 4.3 and 5.3.1 (A). In the tables below the parameters have the following meaning :
N - number of equations
Y - solution vector
T - current time
EPS - time integration error tolerance
DTMIN - initial time-step
L - number of time-steps in amplitude equations for one step in shape equations
H - fixed step-size

*** NUMERICAL RESULTS FOR EXAMPLE 3.5 ***

GEAR REVISED INTEGRATION RULE
GEAR STIFFLY STABLE METHOD 22

N = 60 EPS = 1.0E-04 DTMIN = 1.0E-15

T	Y(1)	Y(10)	Y(40)	Y(50)
0.000E+00	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
1.000E-03	2.724040E+00	2.748672E+00	1.001174E+00	1.001175E+00
2.000E-03	7.420243E+00	7.555052E+00	1.002652E+00	1.002660E+00
3.000E-03	2.021172E+01	2.076500E+01	1.004965E+00	1.005008E+00
4.000E-03	5.505441E+01	5.707297E+01	1.009576E+00	1.009751E+00
5.000E-03	1.499602E+02	1.568642E+02	1.020507E+00	1.021147E+00
6.000E-03	4.085469E+02	4.312236E+02	1.048828E+00	1.051038E+00
7.000E-03	1.113270E+03	1.185708E+03	1.125030E+00	1.132359E+00
8.000E-03	3.032329E+03	3.258838E+03	1.332861E+00	1.356492E+00
9.000E-03	8.259330E+03	8.956534E+03	1.902861E+00	1.977512E+00
1.000E-02	2.249655E+04	2.461614E+04	3.469260E+00	3.701414E+00
1.100E-02	6.127605E+04	6.765561E+04	7.776953E+00	8.490074E+00
1.200E-02	1.669106E+05	1.859546E+05	1.962723E+01	2.179611E+01
1.300E-02	4.546278E+05	5.110788E+05	5.222646E+01	5.876810E+01
1.400E-02	1.238323E+06	1.404672E+06	1.419123E+02	1.615067E+02
1.500E-02	3.373197E+06	3.860942E+06	3.886818E+02	4.470341E+02

COMPUTING TIME WAS 326 S

RK4-RK2 MULTIGRID SOLVER

N = 60 H = 1.0E-04 L = 10

T	Y(1)	Y(10)	Y(40)	Y(50)
0.000E+00	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
1.000E-03	2.743640E+00	2.756077E+00	1.001174E+00	1.001174E+00
2.000E-03	7.472816E+00	7.574549E+00	1.002653E+00	1.002659E+00
4.000E-03	5.543510E+01	5.721029E+01	1.009578E+00	1.009745E+00
5.000E-03	1.509850E+02	1.572286E+02	1.020502E+00	1.021135E+00
6.000E-03	4.112282E+02	4.321047E+02	1.048789E+00	1.051007E+00
7.000E-03	1.120037E+03	1.187536E+03	1.124835E+00	1.132233E+00
8.000E-03	3.050578E+03	3.263661E+03	1.332346E+00	1.356050E+00
9.000E-03	8.308687E+03	8.969407E+03	1.902129E+00	1.975360E+00
1.000E-02	2.262992E+04	2.465034E+04	3.470416E+00	3.691487E+00
1.100E-02	6.163597E+04	6.774581E+04	7.786423E+00	8.454568E+00
1.200E-02	1.678750E+05	1.861840E+05	1.965648E+01	2.169010E+01
1.300E-02	4.572334E+05	5.116853E+05	5.229816E+01	5.847914E+01
1.400E-02	1.245347E+06	1.406254E+06	1.420710E+02	1.607299E+02
1.500E-02	3.391904E+06	3.864785E+06	3.889982E+02	4.448954E+02

COMPUTING TIME WAS 17 S

IMPROVED RK4-RK2 MULTIGRID SOLVER

N = 60 H = 1.0E-04 L = 10

T	Y(1)	Y(10)	Y(40)	Y(50)
0.000E+00	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
1.000E-03	2.723913E+00	2.748413E+00	1.001173E+00	1.001175E+00
2.000E-03	7.419744E+00	7.553473E+00	1.002649E+00	1.002663E+00
4.000E-03	5.505132E+01	5.705096E+01	1.009533E+00	1.009797E+00
5.000E-03	1.499535E+02	1.567906E+02	1.020361E+00	1.021299E+00
6.000E-03	4.084566E+02	4.309009E+02	1.048362E+00	1.051500E+00
7.000E-03	1.112592E+03	1.184228E+03	1.123635E+00	1.133606E+00
8.000E-03	3.030590E+03	3.254572E+03	1.329468E+00	1.359325E+00
9.000E-03	8.255040E+03	8.944440E+03	1.896715E+00	1.981541E+00
1.000E-02	2.248600E+04	2.458176E+04	3.461349E+00	3.702096E+00
1.100E-02	6.124999E+04	6.755749E+04	7.768036E+00	8.476838E+00
1.200E-02	1.668402E+05	1.856670E+05	1.961086E+01	2.174657E+01
1.300E-02	4.544610E+05	5.102658E+05	5.217584E+01	5.863226E+01
1.400E-02	1.237922E+06	1.402358E+06	1.417362E+02	1.611520E+02
1.500E-02	3.372027E+06	3.854093E+06	3.880781E+02	4.460629E+02

COMPUTING TIME WAS 35 S

*** END OF NUMERICAL RESULTS FOR EXAMPLE 3.5 ***

*** NUMERICAL RESULTS FOR EXAMPLE 4.3 ***

EXACT SOLUTION

N = 50

T	Y(10)	Y(20)	Y(30)	Y(40)
0.000E+00	9.980193E-01	4.997803E-01	1.008753E-01	1000000E-01
1.000E-02	9.989379E-01	4.998255E-01	1.010686E-01	1000000E-01
2.000E-02	9.994309E-01	4.998673E-01	1.013044E-01	1000000E-01
3.000E-02	9.996952E-01	4.999102E-01	1.015921E-01	1000000E-01
4.000E-02	9.998368E-01	4.999619E-01	1.019428E-01	1000000E-01
5.000E-02	9.999126E-01	5.000349E-01	1.023704E-01	1000000E-01
6.000E-02	9.999532E-01	5.001519E-01	1.028915E-01	1000000E-01
7.000E-02	9.999750E-01	5.003548E-01	1.035260E-01	1000000E-01
8.000E-02	9.999866E-01	5.007206E-01	1.042983E-01	1000000E-01
9.000E-02	9.999928E-01	5.013920E-01	1.052375E-01	1000000E-01
1.000E-01	9.999962E-01	5.026331E-01	1.063785E-01	1000000E-01

GEAR REVISED INTEGRATION RULE
IMPLICIT ADAMS METHOD 10

N = 50 EPS = 1.0E-05 DTMIN = 1.0E-08

T	Y(10)	Y(20)	Y(30)	Y(40)
0.000E+00	9.980193E-01	4.997803E-01	1.008753E-01	1.000000E-01
1.000E-02	9.974571E-01	4.997443E-01	1.010197E-01	1.000000E-01
2.000E-02	9.967867E-01	4.997027E-01	1.011852E-01	1.000000E-01
3.000E-02	9.960123E-01	4.996550E-01	1.013732E-01	1.000000E-01
4.000E-02	9.951400E-01	4.996004E-01	1.015847E-01	1.000000E-01
5.000E-02	9.941776E-01	4.995386E-01	1.018202E-01	1.000000E-01
6.000E-02	9.931332E-01	4.994689E-01	1.020802E-01	1.000000E-01
7.000E-02	9.920153E-01	4.993912E-01	1.023648E-01	1.000000E-01
8.000E-02	9.908325E-01	4.993050E-01	1.026736E-01	1.000000E-01
9.000E-02	9.895930E-01	4.992103E-01	1.030062E-01	1.000000E-01
1.000E-01	9.883046E-01	4.991069E-01	1.033622E-01	1.000000E-01

COMPUTING TIME WAS 11 S

GEAR REVISED INTEGRATION RULE
GEAR STIFFLY STABLE METHOD 22

N = 50   EPS = 1.0E-05   DTMIN = 1.0E-08

T	Y(10)	Y(20)	Y(30)	Y(40)
0.000E+00	9.980193E-01	4.997803E-01	1.008753E-01	1.000000E-01
1.000E-02	9.974571E-01	4.997443E-01	1.010197E-01	1.000000E-01
2.000E-02	9.967868E-01	4.997027E-01	1.011852E-01	1.000000E-01
3.000E-02	9.960123E-01	4.996550E-01	1.013732E-01	1.000000E-01
4.000E-02	9.951401E-01	4.996004E-01	1.015846E-01	1.000000E-01
5.000E-02	9.941776E-01	4.995386E-01	1.018202E-01	1.000000E-01
6.000E-02	9.931332E-01	4.994689E-01	1.020802E-01	1.000000E-01
7.000E-02	9.920153E-01	4.993912E-01	1.023647E-01	1.000000E-01
8.000E-02	9.908325E-01	4.993050E-01	1.026735E-01	1.000000E-01
9.000E-02	9.895929E-01	4.992103E-01	1.030062E-01	1.000000E-01
1.000E-01	9.883045E-01	4.991069E-01	1.033622E-01	1.000000E-01

COMPUTING TIME WAS   75 S

GEAR REVISED INTEGRATION RULE
GEAR STIFFLY STABLE METHOD 23

N = 50 EPS = 1.0E-05 DTMIN = 1.0E-08

T	Y(10)	Y(20)	Y(30)	Y(40)
0.000E+00	9.980193E-01	4.997803E-01	1.008753E-01	1.000000E-01
1.000E-02	9.974571E-01	4.997443E-01	1.010197E-01	1.000000E-01
2.000E-02	9.967870E-01	4.997027E-01	1.011852E-01	1.000000E-01
3.000E-02	9.960127E-01	4.996550E-01	1.013732E-01	1.000000E-01
4.000E-02	9.951404E-01	4.996004E-01	1.015846E-01	1.000000E-01
5.000E-02	9.941779E-01	4.995386E-01	1.018202E-01	1.000000E-01
6.000E-02	9.931334E-01	4.994689E-01	1.020802E-01	1.000000E-01
7.000E-02	9.920155E-01	4.993912E-01	1.023647E-01	1.000000E-01
8.000E-02	9.908327E-01	4.993050E-01	1.026735E-01	1.000000E-01
9.000E-02	9.895931E-01	4.992103E-01	1.030062E-01	1.000000E-01
1.000E-01	9.883047E-01	4.991069E-01	1.033622E-01	1.000000E-01

COMPUTING TIME WAS 10 S

GEAR REVISED INTEGRATION RULE
GEAR STIFFLY STABLE METHOD 21

N = 50 EPS= 1.0E-05 DTMIN = 1.0E-08

T	Y(10)	Y(20)	Y(30)	Y(40)
0.000E+00	9.980193E-01	4.997803E-01	1.008753E-01	1.000000E-01
1.000E-02	9.974571E-01	4.997443E-01	1.010197E-01	1.000000E-01
2.000E-02	9.967868E-01	4.997027E-01	1.011852E-01	1.000000E-01
3.000E-02	9.960124E-01	4.996550E-01	1.013732E-01	1.000000E-01
4.000E-02	9.951402E-01	4.996004E-01	1.015846E-01	1.000000E-01
5.000E-02	9.941777E-01	4.995386E-01	1.018202E-01	1.000000E-01
6.000E-02	9.931333E-01	4.994689E-01	1.020802E-01	1.000000E-01
7.000E-02	9.920153E-01	4.993912E-01	1.023647E-01	1.000000E-01
8.000E-02	9.908325E-01	4.993050E-01	1.026735E-01	1.000000E-01
9.000E-02	9.895930E-01	4.992103E-01	1.030062E-01	1.000000E-01
1.000E-01	9.883046E-01	4.991069E-01	1.033622E-01	1.000000E-01

COMPUTING TIME WAS 46 S

FOURTH ORDER RUNGE-KUTTA METHOD

N = 50 H = 1.0E-05

T	Y(10)	Y(20)	Y(30)	Y(40)
0.000E+00	9.980193E-01	4.997803E-01	1.008753E-01	1.000000E-01
1.000E-02	9.974564E-01	4.997443E-01	1.010199E-01	1.000000E-01
2.000E-02	9.967860E-01	4.997027E-01	1.011854E-01	1.000000E-01
3.000E-02	9.960114E-01	4.996549E-01	1.013734E-01	1.000000E-01
4.000E-02	9.951400E-01	4.996004E-01	1.015847E-01	1.000000E-01
5.000E-02	9.941776E-01	4.995386E-01	1.018202E-01	1.000000E-01
6.000E-02	9.931332E-01	4.994689E-01	1.020802E-01	1.000000E-01
7.000E-02	9.920153E-01	4.993912E-01	1.023648E-01	1.000000E-01
8.000E-02	9.908325E-01	4.993050E-01	1.026736E-01	1.000000E-01
9.000E-02	9.895917E-01	4.992102E-01	1.030066E-01	1.000000E-01
1.000E-01	9.883033E-01	4.991068E-01	1.033626E-01	1.000000E-01

COMPUTING TIME WAS 1787 S

*** END OF NUMERICAL RESULTS FOR EXAMPLE 4.3 ***

*** NUMERICAL RESULTS FOR EXAMPLE 5.3.1 (A) ***

EXACT SOLUTION

N = 60

T	Y(10)	Y(20)	Y(30)	Y(40)
0.000E+00	-2.580897E-01	-6.117728E-01	-9.829112E-01	-1.236824E+00
1.000E-02	-2.606835E-01	-6.179212E-01	-9.927897E-01	-1.249254E+00
2.000E-02	-2.633034E-01	-6.241314E-01	-1.002767E+00	-1.261810E+00
3.000E-02	-2.659497E-01	-6.304041E-01	-1.012845E+00	-1.274491E+00
4.000E-02	-2.686225E-01	-6.367397E-01	-1.023024E+00	-1.287300E+00
5.000E-02	-2.713222E-01	-6.431391E-01	-1.033306E+00	-1.300238E+00
6.000E-02	-2.740490E-01	-6.496027E-01	-1.043691E+00	-1.313305E+00
7.000E-02	-2.768033E-01	-6.561313E-01	-1.054180E+00	-1.326504E+00
8.000E-02	-2.795852E-01	-6.627256E-01	-1.064775E+00	-1.339836E+00
9.000E-02	-2.823951E-01	-6.693861E-01	-1.075476E+00	-1.353301E+00
1.000E-01	-2.852332E-01	-6.761135E-01	-1.086284E+00	-1.366902E+00

```
                    GEAR REVISED INTEGRATION RULE
                    GEAR STIFFLY STABLE METHOD  22

                 N = 60    EPS = 1.0E-05    DTMIN = 1.0E-15

      T           Y( 10)         Y( 20)         Y( 30)         Y( 40)

0.000E+00 -2.580897E-01 -6.117728E-01 -9.829112E-01 -1.236825E+00
1.000E-02 -2.606821E-01 -6.179184E-01 -9.927845E-01 -1.249246E+00
2.000E-02 -2.633005E-01 -6.241257E-01 -1.002757E+00 -1.261792E+00
3.000E-02 -2.659453E-01 -6.303953E-01 -1.012829E+00 -1.274463E+00
4.000E-02 -2.686166E-01 -6.367279E-01 -1.023003E+00 -1.287262E+00
5.000E-02 -2.713148E-01 -6.431241E-01 -1.033279E+00 -1.300190E+00
6.000E-02 -2.740401E-01 -6.495845E-01 -1.043658E+00 -1.313247E+00
7.000E-02 -2.767927E-01 -6.561098E-01 -1.054141E+00 -1.326435E+00
8.000E-02 -2.795731E-01 -6.627006E-01 -1.064729E+00 -1.339756E+00
9.000E-02 -2.823814E-01 -6.693575E-01 -1.075424E+00 -1.353211E+00
1.000E-01 -2.852179E-01 -6.760814E-01 -1.086226E+00 -1.366801E+00

COMPUTING TIME WAS     107 S
```

PDEMGS

N = 60 EPS = 1.0E-05 NFI= 10 DTMIN= 1.0E-15

T	Y(10)	Y(20)	Y(30)	Y(40)
0.000E+00	-2.580897E-01	-6.117728E-01	-9.829112E-01	-1.236825E+00
1.000E-02	-2.606820E-01	-6.179183E-01	-9.927843E-01	-1.249246E+00
2.000E-02	-2.633003E-01	-6.241252E-01	-1.002756E+00	-1.261790E+00
3.000E-02	-2.659450E-01	-6.303946E-01	-1.012828E+00	-1.274462E+00
4.000E-02	-2.686163E-01	-6.367270E-01	-1.023001E+00	-1.287260E+00
5.000E-02	-2.713187E-01	-6.431226E-01	-1.033276E+00	-1.300186E+00
6.000E-02	-2.740394E-01	-6.495826E-01	-1.043655E+00	-1.313243E+00
7.000E-02	-2.767922E-01	-6.561077E-01	-1.054137E+00	-1.326431E+00
8.000E-02	-2.795723E-01	-6.626982E-01	-1.064726E+00	-1.339751E+00
9.000E-02	-2.823807E-01	-6.693550E-01	-1.075420E+00	-1.353205E+00
1.000E-01	-2.852174E-01	-6.760785E-01	-1.086221E+00	-1.366792E+00

COMPUTING TIME WAS 10 S

*** END OF NUMERICAL RESULTS FOR EXAMPLE 5.3.1 (A) ***

Appendix B

Computer Programs TRAP and RK4

```
      PROGRAM TRMAIN
C
C -------------------------------------------------
C
C MAIN PROGRAM FOR SOLVING THE SYSTEM AY'(T)=F(T,Y) IN
C EXAMPLE 2.5 THIS PROGRAM CALLS THE DRIVER ROUTINE
C TRAP WHICH IS RESPONSIBLE FOR PERFORMING THE
C INTEGRATION. THIS PROGRAM IS ALSO RESPONSIBLE FOR
C THE INITIALISATION OF PARAMETERS REQUIRED BY TRAP
C AND SETTING OF THE INITIAL CONDITIONS
C
C -------------------------------------------------
C
C SEE DRIVER ROUTINE TRAP FOR DEFINITION OF PARAMETERS
C
C -------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON/SYSVAR/N,A(3,3),Y0(3),T0
      COMMON/STATE1/H,ITERMAX
      COMMON/OTHER/DTOUT,TMAX,NCOL1,NCOL2,NCOL3,TNEXT
      DATA N,T0/3,0.D0/
```

```
      DATA H,ITERMAX/0.01D0,10/
      DATA Y0/1.D0,0.D0,-1.D0/
      DATA A/-21,19,40,19,-21,-40,-20,20,-40/
      DATA TMAX,DTOUT/1.1D0,1.D-1/
      DATA NCOL1,NCOL2,NCOL3/1,2,3/
C
      WRITE(*,100)
100   FORMAT(3X,'IMPLICIT TRAPEZOIDAL RULE')
      WRITE(*,*)'H = ',H
      WRITE(*,200) NCOL1,NCOL2,NCOL3
200   FORMAT(/5X,'T',10X,'Y(',I3,')',8X,'Y(',I3,')',8X,
     * 'Y(',I3,')'/)
      WRITE(*,201) T0,Y0(NCOL1),Y0(NCOL2),Y0(NCOL3)
201   FORMAT(1PE10.3,5(1X,1PE13.6))
      TNEXT=T0+DTOUT
C
C CALL DRIVER ROUTINE TRAP
C
      CALL TRAP
C
      END
```

```
      SUBROUTINE TRAP
C
C -------------------------------------------------
C
C     SUBROUTINE TRAP :
C                       THIS ROUTINE IS THE DRIVER
C     ROUTINE FOR INTEGRATING THE SYSTEM AY'(T)=F(T,Y)
C     IN EXAMPLE 2.5. THE ROUTINE IMPLEMENTS THE
C     IMPLICIT TRAPEZOIDAL RULE USING A GENERALISED
C     NEWTON'S ITERATION PROCEDURE. A STARTING
C     APPROXIMATION FOR THE ITERATION IS OBTAINED USING
C     EULER'S METHOD. AN EXACT JACOBIAN MATRIX WAS
C     SUPPLIED AND THE ALGEBRAIC SYSTEM RESULTING FROM
C     IMPLICIT INTEGRATION WAS SOLVED USING LU
C     DECOMPOSITION.
C
C -------------------------------------------------
C
C     PARAMETERS ARE :
C
C     A        - THE N X N MATRIX OF COEFFICIENTS
C     N        - NUMBER OF EQUATIONS
C     H        - TIMESTEP
C     YO       - VECTOR OF DIMENSION N CONTAINING THE
C                INITIAL CONDITIONS. THE SOLUTION VECTOR
C                IS RETURNED IN THIS VECTOR
C     TO       - INITIAL TIME
C     ITERMAX  - MAXIMUM NUMBER OF NEWTON ITERATIONS
C     RJ       - N X N JACOBIN MATRIX
C     DTOUT    - INTERVALS FOR PRINTING OF OUTPUT
c                RESULTS
C     TMAX     - FINAL TIME
C     NCOL1...- COLUMNS FOR OUTPUT.
C
C -------------------------------------------------
C
```

```
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON/SYSVAR/N,A(3,3),Y0(3),T0
      COMMON/STATE1/H,ITERMAX
      COMMON/STATE2/FPYB(3,3),RL(3,3),RU(3,3)
      COMMON/STATE3/FYB(3),X(3)
      COMMON/STATE4/RJ(3,3)
      COMMON/OTHER/DTOUT,TMAX,NCOL1,NCOL2,NCOL3,TNEXT
      DIMENSION ID(3,3),YB(3),DY(3),DYOLD(3)
C
      DO 50 I=1,N
      DO 60 J=1,N
      ID(I,I)=1.D0
      IF (I.NE.J) THEN
      ID(I,J)=0.D0
      ENDIF
60    CONTINUE
50    CONTINUE

C GET INITIAL APPROXIMATION FOR NEWTON ITERATION
C USING EULER'S METHOD

      CALL DIFEQ(N,T0,A,Y0,DY)
      DO 10 I=1,N
       YB(I)=Y0(I)+H*DY(I)
10    CONTINUE
C
C BEGIN NEWTON ITERATION
C
20    CONTINUE
      DO 200 ITER=1,ITERMAX
      DO 30 I=1,N
      DYOLD(I)=DY(I)
30    CONTINUE
      T1=T0+H
      CALL DIFEQ(N,T1,A,YB,DY)
      DO 40 I=1,N
       FYB(I)=YB(I)-Y0(I)-(H/2.D0)*DYOLD(I)-(H/2.D0)*DY(I)
```

```
       FYB(I)=-FYB(I)
40     CONTINUE
C
C COMPUTE THE JACOBIAN MATRIX
C
       CALL RJAC
C
       DO 70 I=1,N
       DO 80 J=1,N
       FPYB(I,J)=ID(I,J)-(H/2.D0)*RJ(I,J)
80     CONTINUE
70     CONTINUE
C
C SOLVE THE LINEAR ALGEBRAIC SYSTEM IN NEWTON'S METHOD
C
       CALL LUDC
       CALL SOLVE
C
       DO 90 I=1,N
90      YB(I)=YB(I)+X(I)
200    CONTINUE
       DO 100 I=1,N
          Y0(I)=YB(I)
100    CONTINUE
       IF (T0.GE.TNEXT) THEN
       WRITE(*,201) T0,Y0(NCOL1),Y0(NCOL2),Y0(NCOL3)
       TNEXT=TNEXT+DTOUT
       ENDIF
       T0=T0+H
       CALL DIFEQ(N,T0,A,Y0,DY)
       IF(T0.LE.TMAX) GOTO 20
201    FORMAT(1PE10.3,5(1X,1PE13.6))
C
       RETURN
       END
```

```
      SUBROUTINE LUDC
C
C -----------------------------------------------------
C
C     SUBROUTINE LUDC :
C                          THIS SUBROUTINE COMPUTES THE L
C     AND U TRIANGULAR MATRICES EQUIVALENT TO THE
C     MATRIX A, SUCH THAT LU=A.
C
C -----------------------------------------------------
C
C     PARAMETERS ARE :
C     RP  - THE N X N MATRIX TO BE REDUCED
C     N   - DIMENSION OF COEFFICIENT MATRIX RP
C     RL  - REDUCED LOWER TRIANGULAR  FORM OF A
C     RU  - REDUCED UPPER TRIANGULAR  FORM OF A
C
C -----------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      COMMON/SYSVAR/N,A(3,3),Y0(3),T0
      COMMON/STATE2/RP(3,3),RL(3,3),RU(3,3)
C
C FIRST FIRST COLUMN OF L EQUALS FIRST COLUMN OF A
C
      DO 10 I=1,N
10    RL(I,1)=RP(I,1)
C
C IF DIAGONAL ELEMENT VERY SMALL PRINT ERROR MESSAGE
C AND RETURN
C
      IF (ABS(RP(1,1)).LT.1.D-5) THEN
        WRITE(*,100)
        RETURN
      ENDIF
C
C
```

```
C COMPUTE ELEMENTS FOR FIRST ROW OF U
C
      DO 20 J=1,N
20    RU(1,J)=RP(1,J)/RL(1,1)
C
C COMPLETE COMPUTING THE L AND U ELEMENTS. THE IDEA IS
C TO COMPUTE A COLUMN OF L'S, AND THEN A ROW OF U'S.
C
      DO 30 J=2,N
C
C FIRST COMPUTE A COLUMN OF L'S
C
       DO 40 I=J,N
        SUM1=0.D0
        DO 50 K=1,J-1
        SUM1=SUM1+RL(I,K)*RU(K,J)
50      CONTINUE
       RL(I,J)=RP(I,J)-SUM1
40     CONTINUE
C
C TEST FOR TOO SMALL DIAGONAL ELEMENT
C
      IF (ABS(RP(J,J)).LT.1.D-5) THEN
        WRITE(*,100)
        RETURN
       ENDIF
C
C NOW GET A ROW OF U'S
C
       RU(J,J)=1.D0
       DO 60 I=J+1,N
        SUM2=0.D0
        DO 80 K=1,J-1
        SUM2=SUM2+RL(J,K)*RU(K,I)
80      CONTINUE
       RU(J,I)=(RP(J,I)-SUM2)/RL(J,J)
60     CONTINUE
```

```
30    CONTINUE
C
100   FORMAT(/,' VERY SMALL DIAGONAL ELEMENT INDICATES',
     *'A NEARLY SINGULAR MATRIX.')
C
      RETURN
      END
```

```
      SUBROUTINE SOLVE
C
C -----------------------------------------------------
C
C     SUBROUTINE SOLVE :
C                       THIS SUBROUTINE IS USED TO FIND
c     THE SOLUTION TO A SYSTEM OF EQUATIONS, AX = B,
c     AFTER THE L AND U MATRICES OF A HAS BEEN FOUND.
c     THE SOLUTION IS RETURNED IN THE VECTOR X.
C
C -----------------------------------------------------
C
C     PARAMETERS ARE :
C     RL - REDUCED LOWER TRIANGULAR FORM OF A
C     RU - REDUCED UPPER TRIANGULAR FORM OF A
C     N  - DIMENSION OF A
C     B  - THE VECTOR OF RIGHT HAND SIDES
C     X  - SOLUTION VECTOR
C
C -----------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      COMMON/SYSVAR/N,A(3,3),Y0(3),T0
      COMMON/STATE2/RP(3,3),RL(3,3),RU(3,3)
      COMMON/STATE3/B(3),X(3)
C
C COMPUTE THE B VECTOR
C
      B(1)=B(1)/RL(1,1)
      DO 10 I=2,N
      SUM1=0.d0
      DO 20 K=1,I-1
      SUM1=SUM1+RL(I,K)*B(K)
20    CONTINUE
      B(I)=(B(I)-SUM1)/RL(I,I)
10    CONTINUE
```

```
C
C COMPUTE THE SOLUTION VECTOR
C
      X(N)=B(N)
      DO 30 J=N-1,1,-1
      SUM2=0.d0
      DO 40 K=J+1,N
      SUM2=SUM2+RU(J,K)*X(K)
40    CONTINUE
      X(J)=B(J)-SUM2
30    CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE RJAC
C
C -----------------------------------------------------
C
C     SUBROUTINE RJAC :
C                       SUBROUTINE CONTAINING THE
c     ELEMENTS OF THE JACOBIAN MATRIX J. THE USER IS
c     REQUIRED TO SUPPLY AN ANALYTIC JACOBIAN MATRIX.
C
C -----------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON/SYSVAR/N,A(3,3),Y0(3),T0
      COMMON/STATE4/RJ(3,3)

      DO 10 I=1,N
       DO 20 J=1,N
       RJ(I,J)=A(I,J)
20     CONTINUE
10    CONTINUE
      RETURN
C
      END
```

```
      SUBROUTINE DIFEQ(N,T,A,Y,DY)
C -----------------------------------------------------
C     SUBROUTINE DIFEQ :
C                         THIS SUBROUTINE EVALUATES THE
c     RIGHT HAND SIDE OF THE SYSTEM AY'=F. THE RIGHT
c     HAND SIDE VECTOR IS RETURNED IN THE VECTOR DY.
C
C -----------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(3,3),Y(3),DY(3)

      DO 10 I=1,N
10      DY(I)=0.D0
      DO 20 I=1,N
      DO 30 J=1,N
       DY(I)=DY(I)+A(I,J)*Y(J)
30    CONTINUE
20    CONTINUE
C
      RETURN
      END
```

```
      PROGRAM RKMAIN
C
C ---------------------------------------------------
C
C MAIN PROGRAM FOR SOLVING THE SYSTEM AY'(T) = F(T,Y)IN
C EXAMPLE 2.5. THIS PROGRAM CALLS THE DRIVER ROUTINE
C RK4 WHICH PERFORMS THE INTEGRATION. THE
C INITIALISATION OF THE REQUIRED PARAMETERS AND SETTING
C OF THE INITIAL CONDITIONS IS ALSO DONE HERE.
C
C ---------------------------------------------------
C
C     PARAMETERS ARE :
C     T0      -  CURRENT TIME
C     N       -  NUMBER OF EQUATIONS
C     Y       -  SOLUTION VECTOR
C     DT      - TIME STEP
C     TMAX    - FINAL TIME
C     DTOUT   - INTERVALS FOR PRINTING OF RESULTS
C     NCOL1...- COLUMNS FOR OUTPUT
C
C ---------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /SYSVAR/ T,TMAX,TNEXT,DT
      COMMON /STATE1/ N,Y(200),GN(200)
      DATA N,T0,DT/3,0.D0,0.01D0/
      DATA TMAX,DTOUT/1.D0,0.1D0/
      DATA NCOL1,NCOL2,NCOL3/1,2,3/
C

      DK = 1.D0
      TNEXT = T0+DTOUT
C
      CALL INIT(N,Y)
      CALL DIFEQ2
      WRITE(*,102)
```

```
102   FORMAT(3X,'FOURTH ORDER RUNGE-KUTTA METHOD')
      WRITE(*,*)' DT = ',DT
      WRITE(*,*) NCOL1,NCOL2,NCOL3
    2 FORMAT(/5X,' T',10X,'Y(',I3,')',8X,'Y(',I3,')',
     * 8X,'Y(',I3,')'/)
      WRITE(*,201) T,Y(NCOL1),Y(NCOL2),Y(NCOL3)
201   FORMAT(1PE10.3,3(1X,1PE13.6))
C
10    CONTINUE
      CALL RK4
      IF(T.GE.TNEXT) THEN
      WRITE(*,201) T,Y(NCOL1),Y(NCOL2),Y(NCOL3)
      TNEXT = TNEXT+DTOUT
      ENDIF
      IF(T.LE.TMAX) GOTO 10
C
      END
```

```
         SUBROUTINE RK4
C
C -------------------------------------------------
C
C     SUBROUTINE RK4 :
C                      DRIVER ROUTINE FOR INTEGRATING
C     A SYSTEM AY'(T)=F(T,Y). THE FOURTH ORDER RUNGE-
C     KUTTA METHOD WITH FIXED STEPSIZE IS IMPLEMENTED.
C
C -------------------------------------------------
C
C     PARAMETERS ARE :
C     T    - CURRENT TIME
C     DT   - STEPSIZE
C     N    - NUMBER OF EQUATIONS
C     Y    - SOLUTION VECTOR
C
C -------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION YOLD(200),RK1(200),RK2(200),RK3(200)
      COMMON/SYSVAR/ T,TMAX,TNEXT,DT
      COMMON/STATE1/ N,Y(200),GN(200)
C
      TINIT=T
      SIXTH=1.D0/6.D0
      S=DT/2.D0

C     CALCULATE FIRST DERIVATIVE
C
      CALL  DIFEQ2
      DO 10 I=1,N
C
      YOLD(I)=Y(I)
      RK1(I)=GN(I)*S
      Y(I)=YOLD(I)+RK1(I)
C
```

```
   10 CONTINUE
C
C     CALCULATE SECOND DERIVATIVE
C
      T=T+S
C
      CALL  DIFEQ2
      DO 20 I=1,N
C
      RK2(I)=GN(I)*S
      Y(I)=YOLD(I)+RK2(I)
C
   20 CONTINUE
C     CALCULATE THIRD DERIVATIVE
      CALL  DIFEQ2
      DO 30 I=1,N

      RK3(I)=GN(I)*DT
      Y(I)=YOLD(I)+RK3(I)

   30 CONTINUE
C
C     CALCULATE FOURTH DERIVATIVE
C
      T=T+S
C
      CALL  DIFEQ2
C
C     FINAL ESTIMATE
C
      DO 40 I=1,N

      GN(I)=GN(I)*DT
      Y(I)=SIXTH*(2.D0*(RK1(I)+RK3(I))+4.D0*RK2(I)
     * +GN(I))+YOLD(I)

   40 CONTINUE
```

```
T=TINIT+DT

RETURN
END
```

```
        SUBROUTINE DIFEQ2
C
C -----------------------------------------------------
C
C     SUBROUTINE DIFEQ2 :
C                          CONTAINS THE RIGHT HAND SIDE
C     OF THE SYSTEM AY'(T) = F(T,Y) IN EXAMPLE 2.5.
C
C -----------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /SYSVAR/ T,TMAX,TNEXT,DT
      COMMON /STATE1/N,Y(200),DY(200)
      DIMENSION A(3,3)

      A(1,1) = -21.d0
      A(1,2) = 19.d0
      A(1,3) = -20.d0
      A(2,1) = 19.d0
      A(2,2) = -21.d0
      A(2,3) = 20.d0
      A(3,1) = 40.d0
      A(3,2) = -40.d0
      A(3,3) = -40.d0

      DO 10 I=1,N
10      DY(I)=0.D0
      DO 20 I=1,N
      DO 30 J=1,N
       DY(I)=DY(I)+A(I,J)*Y(J)
30    CONTINUE
20    CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE INIT(N,Y)
C -----------------------------------------------------
C
C     SUBROUTINE INIT :
C                       CONTAINS THE VECTOR OF INITIAL
C     CONDITIONS.
C
C -----------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION Y(N)

      Y(1)=1.d0
      Y(2)=0.d0
      Y(3)=-1.d0
C
      RETURN
      END
```

Appendix C

Portion of Program MGS2

Portion of computer code shown below replaces the main loop in MGS1. The user should insert a dimension statement of the form
DIMENSION FIOLD(20), VOLD(200)
The variables in the code below have the following meanings

```
NORDR1 - NUMBER OF AMPLITUDE EQUATIONS
NORDR2 - NUMBER OF SHAPE EQUATIONS
L      - NUMBER OF TIME STEPS IN AMPLITUDE EQUATIONS
       - FOR ONE STEP IN SHAPE EQUATIONS
S      - ITERATION NUMBER FOR INNER ITERATIONS
FI     - AMPLITUDE FUNCTION
V      - SHAPE VECTOR
YVAL   - SOLUTION VECTOR
```

```
C    ****** BEGINNING OF MAIN LOOP FOR MGS2 ******
C
      DO 11 I=1,NORDR1
11      FIOLD(I)=FI(I)
      C=0
      TOLD=T
      T2OLD=T2
      S=0
```

```
15    CONTINUE
      C=C+1
C
C ---------------------------------------------
C
      CALL RK4
C
C ---------------------------------------------
C
      IF(C.LT.L) GOTO 15
      C=0
       IF(S.EQ.0) THEN
        DO 14 I=1,NORDR1
          FI1(I)=FI(I)
          FI(I)=(FIOLD(I)+FI(I))/2
14      CONTINUE
      DO 12 I=1,NORDR2
12      VOLD(I)=V(I)
       ENDIF
        IF (S.EQ.1) THEN
         DO 16 I=1,NORDR2
16         V(I)=VOLD(I)
         DO 19 I=1,NORDR1
19         FI(I)=(FI1(I)+FI(I))/2
        ENDIF
C
C ---------------------------------------------
C
      CALL RK2
C
C ---------------------------------------------
C
      S=S+1
      IF(S.EQ.1) THEN
        T=TOLD
        T2=T2OLD
        DO 35 I=1,NORDR1
```

```
35         FI(I)=FIOLD(I)
          DO 36 I=1,NORDR2
            V(I)=(VOLD(I)+V(I))/2
36        CONTINUE
         GOTO 15
       ELSE
         S=0
         TOLD=T
         T2OLD=T2
         DO 17 I=1,NORDR1
17         FIOLD(I) = FI(I)
       ENDIF
C
       IF( T.GE.TNEXT)THEN
       DO 140 J=1,M
       DO 140 I=1+(J-1)*K,J*K
140    YVAL(I) = FI(J)*V(I)
       WRITE(*,999)T,YVAL(NCOL1),YVAL(NCOL2),
      *YVAL(NCOL3),YVAL(NCOL4),YVAL(NCOL5)
999    FORMAT(1PE10.3,5(1X,1PE13.6))
       WRITE(2,9991)T,FI(1),FI(2),DT,V(10),
      * V(40),DT2
9991   FORMAT(1PE10.3,2(1X,1PE12.5),1PE8.1,2(1X,1PE12.5),
      *1PE8.1)
       DK=DK+1.D0
       TNEXT=DK*DTOUT
       ENDIF
       IF(T.LT.TMAX)                              GOTO 15
C
C      ********** END OF MAIN LOOP FOR MGS2 **********
```

Appendix D

PDEMGS and User Defined Routines

```
      SUBROUTINE PDEMGS(NPDE,MPTS,T,Y,DY)
C
C  This is DOUBLE PRECISION version !
C -------------------------------------------------
C
C     SUBROUTINE PDEMGS :
C                           PDEMGS IS AN INTERFACE
C     ROUTINE WHICH USES CENTRED DIFFERENCE
C     APPROXIMATIONS TO CONVERT ONE-DIMENSIONAL SYSTEMS
C     OF PARTIAL DIFFERENTIAL INTO A SYSTEM OF ORDINARY
C     DIFFERENTIAL EQUATIONS , DY = F(T,X,Y). THIS
C     ROUTINE IS INTENDED TO BE USED WITH MGS [1].
C
C                         REFERENCES
C                         ----------
C
C     [1] KOZAKIEWICZ, J.M., MIKA, J.R., MULTIGRID
C     METHOD FOR NUMERICAL SOLUTION OF ORDINARY
C     DIFFERENTIAL EQUATIONS, UNIVERSITY OF NATAL, DEPT.
C     OF MATHEMATICS AND APPLIED MATHEMATICS, DURBAN,
C     SOUTH AFRICA, 1991.
C
```

```
C ------------------------------------------------
C
C     THE INPUT PARAMTERS ARE :
C       NPDE = NUMBER OF PARTIAL DIFFERENTIAL EQUATIONS.
C       MPTS = NUMBER OF SPATIAL GRID POINTS.
C          T = CURRENT VALUE OF TIME.
C          Y = AN NPDE*MPTS ARRAY CONTAINING THE
C              COMPUTED SOLUTION AT TIME T.
C
C     THE OUTPUT PARAMETERS ARE :
C         DY = AN NPDE*MPTS ARRAY CONTAINING THE RIGHT
C              HAND SIDE OF THE RESULTING SYSTEM OF ODES,
C              F(T,X,Y), OBTAINED BY DISCRETISING THE
C              GIVEN PDEs.
C
C     THE USER IS RQUIRED TO INSERT DIMENSION STATEMENTS
C     OF THE FOLLOWING FORM :
C       DIMENSION U(**), UX(**), UXX(**), UDOT(**)
C       DIMENSION ALPHA(**), BETA(**), GAMMA(**)
C     THE SYMBOLS ** DENOTE THE ACTUAL VALUE OF THE NPDE
C     FOR THE PROBLEM BEING SOLVED.
C
C     THE COMMON BLOCK MESH CONTAINS THE USER SPECIFIED
C     SPATIAL MESH POINTS AND THE MESHSIZE H.
C
C ------------------------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON/MESH/X(100),H
      DIMENSION Y(200), DY(200)
      DIMENSION U(2),UX(2),UXX(2),UDOT(2)
      DIMENSION ALPHA(2),BETA(2),GAMMA(2)
C
C ------------------------------------------------
C
C  UPDATE THE LEFT BOUNDARY VALUES
C
```

```
      DO 10 K=1,NPDE
        L=(K-1)*MPTS
        U(K)=Y(1+L)
10    CONTINUE
      CALL BNDRY(T,X(1),U,ALPHA,BETA,GAMMA,NPDE)
      KTEST=0
      DO 20 K=1,NPDE
        L=(K-1)*MPTS
        IF (BETA(K).NE.0.D0) GOTO 20
        Y(1+L)=GAMMA(K)/ALPHA(K)
        KTEST=KTEST+1
20    CONTINUE
      IF (KTEST.EQ.NPDE) GOTO 80
C
C  FORM APPROXIMATIONS TO UX AT THE LEFT BOUNDARY
C
      DO 40 K=1,NPDE
        L=(K-1)*MPTS
        IF(BETA(K).NE.0.D0) GOTO 30
        UX(K)=(Y(2+L)-Y(1+L))/2*H
        GOTO 40
30      UX(K)=(GAMMA(K)-ALPHA(K)*Y(1+L))/BETA(K)
40    CONTINUE
C
C  FORM APPROXIMATIONS TO UXX AT THE LEFT BOUNDARY
C
      DO 60 K=1,NPDE
        L=(K-1)*MPTS
        IF (BETA(K).NE.0.D0) GOTO 50
        UXX(K)=(2.D0/H**2)*(Y(2+L)-Y(1+L))
        GOTO 60
50      UXX(K)=(2.D0/H)*((Y(2+L)-Y(1+L))/H -
     *  (GAMMA(K)-ALPHA(K)*Y(1+L))/BETA(K))
60    CONTINUE
C
C  EVALUATE RIGHTHAND SIDE OF PDEs AT THE LEFT BOUNDARY
C
```

```
      CALL F(T,X(1),U,UX,UXX,UDOT,NPDE)
      DO 70 K=1,NPDE
        L=(K-1)*MPTS
        DY(1+L)=UDOT(K)
70    CONTINUE
C
C  SET DY=0 FOR KNOWN LEFT BOUNDARY VALUES
C
80    DO 90 K=1,NPDE
        L=(K-1)*MPTS
        IF(BETA(K).EQ.0.D0) DY(1+L)=0.D0
90    CONTINUE
C
C  UPDATE THE RIGHT BOUNDARY VALUES
C
      DO 100 K=1,NPDE
        L=K*MPTS
        U(K)=Y(L)
100   CONTINUE
      CALL BNDRY(T,X(MPTS),U,ALPHA,BETA,GAMMA,NPDE)
      KTEST=0
      DO 110 K=1,NPDE
        L=K*MPTS
        IF (BETA(K).NE.0.D0) GOTO 110
        Y(L)=GAMMA(K)/ALPHA(K)
        KTEST=KTEST+1
110   CONTINUE
C
C -------------------------------------------------
C
C  MAIN LOOP TO FORM ORDINARY DIFFERENTIAL EQUATIONS AT
C  THE GRID POINTS.
C
      DO 130 I=2,MPTS-1
        DO 120 K=1,NPDE
          L=(K-1)*MPTS
          U(K)=Y(I+L)
```

```
C
C  EVALUATE UX AT THE I-TH GRID POINT
C
          UX(K)=(Y(I+1+L)-Y(I-1+L))/(2.D0*H)
C
C  EVALUATE UXX AT THE I-TH GRID POINT

          UXX(K)=(Y(I+1+L)-2.D0*Y(I+L)+Y(I-1+L))/(H**2)
120     CONTINUE
C
C  EVALUATE RIGHTHAND SIDE OF PDEs AT THE I-TH GRID POINT
C
        CALL F(T,X(I),U,UX,UXX,UDOT,NPDE)
        DO 140 K=1,NPDE
          L=(K-1)*MPTS
          DY(I+L)=UDOT(K)
140     CONTINUE
130   CONTINUE
      IF(KTEST.EQ.NPDE) GOTO 210
C
C -------------------------------------------------
C
C  FINISH UPDATING RIGHT BOUNDARY IF NECESSARY
C
      DO 150 K=1,NPDE
        L=K*MPTS
        U(K)=Y(L)
150   CONTINUE
      CALL BNDRY(T,X(MPTS),U,ALPHA,BETA,GAMMA,NPDE)
C
C  FORM APPROXIMATIONS TO UX AT RIGHT BOUNDARY
C
      DO 170 K=1,NPDE
        L=K*MPTS
        IF(BETA(K).NE.0.D0) GOTO 160
        UX(K)=(Y(L)-Y(L-1))/2*H
        GOTO 170
```

```
160      UX(K)=(GAMMA(K)-ALPHA(K)*Y(L))/BETA(K)
170    CONTINUE
C
C  FORM APPROXIMATIONS TO UXX AT RIGHT BOUNDARY
C
       DO 190 K=1,NPDE
         L=K*MPTS
         IF(BETA(K).NE.0.D0) GOTO 180
         UXX(K)=(2/H**2)*(Y(L-1)-Y(L))
         GOTO 190
180      UXX(K)=(2.D0/H)*((Y(L-1)-Y(L))/H+
     *   (GAMMA(K)-ALPHA(K)*Y(L))/BETA(K))
190    CONTINUE
C
C  EVALUATE UXX AT RIGHT BOUNDARY
C
       CALL F(T,X(MPTS),U,UX,UXX,UDOT,NPDE)
       DO 200 K=1,NPDE
       L=K*MPTS
       DY(L)=UDOT(K)
200    CONTINUE
C
C      SET DY =0 FOR KNOWN BOUNDARY VALUES
C
210    DO 220 K=1,NPDE
         L=K*MPTS
         IF(BETA(K).EQ.0.D0) DY(L)=0.D0
220    CONTINUE
C
C -------------------------------------------------------
       RETURN
       END
```

```
      SUBROUTINE F(T,X,U,UX,UXX,FVAL,NPDE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION U(NPDE),UX(NPDE),UXX(NPDE),FVAL(NPDE)

C F ROUTINE FOR EXAMPLE 5.3.1 (A)
C
      FVAL(1)=0.5D0*(UXX(1)+U(1)-(4.D0*X+2.D0)*EXP(X+T))
      RETURN
      END


      SUBROUTINE BNDRY(T,X,U,ALPHA,BETA,GAMMA,NPDE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION U(NPDE),ALPHA(NPDE),BETA(NPDE),GAMMA(NPDE)
C
C  BNDRY ROUTINE FOR EXAMPLE 5.3.1 (A)
C
      ALPHA(1)=1.D0
      BETA(1)=0.D0
      GAMMA(1)=0.D0
      RETURN
      END
```

```
      SUBROUTINE F(T,X,U,UX,UXX,FVAL,NPDE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION U(NPDE),UX(NPDE),UXX(NPDE),FVAL(NPDE)

C F ROUTINE FOR EXAMPLE 5.3.1 (B)
C
      FVAL(1)=0.003D0*UXX(1)-U(1)*UX(1)
      RETURN
      END


      SUBROUTINE BNDRY(T,X,U,ALPHA,BETA,GAMMA,NPDE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION U(NPDE),ALPHA(NPDE),BETA(NPDE),GAMMA(NPDE)
C
C BNDRY ROUTINE FOR EXAMPLE 5.3.1 (B)
C
      ALPHA(1)=1.D0
      BETA(1)=0.D0
C
C EXAC IS A FUNCTION ROUTINE FOR COMPUTING THE EXACT
C SOLUTION VALUES
C
      GAMMA(1)=EXAC(X,T)
      RETURN
      END
```

```
      SUBROUTINE F(T,X,U,UX,UXX,FVAL,NPDE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION U(NPDE),UX(NPDE),UXX(NPDE),FVAL(NPDE)
C F ROUTINE FOR EXAMPLE 5.3.1 (C)
C
      FVAL(1)=UX(1)*UX(2)+(U(2)-1.D0)*UXX(1)
      FVAL(1)=FVAL(1)+(16*X*T-2*T-16*(U(2)-1))*(U(1)-1)
      FVAL(1)=FVAL(1)+10*X*EXP(-4*X)
      FVAL(2)=UXX(2)+UX(1)+4*U(1)-4.D0+X**2-2*T-10*T*EXP(-4*X)
      RETURN
      END



      SUBROUTINE BNDRY(T,X,U,ALPHA,BETA,GAMMA,NPDE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION U(NPDE),ALPHA(NPDE),BETA(NPDE),GAMMA(NPDE)
C
C     BNDRY ROUTINE FOR EXAMPLE 5.3.1 (C)
C
      IF(X.EQ.0.D0)THEN
      ALPHA(1)=1.D0
      BETA(1)=0.D0
      GAMMA(1)=1.D0
      ALPHA(2)=1.D0
      BETA(2)=0.D0
      GAMMA(2)=1.D0
      ENDIF
      IF(X.EQ.1.D0)THEN
      ALPHA(1)=3.D0
      BETA(1)=1.D0
      GAMMA(1)=3.D0
      ALPHA(2)=-EXP(U(1))
      BETA(2)=5.D0
      GAMMA(2)=-EXP(U(1))
      ENDIF
```

```
RETURN
END
```

```
      SUBROUTINE F(T,X,U,UX,UXX,FVAL,NPDE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION U(NPDE),UX(NPDE),UXX(NPDE),FVAL(NPDE)
C
C F ROUTINE FOR EXAMPLE 5.3.1 (D)
C
      FVAL(1)=(U(2)**2)*UXX(1)+2*U(2)*UX(1)*UX(2)-U(1)*U(2)
      FVAL(1)=FVAL(1)-(U(1)**2)+10.D0
      FVAL(2)=(U(1)**2)*UXX(2)+2*U(1)*UX(1)*UX(2)+UXX(1)
      FVAL(2)=FVAL(2)+U(1)*U(2)-U(2)**2
      RETURN
      END



      SUBROUTINE BNDRY(T,X,U,ALPHA,BETA,GAMMA,NPDE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION U(NPDE),ALPHA(NPDE),BETA(NPDE),GAMMA(NPDE)
C
C BNDRY ROUTINE FOR EXAMPLE 5.3.1 (D)
C
      PI=4.D0*ATAN(1.D0)
C
C DEFINE ALPHA(K),BETA(K),GAMMA(K) AT THE LEFT BOUNDARY
C
      IF(X.EQ.0.D0) THEN
      ALPHA(1)=1.D0
      BETA(1)=0.D0
      GAMMA(1)=0.5D0
      ALPHA(2)=1.D0
      BETA(2)=0.D0
      GAMMA(2)=PI
      ENDIF
C
C DEFINE ALPHA(K),BETA(K),GAMMA(K) AT THE RIGHT BOUNDARY
C
      IF(X.EQ.1.D0) THEN
```

```
ALPHA(1)=0.D0
BETA(1)=1.D0
GAMMA(1)=0.5D0-DSIN(U(1)*U(2))
ALPHA(2)=0.D0
BETA(2)=1.D0
GAMMA(2)=1.D0+DCOS(U(1)*U(2))
ENDIF
RETURN
END
```

```
      SUBROUTINE F(T,X,U,UX,UXX,FVAL,NPDE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION U(NPDE),UX(NPDE),UXX(NPDE),FVAL(NPDE)
C
C F ROUTINE FOR CHEMICAL KINETICS PROBLEM
C
      D=1.D-4
      A1=4.D5
      A2=272.443800016D0
      A3=1.D-4
      A4=0.007D0
      A5=3.67D-16
      A6=4.13D-12
      B1=272.4438D0
      B2=1.00016D-4
      B3=3.67D-15
      B4=3.57D-15
      C1=1.6D-8
      C2=0.007D0
      C3=4.1283D-12
      C4=3.57D-15
      D1=7.000016D-3
      D2=3.57D-15
      D3=4.128D-12
      IF (X.GE.0.475D0.AND.X.LE.0.575D0) THEN
      S=3250.D0
      ELSE
      S=360.D0
      ENDIF

      FVAL(1)=D*UXX(1)+A1-A2*U(1)+A3*U(2)+A4*U(4)-
     * A5*U(1)*U(2)-A6*U(1)*U(4)
      FVAL(2)=D*UXX(2)+B1*U(1)-B2*U(2)+B3*U(1)*U(2)-
     *B4*U(2)*U(3)
      FVAL(3)=D*UXX(3)-C1*U(3)+C2*U(4)+C3*U(1)*U(4)-
     *C4*U(2)*U(3)+800.D0+S
      FVAL(4)=D*UXX(4)-D1*U(4)+D2*U(2)*U(3)-D3*U(1)*U(4)
```

```
*+800.D0

 RETURN
 END



 SUBROUTINE BNDRY(T,X,U,ALPHA,BETA,GAMMA,NPDE)
 IMPLICIT DOUBLE PRECISION(A-H,O-Z)
 DIMENSION U(NPDE),ALPHA(NPDE),BETA(NPDE),GAMMA(NPDE)

 ALPHA(1)=0.D0
 BETA(1)=1.D0
 GAMMA(1)=0.D0
 ALPHA(2)=0.D0
 BETA(2)=1.D0
 GAMMA(2)=0.D0
 ALPHA(3)=0.D0
 BETA(3)=1.D0
 GAMMA(3)=0.D0
 ALPHA(4)=0.D0
 BETA(4)=1.D0
 GAMMA(4)=0.D0
 RETURN
 END
```