AMPLITUDE-SHAPE METHOD

FOR THE NUMERICAL SOLUTION

OF ORDINARY DIFFERENTIAL

EQUATIONS

by

Nabendra Parumasur

Submitted in partial fulfilment of the

requirements for the degree of

Doctor of Philosophy,

in the

Department of Mathematics and Applied Mathematics,

University of Natal

Durban

1997

Preface

The theoretical work described in this thesis was carried out in the Department of Mathematics and Applied Mathematics, University of Natal, Durban, from January 1995 to December 1997, under the supervision of Professor Janusz R Mika and Dr Jacek Banasiak.

The 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.

Acknowledgements

I would like to express my gratitude to Professor J R Mika and Dr J Banasiak, under whose supervision this study was completed. It has been a pleasure to be guided and advised by such expert and willing mentors.

I would also like to extend my thanks to :

- The Foundation of Research and Development (FRD) of South Africa for financial assistance received,
- The Department of Mathematics and Applied Mathematics, University of Durban-Westville for the use of computer facilities,
- Miss Sanjana Brijball of the Department of Industrial Psychology, University of Durban-Westville for her unfailing support and encouragement,
- My family for their patience and numerous sacrifices made.

Abstract

In this work, we present an amplitude-shape method for solving evolution problems described by partial differential equations. The method is capable of recognizing the special structure of many evolution problems. In particular, the stiff system of ordinary differential equations resulting from the semi-discretization of partial differential equations is considered. The method involves transforming the system so that only a few equations are stiff and the majority of the equations remain non-stiff. The system is treated with a mixed explicit-implicit scheme with a built-in error control mechanism. This approach proved to be very effective for the solution of stiff systems of equations describing spatially dependent chemical kinetics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The mathematical description of physical phenomena occuring in practical applications is usually expressed in terms of ordinary differential equations (ODEs). The formulation of solution techniques for such equations has been a focus of much interest in the past and still remains a very active area of research. In the literature, for instance, a vast number of papers have been devoted to the solution of the initial value problem (IVP) for the system of first-order ODEs

$$\begin{aligned} \frac{dx}{dt} &= f(x,t) \\ x(0) &= x_0. \end{aligned} \tag{1.1}$$

However, in many realistic practical applications the system (1.1) is so complicated as to preclude an analytic solution, and one has to resort to numerical methods for solving (1.1). On the other hand, the numerical solution of (1.1) presents a formidable challenge when the system of ODEs is *stiff* (see Sec. 2.6).

Our discussion thus far is concerned with the system of ODEs (1.1) which describes the evolution of a physical system with time. However, in many practical applications the

evolution of the system depends on both time and space. For example, if we are interested in the pollution effects caused by chemical reactions in the atmosphere, then the evolution of the system would depend on how the level of the pollutant changes with time, and the spatial distribution of the pollutant. A proper description of such a physical system would therefore, require the use of partial differential equations (PDEs) to describe its evolution.

For the numerical approach, PDEs have to be first converted into an algebraic form which is suitable for computation. The *method of lines* (MOL) (see Sec. 2.5) provides a way for doing this. Basically, the MOL is a numerical technique for converting a PDE into a system of ODEs via a discretization in the space-like independent variables. With this technique some classes of PDEs lead directly to systems of ODEs of the form (1.1), the manifestations of which imply that the recent advances in the numerical solution of (1.1) can be applied to the solution of PDEs. However, special attention needs to be given to the fact that the MOL does not recognize the physical structure of the PDEs, and in most cases the resulting system of ODEs is stiff.

The primary objective of this study is to establish the amplitude shape-method (ASM) as a viable approach for the numerical solution of (1.1) and to investigate the feasibility of applying the method to stiff systems of ODEs. The ASM involves transforming the system (1.1) so that only a few equations remain stiff, the majority of the equations are nonstiff. As a result, a significant reduction in the computational effort may be possible when the method is applied to large stiff systems of ODEs. A particularly attractive feature of the method is its ability to recognize the special structure of many systems of ODEs resulting from the MOL. In particular, the method may be useful for the solution of large stiff systems of ODEs resulting from PDEs describing the evolution of physical systems. In this study it will become evident that the ASM is capable of exploiting the physical structure of such systems of ODEs and proves to be extremely advantageous.

In chapter 2 we discuss the numerical solution of ODEs. We focus on the class of one-step

methods and in particular, consider the popular class of Runge-Kutta methods. This is followed by a discussion on the role of extrapolation in estimating the error and changing the stepsize. Thereafter, the MOL is introduced and the theory on stiff ODEs is reviewed. We conclude chapter 2 by including a class of chemical problems which is chosen as an application. Chapter 3 concerns the presentation of the amplitude-shape method. The amplitude-shape equations are derived and a corresponding amplitude-shape system obtained. We then establish the equivalence of the amplitude-shape system and the original system of ODEs. In chapter 4 we consider the numerical solution of the amplitude-shape equations. Two numerical schemes are presented for this purpose and the proofs of consistency and convergence are given for one of them. The programming strategy is described and a set of algorithms is provided. Chapter 5 is devoted to the numerical experimentation, which is intended to demonstrate the effectiveness and robustness of the ASM. The numerical results for a variety of non-trivial examples are reported. Finally, chapter 6 concludes this study with a general discussion of the material presented in this thesis.

Chapter 2

Numerical Solution of Ordinary Differential Equations

2.1 Introduction

We consider the IVP for a system[1] of n first order ODEs in the form

$$\begin{aligned} \frac{dx}{dt} &= f(x,t), \\ x(0) &= q, \qquad t \geq 0, \end{aligned} \tag{2.1}$$

where

[1]Throughout this study we consider systems of ODEs, which require the use of vectors. Any attempt to adopt the standard convention of typefacing vectors in bold font, would only blemish the appearance of the text. Instead, we write systems such as (2.1), together with (2.2) to denote that t is a scalar and x and f are n-dimensional vectors. Occasionally, we shall also use the notation $t \in \mathbf{R}$ to denote that t is a scalar and $x \in \mathbf{R}^n$ to denote that x is an n-dimensional vector. In all other instances, it will be clear from the context whether a particular symbol represents a scalar or a vector.

$$\begin{aligned} x : \mathbf{R} &\longrightarrow \mathbf{R}^n, \\ f : \mathbf{R}^{n+1} &\longrightarrow \mathbf{R}^n, \qquad n \geq 1, \end{aligned} \tag{2.2}$$

and $q \in \mathbf{R}^n$ is the given initial vector. We choose

$$\|x\| = \max\{|x_1|, ..., |x_n|\}$$

as the norm in $\mathbf{R}^n$.

Firstly, we establish the conditions under which the system (2.1) has a unique solution. These conditions are related to the following well known theorem, of the theory of ODEs, which is stated here without proof (see [8]).

Theorem 2.1.1 *Let* $T_\alpha = [0, \alpha]$, $\alpha > 0$,

$$D = \{x \in \mathbf{R}^n : \|x - q\| \leq d, \;\; d > 0\}, \tag{2.3}$$

and $E_\alpha = D \times T_\alpha$. *If*

(i) f *is defined and continuous on* E_α, *and*

(ii) f *satisfies a Lipschitz condition with respect to* x *on* E_α *with a constant* L

$$\|f(x_1, t) - f(x_2, t)\| \leq L \|x_1 - x_2\|, \quad \forall (x_1, t), (x_2, t) \in E_\alpha,$$

then (2.1) has a unique solution on $E_a = D \times T_a$ *with* $T_a = [0, a]$ *and where* $a = \min\left\{\alpha, \frac{d}{M}\right\}$ *and*

$$M = \sup_{(x,t) \in E_\alpha} \|f(x, t)\|.$$

We will be concerned only with IVPs which satisfy conditions (i) and (ii) in Theorem 2.1.1. For this we state the following definition.

Definition 2.1.1 *The IVP (2.1) is said to be* **well posed** *(abbreviated WPIVP) if it possesses a unique solution for all* $q \in \mathbf{R}^n$.

Next we give a stability property of the IVP (2.1). For this we consider the perturbed IVP of (2.1)

$$\begin{aligned} \frac{dy}{dt} &= f(y,t) + \eta(t) \\ y(0) &= q + \delta, \end{aligned} \tag{2.4}$$

where $\eta(t)$ and δ are perturbations of f and q, respectively. We denote by $(\eta(t), \delta)$ such a perturbation of (2.1). Lambert

Definition 2.1.2 *(Lambert [13]). The IVP (2.1) is said to be* **totally stable** *(abbreviated TSIVP) if there exists a constant* $S \geq 0$ *such that for all* $t \in T_a$

$$\|y(t) - y^*(t)\| \leq S\epsilon,$$

whenever

$$\|\eta(t) - \eta^*(t)\| \leq \epsilon \quad \textit{and} \quad \|\delta - \delta^*\| \leq \epsilon,$$

where $y(t)$ *and* $y^*(t)$ *are the solutions of (2.4) corresponding to the perturbations* $(\eta(t), \delta)$ *and* $(\eta^*(t), \delta^*)$ *of (2.1).*

The above definition ensures that the solution of (2.1) continuously depends on the perturbations of f and q. The importance of such a requirement is that any numerical method applied to (2.1) will give rise to numerical errors by way of discretization and round-off, which is equivalent to perturbing (2.1). If (2.1) is not totally stable then no numerical method would be able to produce an acceptable solution. Gear [6] shows that a WPIVP is totally stable, that is

$$WPIVP \Rightarrow TSIVP.$$

Typical numerical methods for solving (2.1) are methods which begin at x_0 and compute successive approximations $x_1, x_2, ..., x_n, ...$, to the exact solution, $x(t)$, of (2.1) at the discrete values $t_1, t_2, ..., t_n, ...$ of t. These methods are called *discrete* methods and usually fall into one of two main categories, namely, *explicit* or *implicit* methods. Explicit methods are easier and cheaper to implement because of their simple arithmetic and modest storage requirements, but they are subject to numerical instabilities, and they might be completely ineffective in case of stiff systems of ODEs (see Section 2.6). On the other hand, implicit methods have better stability properties and are capable of handling stiff ODEs. In the next section we review some results of importance from the theory of one-step methods.

2.2 One-Step Methods

One-step methods belong to the simplest category of discrete methods but at the same time they are very effective and widely used in practical applications. When applied to the system (2.1) they have the form

$$x_{n+1} = x_n + h\phi_f(x_{n+1}, x_n, t_n, h), \tag{2.5}$$

where the subscript f indicates that the dependence of ϕ on x_{n+1}, x_n, t_n is through the function f. The points $t_n \in \mathcal{T}_h$, where the set $\mathcal{T}_h$ is defined as

$$\mathcal{T}_h = \{t_n : t_n = nh,\ n = 0, 1, ..., N,\ t_N = a\}, \tag{2.6}$$

and h is the size of a single step taken in advancing the solution from t_n to t_{n+1}. For convenience we consider a fixed time step. We impose the following conditions on ϕ_f in (2.5),

$$\begin{gathered} \phi_{f\equiv 0}(x_{n+1}, x_n, t_n, h) \equiv 0, \\ \left\|\phi_f(x_{n+1}, x_n, t_n, h) - \phi_f\left(x^*_{n+1}, x^*_n, t_n, h\right)\right\| \leq M\left(\left\|x_{n+1} - x^*_{n+1}\right\| + \left\|x_n - x^*_n\right\|\right), \end{gathered} \tag{2.7}$$

where M is a constant. If ϕ_f is independent of x_{n+1}, then the method (2.5) is said to be *explicit*, otherwise it is *implicit*. The simplest examples of one-step methods are the explicit and implicit Euler methods. The *explicit Euler method* is defined by

$$\phi_f(x_{n+1}, x_n, t_n, h) = f(x_n, t_n),$$

that is

$$x_{n+1} = x_n + hf(x_n, t_n), \tag{2.8}$$

and the *implicit Euler method* is defined by

$$\phi_f(x_{n+1}, x_n, t_n, h) = f(x_{n+1}, t_{n+1}),$$

that is

$$x_{n+1} = x_n + hf(x_{n+1}, t_{n+1}). \tag{2.9}$$

By far the most popular class of one-step methods are the *Runge-Kutta* methods obtained with

$$\phi_f(x_{n+1}, x_n, t_n, h) = \sum_{i=1}^{s} b_i k_i,$$

$$k_i = f\left(x_n + h\sum_{j=1}^{s} a_{ij}k_j, t_n + c_i h\right),$$

where a_{ij}, b_i and c_i are constants. We discuss this class of one-step methods in the next section.

Before stating some basic definitions we introduce the following notations.

Firstly, we use the notation

$$F(h) = O(h^p), \tag{2.10}$$

to imply the existence of finite constants $C \geq 0$ and $h_0 > 0$ such that

$$\|F(h)\| \leq Ch^p, \tag{2.11}$$

for all $h \leq h_0$.

Secondly, we denote by

$$\lim_{\substack{h\to 0 \\ t=nh}} F(h,n), \tag{2.12}$$

the limit in which $h \to 0$ and $n \to \infty$ simultaneously in such a way that $t = nh$ remains fixed. In other words, the limit is evaluated by setting $n = t/h$ in $F(h,n)$ before letting $h \to 0$.

Definition 2.2.1 *The method defined by (2.5) is said to be* **convergent** *if when applied to a WPIVP*

$$\lim_{\substack{h\to 0 \\ t=nh}} x_n = x(t), \tag{2.13}$$

holds for all $t \in T_a$ and all q.

Definition 2.2.2 *The* **local truncation error** *of the method (2.5) at t_{n+1}, denoted by l_{n+1}, is defined by*

$$l_{n+1} = x(t_{n+1}) - x(t_n) - h\phi_f(x(t_{n+1}), x(t_n), t_n, h), \tag{2.14}$$

where $x(t)$ is the exact solution of (2.1).

Definition 2.2.3 *The* **global error** *of the method (2.5) at t_{n+1}, denoted by e_{n+1}, is defined by*

$$e_{n+1} = x_{n+1} - x(t_{n+1}). \tag{2.15}$$

Definition 2.2.4 *The method (2.5) is said to be of* **order** *p if p is the largest integer such that*

$$l_{n+1} = O\left(h^{p+1}\right). \tag{2.16}$$

Definition 2.2.5 *The method (2.5) is said to be* **consistent** *if, when applied to a WPIVP, gives*

$$\lim_{\substack{h\to 0 \\ t=nh}} \frac{l_{n+1}}{h} = 0. \tag{2.17}$$

We establish the necessary and sufficient condition for (2.5) to be consistent.

For the WPIVP (2.1), comprized of m ODEs, we know that $x'(t)$ is continuous. So by the Mean Value theorem

$$x_i(t_{n+1}) - x_i(t_n) = hx_i'(\xi_i), \quad \xi_i \in (t_n, t_{n+1}), \quad i = 1, 2, ..., m.$$

It is convenient to express this in a vector form by letting

$$x'(\xi) := [x_1'(\xi_1), x_2'(\xi_2), ..., x_m'(\xi_m)]^T,$$

denote the vector with components x_i', $i = 1, 2, ..., m$ each of which is evaluated at a particular mean value $\xi_i \in (t_n, t_{n+1})$, $i = 1, 2, ..., m$. That is

$$x(t_{n+1}) - x(t_n) = hx'(\xi).$$

It then follows from (2.14) that

$$\frac{1}{h} l_{n+1} = x'(\xi) - \phi_f(x(t_{n+1}), x(t_n), t_n, h).$$

Now as $h \to 0$, $t = nh$ fixed ($t_n \to t$), by (2.7)

$$x'(\xi) \to x'(t)$$

$$\phi_f(x(t_{n+1}), x(t_n), t_n, h) \to \phi_f(x(t), x(t), t, 0).$$

But $x'(t) = f(x(t), t)$ so the condition for the method (2.5) to be consistent is

$$\phi_f(x(t), x(t), t, 0) = f(x(t), t). \tag{2.18}$$

This is the sufficient condition.

Assume that the iterations x_n produced by (2.5) tend to some function $z(t) \in C^1[0,a]$. Then as $h \to 0$, $t = nh$ fixed,

$$\frac{x_{n+1} - x_n}{h} \to z'(t),$$

$$\phi_f(x_{n+1}, x_n, t_n, h) \to \phi_f(z(t), z(t), t, 0).$$

It then follows that if $z(t)$ is the solution of (2.1), then (2.18) is satisfied. Thus (2.18) is sufficient and necessary.

In the previous section we considered a stability property of (2.1) by perturbing the function f and the initial value q. We now consider a stability property of the discretization method (2.5) by perturbing the functions ϕ_f and the starting value q, in (2.5). For this we consider the perturbed difference system

$$\begin{aligned} z_{n+1} &= z_n + h[\phi_f(z_{n+1}, z_n, t_n, h) + \delta_n], \qquad n = 1, 2, ..., N \\ z_0 &= q + \delta_0 \end{aligned} \tag{2.19}$$

with perturbations $\{\delta_n,\ n = 0, 1, ..., N\}$ and perturbed solutions $\{z_n,\ n = 0, 1, ..., N\}$ of (2.5).

Definition 2.2.6 *(Lambert [13]). Let $\{\delta_n,\ n = 0, 1, ..., N\}$ and $\{\delta_n^*,\ n = 0, 1, ..., N\}$ be two perturbations of (2.5) and let $\{z_n,\ n = 0, 1, ..., N\}$ and $\{z_n^*,\ n = 0, 1, ..., N\}$ be the corresponding perturbed solutions of (2.5). If $\exists\ S \geq 0$ and $h_0 > 0$, such that, $\forall h \in (0, h_0]$*

$$\|\delta_n - \delta_n^*\| \leq \epsilon \Rightarrow \|z_n - z_n^*\| \leq S\epsilon, \quad 0 \leq n \leq N,$$

then (2.5) is said to be **zero stable**.

Zero-stability is a property of the discretization method. This property ensures that the method is not over-sensitive to perturbations. This is an important requirement as the

following observation shows. Computers store numbers with finite precision. In this way round-off errors result when the functions ϕ_f are computed and the starting value q is stored on a computer. Perturbing the difference system has a similar effect. Thus, requiring that the difference system be zero-stable ensures that the method is not over-sensitive to round-off error.

Finally we have the following result (see [13]):

Theorem 2.2.1 *A consistent one step method is zero stable.*

In the next section we discuss a popular class of one-step methods called Runge-Kutta methods.

2.3 Runge-Kutta Methods

The general *s-stage Runge-Kutta* (RK) method for (2.1) can be written in the form

$$x_{n+1} = x_n + h\sum_{i=1}^{s} b_i k_i, \tag{2.20}$$

$$k_i = f\left(x_n + h\sum_{j=1}^{s} a_{ij}k_j, t_n + c_i h\right), \qquad 1 \leq i \leq s, \tag{2.21}$$

where a_i,b_i and c_i are real parameters which define the method. It is convenient to write these parameters in the form of an array

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array},$$

called the *Butcher array* (see [13]). This array contains the vectors

$$c = [c_1, c_2, ..., c_s]^T,$$

$$b = [b_1, b_2, ..., b_s]^T,$$

and the matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \ddots & a_{1s} \\ a_{21} & a_{22} & \ddots & a_{2s} \\ \vdots & & & \vdots \\ a_{s1} & a_{s2} & \ddots & a_{ss} \end{bmatrix}.$$

If $a_{ij} = 0$ for $j \geq i$, $i = 1, 2, ..., s$ the method is *explicit*, otherwise, the method is *implicit*. Thus an s-stage RK method is completely defined by specifying its Butcher array. For example, the only 1-stage explicit RK method is the explicit Euler method (2.8) defined by

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

and the array

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

defines the well known 2-stage explicit RK method

$$\begin{aligned} x_{n+1} &= x_n + \frac{1}{2}h\left(k_1 + k_2\right) \\ k_1 &= f\left(x_n, t_n\right) \\ k_2 &= f\left(x_n + hk_1, t_n + h\right). \end{aligned} \tag{2.22}$$

On the other hand, an example of a 1-stage implicit RK method is defined by

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

This may be written as

$$\begin{aligned} x_{n+1} &= x_n + hk_1 \\ &= x_n + hf\left(x_n + hk_1, t_n + h\right) \\ &= x_n + hf\left(x_n + \left(x_{n+1} - x_n\right), t_{n+1}\right) \\ &= x_n + hf\left(x_{n+1}, t_{n+1}\right) \end{aligned}$$

and is seen to be the same as the implicit Euler method (2.9). Similarly, the 2-stage implicit RK method defined by

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

can be written as

$$x_{n+1} = x_n + \frac{1}{2}h\left(f\left(x_n, t_n\right) + f\left(x_{n+1}, t_{n+1}\right)\right) \tag{2.23}$$

which is the *implicit trapezoidal rule.*

2.4 Extrapolation

We begin with the application of *Richardson's extrapolation* to the system of ODEs (2.1). Before proceeding we note that since the approximations produced by (2.5) also depend on the stepsize h we let $x\left(t,h\right)$ denote the approximation, at t, obtained by a one-step method of the form (2.5), with stepsize $h > 0$. Then, if (2.5) defines a convergent method of order p, $p \leq N$, the approximation $x\left(t,h\right)$ possesses an asymptotic expansion of the form

$$x\left(t,h\right) = x\left(t\right) + \sum_{j=p}^{N} \gamma_j\left(t\right) h^j + E_{N+1}\left(t,h\right) h^{N+1}, \tag{2.24}$$

$$\gamma_k\left(t_0\right) = 0, \qquad k = p, p+1, ..., \tag{2.25}$$

where the functions $\gamma_i\left(t\right)$ are independent of h and the remainder term $E_{N+1}\left(t,h\right)$ is bounded for fixed t and $h > 0$. The validity of such expansions depends on f being sufficiently differentiable and can be found in [7].

We use (2.24) to estimate the global error as follows. Suppose we apply a method of order p which permits an asymptotic expansion of the form (2.24) to (2.1) and obtain, with stepsize $h > 0$, an approximation $x\left(t,h\right)$ at t. Then the global error, denoted by $e\left(t,h\right)$, has the

form

$$e(t,h) := x(t,h) - x(t) = h^p \gamma_p(t) + O\left(h^{p+1}\right). \tag{2.26}$$

Similarly, we obtain another approximation $x\left(t, \frac{h}{2}\right)$ at t with stepsize $\frac{h}{2}$ so that

$$e\left(t, \frac{h}{2}\right) := x\left(t, \frac{h}{2}\right) - x(t) = \left(\frac{h}{2}\right)^p \gamma_p(t) + O\left(h^{p+1}\right). \tag{2.27}$$

Now for sufficiently small h and $\gamma_p(t) \neq 0$ subtracting (2.27) from (2.26) gives

$$x(t,h) - x\left(t, \frac{h}{2}\right) \cong \gamma_p(t)\left(\frac{h}{2}\right)^p (2^p - 1),$$

that is

$$\gamma_p(t)\left(\frac{h}{2}\right)^p \cong \frac{x(t,h) - x\left(t, \frac{h}{2}\right)}{2^p - 1}.$$

Substituting this into (2.27) gives, for sufficiently small h, the following estimate for the global error:

$$e\left(t, \frac{h}{2}\right) \cong \frac{x(t,h) - x\left(t, \frac{h}{2}\right)}{2^p - 1}. \tag{2.28}$$

Moreover, we see that if we substitute (2.28) into (2.27), we have

$$x(t,h) = x\left(t, \frac{h}{2}\right) + \frac{E}{2^p - 1} + O\left(h^{p+1}\right), \tag{2.29}$$

where

$$E = x\left(t, \frac{h}{2}\right) - x(t,h), \tag{2.30}$$

gives the estimate of the local truncation error. Equation (2.29) provides an improved approximation of the numerical solution.

The procedure outlined above is the basic idea of Richardson's extrapolation. More advanced extrapolation schemes based on polynomial extrapolation and rational function extrapolation are also possible. We describe the process of polynomial extrapolation below since this is preferred to rational function extrapolation in practice.

Error and stepsize control is achieved by requiring that the criterion

$$\|e(t,h)\| < \epsilon, \tag{2.31}$$

is adhered to for a prescribed tolerance ϵ. In practice at $t = t_0$ we need to take an optimal stepsize, say h_1, such that (2.31) is satisfied. However, such a stepsize is not known a priori and the best one can do is to obtain an estimate of it as follows. For a method of order p the error can be estimated according to (2.26) by

$$e(t,h) \cong \gamma_p(t)\, h^p. \tag{2.32}$$

By expanding $\gamma_p(t)$ about t_0 and by assuming no initial error (i.e. $\gamma_p(t_0) = 0$ from (2.25)) we have as a first order approximation

$$e(t,h) \cong \gamma_p'(t_0)\,(t - t_0)\, h^p. \tag{2.33}$$

For the stepsize h_1 we therefore require

$$\|e(t_0 + h_1, h_1)\| < \epsilon,$$

and by (2.33) this becomes

$$\left\|\gamma_p'(t_0)\right\| h_1^{p+1} < \epsilon. \tag{2.34}$$

Solving for h_1 gives

$$h_1 < \left(\frac{\epsilon}{\left\|\gamma_p'(t_0)\right\|}\right)^{\frac{1}{p+1}}. \tag{2.35}$$

From this it is seen that to determine h_1 we need the value of $\gamma_p'(t_0)$. In practice, we can approximate $\gamma_p'(t_0)$ by Richardson's extrapolation as follows. Choose a stepsize, say h_0, such that (2.31) is satisfied and obtain from (2.28)

$$e\left(t_0 + h_0, \frac{h_0}{2}\right) \cong \frac{E_0}{2^p - 1},$$

where $E_0 \equiv x(t_0 + h_0, h_0) - x\left(t_0 + h_0, \frac{h_0}{2}\right)$. Then, as a consequence of (2.33), we have

$$\gamma_p'(t_0)\, h_0 \left(\frac{h_0}{2}\right)^p \cong \frac{E_0}{2^p - 1},$$

that is

$$\gamma_p'(t_0) \cong \left(\frac{2^p}{2^p - 1}\right)\left(\frac{E_0}{h_0^{p+1}}\right).$$

By substituting this value for $\gamma_p'(t_0)$ into (2.35), we see that h_1 has to be chosen such that

$$h_1 < \left(\frac{\epsilon}{E_0}\right)^{\frac{1}{p+1}} h_0. \tag{2.36}$$

Once an appropriate value of h_1 corresponding to the prescribed tolerance ϵ is obtained the above process is repeated at $t = t + h_1$. In general, the stepsize for the next integration step is estimated by

$$h_{next} \cong \left(\frac{\epsilon}{E}\right)^{\frac{1}{p+1}} h_{previous}. \tag{2.37}$$

Polynomial extrapolation can be used to approximate $x(t,0) = x(t)$ in the following way:

Let $P_m^{(i)}(h)$ denote the polynomial functions

$$P_m^{(i)}(h) = a_1^{(i)} + a_2^{(i)}h + a_3^{(i)}h^3 + ... + a_{m+1}^{(i)}h^m,$$

with the requirement that

$$P_m^{(i)}(h_j) = x(t,h_j), \qquad j = i, i+1, ..., i+m+1$$

where $\{h_j\}$ is a strictly decreasing sequence of stepsizes. Then the extrapolated values, denoted by $T_{i,m}$

$$T_{i,m} := P_m^{(i)}(0) \approx x(t,0),$$

can be generated recursively using the formula (see [3, 4])

$$T_{i,1} = x(t,h_i) \qquad i = 1,2,... \tag{2.38}$$

$$T_{i,j} = T_{i,j-1} + \frac{T_{i,j-1} - T_{i-1,j-1}}{\frac{n_i}{n_{i-j+1}} - 1} \qquad \begin{matrix} j = 2,3,... \\ i = 2,3,... \end{matrix} \tag{2.39}$$

We describe the application of polynomial extrapolation to (2.1). The process starts with the repeated application of (2.5) over the interval $[t_n, t_{n+1}]$ with a sequence of smaller stepsizes (sub-stepsizes). Consequently, a sequence of approximations are produced at the points $t_n \in \mathcal{T}_h$ in the following way:

Beginning at $t = t_0$, a stepsize $h = h_1$ is chosen and (2.5) is applied over the interval $[t_0, t_1]$ to obtain an approximation $x(t_1, h_1)$ at $t = t_1$. Thereafter the integration is repeated successively over the interval $[t_0, t_1]$ with a sequence of sub-stepsizes

$$h_i = h/n_i \qquad i = 2, 3, ..., \tag{2.40}$$

where $n_i \in F = \{n_i \in \mathbf{N} : n_{i+1} > n_i,\ i = 1, 2, ...\}$. As a result, a sequence of approximations $\{x(t_1, h_i)\}$ are produced at $t = t_1$. Now provided (2.5) permits an asymptotic expansion of the form (2.24) we can set at $\bar{t} = t_1$

$$T_{i,1} := x(\bar{t}, h_i) \qquad i = 1, 2, ..., \tag{2.41}$$

and apply (2.39) to generate further approximations to $x(t)$ at $t = t_1$. In this way an extrapolation tableau

$$\begin{array}{lll} T_{11} & & \\ T_{21} & T_{22} & \\ T_{31} & T_{32} & T_{33} \end{array} \tag{2.42}$$

is created. The first column of (2.42) contains the approximations (2.41) while the remaining columns are generated column by column using (2.39). The last entry in (2.42) represents the final approximation to $x(t_1)$. Similarly, the procedure is continued at the points $t_1, ..., t_N$ with the starting value of the current step taken as the final approximation of the previous step. Since (2.39) is based on extrapolating to $h = 0$ by using a polynomial the process outlined above is referred to as polynomial extrapolation.

Each $T_{i,j}$ in (2.42) represents an approximation to $x(t)$ at $t = t_{n+1}$. The local error in the computed solution in a fixed column M and row $M + 1$ is given by

$$E_{M+1,M} := \|T_{M+1,M} - x(t)\|. \tag{2.43}$$

Under the assumption[2] that the higher order approximation is more accurate than the

[2]This basic assumption is made in every ODE integrator.

lower order approximation, that is

$$E_{M+1,M+1} << E_{M+1,M}, \tag{2.44}$$

$E_{M+1,M}$ can be estimated by

$$E_{M+1,M} \cong \|T_{M+1,M} - T_{M+1,M+1}\| . \tag{2.45}$$

A very effective strategy for the order and stepsize control is due to Deuflhard [3, 4]. There the order and stepsize is allowed to vary simultaneously during the integration process. The local error in the computed solution returned from the kth column of (2.42) can be estimated by (2.45) as

$$E_{k+1,k} \cong \|T_{k+1,k} - T_{k+1,k+1}\| . \tag{2.46}$$

Error control is enforced by requiring that

$$E_{k+1,k} < \epsilon, \tag{2.47}$$

where ϵ is the prescribed error tolerance, as before. If this condition is fulfilled then the order of the method is $k+1$ and the stepsize for the next integration step can be estimated by

$$h_{next} \cong \left(\frac{\epsilon}{E_{k+1,k}} \right)^{\frac{1}{k+1}} . \tag{2.48}$$

In order to compare the work for different columns k, Deuflhard introduces the normalized work per unit step

$$W_k := \frac{A_{k+1}}{H_k} H, \tag{2.49}$$

where A_{k+1} is the work to obtain column k, and is defined recursively by

$$\begin{aligned} A_1 &:= n_1 + 1 \\ A_{k+1} &:= A_k + n_{k+1}. \end{aligned} \tag{2.50}$$

The optimal column index q is then defined by

$$W_q = \min_{k=1,2,\ldots,k_f} W_k, \tag{2.51}$$

where k_f is the final column in which the error criterion (2.47) was satisfied. Having determined q, the stepsize for the next step can be estimated by

$$H_q \cong \left(\frac{\epsilon}{E_{q+1,q}}\right)^{\frac{1}{q+1}}. \tag{2.52}$$

At this stage the derivation of the stepsize predictor is completed. However, in order to realize an efficient order and stepsize control mechanism, Deuflhard makes the following refinements to the above strategy.

Firstly, if convergence occurs in column

$$q = k_f < k_{\max},$$

where $k_{\max}$ is the largest allowable column, then instead of taking H_q as the next step, one might aim to increase the stepsize so that convergence occurs in column $q+1$. For this purpose, Deuflhard introduces the factors $\alpha(k,q)$

$$\begin{aligned} \alpha(k,q) &:= \epsilon^{\frac{A_{k+1}-A_{q+1}}{(k+1)(A_q-A_1+1)}}, \qquad k<q \\ \alpha(q,q) &\equiv 1, \end{aligned} \tag{2.53}$$

by which H_k is to be multiplied, so that convergence occurs in column $q+1$. Thus, the stepsize H_{q+1} can be estimated by

$$H_{q+1} \cong \alpha(q,q+1)\, H_q. \tag{2.54}$$

Now, (2.54) is only efficient if the work per unit step is not increased, that is if

$$W_{q+1} < W_q.$$

In other words if

$$\frac{A_{q+2}}{H_{q+1}} < \frac{A_{q+1}}{H_q},$$

that is if

$$A_{q+2} < \alpha(q,q+1)\, A_{q+1}. \tag{2.55}$$

It is to be noted, in the above discussion, the quantities W_k and the factors $\alpha(k,q)$ depend only on ϵ and n_k so they are computed once during initialization.

Secondly, a stepsize reduction strategy has to be considered. This is addressed by computing stepsize estimates

$$\bar{H}_k := \alpha(k,q)\, H_k, \qquad k = 1, 2, ..., q-1, \tag{2.56}$$

during the current step. Then if

$$\alpha(k, q+1)\, \bar{H}_k < H, \tag{2.57}$$

that is if $\bar{H}_k$ is "too small", the current step is rejected and the step is redone using $\bar{H}_k$. During initialization this check is made for all k. Thereafter, the convergence is only checked within an order window

$$\max(1, q-1) \leq k \leq \min(k_{\max}, q+1). \tag{2.58}$$

In the next section we introduce the method of lines for solving partial differential equations.

2.5 The Method of Lines

The *method of lines* (MOL) is a numerical technique which is commonly used for solving partial differential equations (PDEs). The solution process involves the semi-discretization of a PDE with respect to the spatial variable(s). This results in the PDE being converted into a system of ODEs.

As an illustrative example we consider the linear[3] one-dimensional diffusion equation

[3]The MOL is also applicable to nonlinear PDEs

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial \rho^2}, \qquad 0 \leq \rho \leq 1, \qquad t \geq 0, \tag{2.59}$$

with the boundary conditions

$$u(0,t) = u(1,t) = 0, \tag{2.60}$$

and initial condition

$$u(\rho,0) = g(\rho). \tag{2.61}$$

A spatial discretization of (2.59) is accomplished by specifying a uniform mesh

$$\mathcal{M} = \left\{\rho_i : \rho_i = i\Delta\rho,\ i = 0,1,...,n+1,\ \Delta\rho = (n+1)^{-1}\right\} \tag{2.62}$$

on the interval $[0,1]$, and by using the discrete approximations

$$u(\rho_i,t) \approx x_i(t), \tag{2.63}$$

$$\frac{\partial^2 u}{\partial \rho^2} \approx \frac{1}{(\Delta\rho)^2}(x_{i-1} - 2x_i + x_{i+1}), \quad i = 1,2,...,n \tag{2.64}$$

for the true solution and the spatial derivative at the interior points of $\mathcal{M}$, respectively. The boundary conditions of (2.59) imply that

$$x_0(t) = x_{n+1}(t) = 0. \tag{2.65}$$

Substitution of (2.64) into (2.59) results in a system of n first order ODEs of the form

$$\begin{aligned}
\frac{dx_1(t)}{dt} &= \frac{1}{(\Delta\rho)^2}(-2x_1 + x_2), \\
\frac{dx_2(t)}{dt} &= \frac{1}{(\Delta\rho)^2}(x_1 - 2x_2 + x_3), \\
&\vdots \\
\frac{dx_{n-1}(t)}{dt} &= \frac{1}{(\Delta\rho)^2}(x_{n-2} - 2x_{n-1} + x_n), \\
\frac{dx_n(t)}{dt} &= \frac{1}{(\Delta\rho)^2}(x_{n-1} - 2x_n).
\end{aligned}$$

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

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \frac{1}{(\Delta\rho)^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix},$$

that is as

$$\frac{dx}{dt} = Ax, \tag{2.66}$$

where $x = [x_1, x_2, ..., x_n]^T$ and A is an $(n \times n)$ matrix given by

$$A = \frac{1}{(\Delta\rho)^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}$$

The matrix of A has n distinct eigenvalues which are given by [23]

$$\lambda_j = \frac{-2}{(\Delta\rho)^2}\left(1 + \cos\frac{j\pi}{n+1}\right), \qquad j = 1, 2, ..., n. \tag{2.67}$$

We discuss the topic of stiff ODEs next.

2.6 Stiff Ordinary Differential Equations

The theory on stiff ordinary differential equations has been reviewed by a number of authors [1, 2, 6, 13, 19, 24], see also Parumasur [21]. Here we intend to give an overview of the topic, discussing only those aspects of the theory which are relevant to the present study only. We begin by considering a special class of ODEs in the form (2.1)

$$\epsilon\frac{dx}{dt} = f(x,t), \qquad 0 < \epsilon \ll 1, \qquad x(0) = x_0. \tag{2.68}$$

The small value of ϵ makes the solution of (2.68) difficult. When integrating such a system with an explicit numerical method, the main concern is that of numerical stability. The small value of ϵ forces the algorithm to choose relatively small time steps in order to fulfil the stability requirements of the algorithm.

To expose these requirements we apply the explicit Euler method (2.8) to the system (2.66), noting that $(\Delta\rho)^2$ in (2.66) has the same role as ϵ in (2.68). As a result we obtain a recurrence relation for the approximations x_n

$$x_{n+1} = x_n + hAx_n,$$

or

$$\begin{aligned} x_{n+1} &= (I + hA)\, x_n \\ &= (I + hA)^n\, x_0 \end{aligned}$$

where I is the $(n \times n)$ identity matrix. Thus to ensure that the approximations x_n remain bounded as $n \to \infty$ we have to require [4]

$$|1 + h\lambda_{max}| < 1 \tag{2.69}$$

where λ_{max} is the largest eigenvalue of A. This condition may have serious implications as to the selection of the stepsize h. For example, in case of the MOL, for large n we see from (2.67)

$$\lambda_{max} \approx -\frac{4}{(\Delta\rho)^2},$$

and the requirement (2.69) becomes

$$h < \frac{(\Delta\rho)^2}{2}. \tag{2.70}$$

For $\Delta\rho$ small (2.70) means that the value of h has to be chosen extremely small. The stepsize is restricted by the requirement on numerical stability rather than that of accuracy. ODEs

[4] Recall that a matrix B^n tends to zero as $n \to \infty$ only if the largest eigenvalue of B has magnitude less than unity

having such a property are called *stiff ODEs*. It follows that explicit numerical methods are not suitable for the solution of stiff ODEs, especially those resulting from the MOL.

The solution of stiff ODEs requires the more powerful implicit numerical methods. The simplest such method is the implicit Euler method (2.9). When used to solve (2.66) it produces the recurrence relation

$$x_{n+1} = x_n + hAx_{n+1}, \tag{2.71}$$

or

$$\begin{aligned} x_{n+1} &= (I - hA)^{-1} x_n \\ &= C^n x_0, \end{aligned}$$

where $C = (I - hA)^{-1}$. From (2.67) we see that the eigenvalues of C will be positive, so the method is stable provided

$$\left\| (1 - h\lambda_{max})^{-1} \right\| < 1 \tag{2.72}$$

and for the MOL the condition is that

$$\left(1 + \frac{4}{(\Delta\rho)^2} h \right)^{-1} < 1. \tag{2.73}$$

Thus the method is stable for arbitrarily large stepsizes.

The main drawback of implicit methods is that they require the solution of a system of nonlinear equations. Simple *functional iteration* may be used for nonstiff ODEs. For the nonlinear system of equations

$$x = g(x), \tag{2.74}$$

where

$$g : \mathbf{R}^n \longrightarrow \mathbf{R}^n$$

the iteration takes the form

$$x^{(k+1)} = g\left(x^{(k)}\right), \qquad k = 0, 1, 2, ..., \tag{2.75}$$

where $x^{(0)}$ is a suitable starting value. The following theorem (see [13]) states the conditions under which (2.74) has a unique solution and (2.75) converges.

Theorem 2.6.1 *Let $g(x)$ satisfy a Lipschitz condition*

$$\|g(x_1) - g(x_2)\| \leq K \|x_1 - x_2\| \tag{2.76}$$

for all x_1, x_2, with Lipschitz constant K such that $0 < K < 1$. Then there exists a unique solution $x = x^*$ of (2.74). Furthermore, the iterates $x^{(k)}$ produced by (2.75) converge to x^*, that is $x^{(k)} \to x^*$ as $k \to \infty$.

For the implicit Euler method (2.9) equation (2.75) implies

$$x_{n+1}^{(k+1)} = x_n + hf\left(x_{n+1}^{(k)}, t_{n+1}\right), \qquad k = 0, 1, 2, ..., \tag{2.77}$$

where $x_{n+1}^{(0)}$ is usually obtained by the explicit Euler method (2.8), that is

$$x_{n+1}^{(0)} = x_n + f(x_n, t_n).$$

Subtracting (2.77) from (2.9) and taking norms gives

$$\begin{aligned} \left\|x_{n+1} - x_{n+1}^{(k+1)}\right\| &\leq h \left\|f(x_{n+1}, t_{n+1}) - f\left(x_{n+1}^{(k)}, t_{n+1}\right)\right\| \\ &\leq hL \left\|x_{n+1} - x_{n+1}^{(k)}\right\| \end{aligned}$$

where we assume the Lipschitz condition (2.76) holds for (2.77). Thus, according to Theorem 2.6.1, the iteration (2.77) converges provided

$$hL < 1. \tag{2.78}$$

Clearly, for stiff ODEs with large Lipschitz constant L functional iteration is not a suitable choice. Stiff ODEs require the more robust *Newton iteration*. When applied to the nonlinear system of equations

$$F(x) = 0 \tag{2.79}$$

where

$$F : \mathbf{R}^n \longrightarrow \mathbf{R}^n$$

the iteration has the form

$$x^{(k+1)} = x^{(k)} - J^{-1}\left(x^{(k)}\right) F\left(x^{(k)}\right), \qquad k = 0, 1, 2, ..., \tag{2.80}$$

where $J \equiv \frac{\partial F}{\partial x}$ is the Jacobi matrix of F.

For the implicit Euler method (2.9) the iteration takes the form

$$x_{n+1}^{(k+1)} = x_{n+1}^{(k)} - \left(I - hJ\left(x_n^{(k)}\right)\right)^{-1}\left(x_{n+1}^{(k)} - hf\left(x_{n+1}^{(k)}, t_{n+1}\right) - x_n\right). \tag{2.81}$$

In practice it is more convenient to write (2.81) in the form

$$P\Delta x^{(k)} = -F\left(x_{n+1}^{(k)}, x_n, t_{n+1}\right), \tag{2.82}$$

where

$$P := I - hJ,$$

$$F\left(x_{n+1}^{(k)}, x_n, t_{n+1}\right) := x_{n+1}^{(k)} - hf\left(x_{n+1}^{(k)}, t_{n+1}\right) - x_n,$$

and

$$\Delta x^{(k)} := x_{n+1}^{(k+1)} - x_{n+1}^{(k)},$$

is the increment for the difference in the iterations at each step. Thus the new iterate is given by

$$x_{n+1}^{(k+1)} = x_{n+1}^{(k)} + \Delta x^{(k)}.$$

Finally, as an application, to demonstrate the effectiveness of the method to be proposed, we consider a class of problems from chemical kinetics.

2.7 Class of Problems

When modelling chemical kinetics transport problems in the atmosphere the practitioner is usually faced with a large system of stiff PDEs. The stiffness in these problems is mainly due to

(i) chemical reactions having time constants which are widely separated in magnitude,

(ii) chemical reactions having time constants which are smaller than those for particle transport, and

(iii) the spatial discretization of the PDEs.

The solution of stiff chemical kinetics systems requires an implict method which have additional cost and storage requirements associated with them. On the other hand, the solution of chemical systems which couple fluid transport with chemical kinetics is much more difficult than that of chemical kinetics equations alone, since in this instance the governing equations are stiff systems of PDEs. The discretization of even a few PDEs by the MOL can lead to an extremely large system of ODEs, the numerical solution of which may have severe cost and storage constraints.

We consider a class of chemical problems which we feel will be adequate in demonstrating the versatility and usefulness of the amplitude-shape method to be presented. Furthermore, the class of problems to be considered will be a good representation of many actual physical problems encountered in chemical kinetics transport modelling.

In the study of chemical kinetics transport phenomena in the atmosphere the physical problem is often expressed as a system of PDEs [1]

$$\frac{\partial c^k}{\partial t} = \frac{\partial}{\partial \rho}\left(D\frac{\partial c^k}{\partial \rho} + Vc^k\right) + R^k + S^k, \quad k = 1, 2, ..., m, \tag{2.83}$$

where

c^k - concentration of the chemical species,

D - matrix of diffusion coefficients,

V - vector of mean atmospheric motion,

R^k - kinetic rate for c^k,

S^k - external source rate for c^k,

and ρ and t denote the spatial variable and time, respectively, with $a \leq \rho \leq b$ and $t \geq 0$. To obtain the system of ODEs in the form (2.1) we use the MOL. Firstly, we divide the interval $[a, b]$ into $n-1$ equal subintervals of size

$$h = \frac{b-a}{n-1}.$$

This creates a mesh of n equally spaced points

$$\rho_i = a + (i-1)\,h, \quad i = 1, 2, ..., n,$$

on the interval $[a, b]$. We associate with each of these points the functions x_i^k, defined by

$$x_i^k(t) = c^k(\rho_i, t), \quad k = 1, 2, ..., m, \quad i = 1, 2, ..., n, \tag{2.84}$$

that represent approximations of the true solution values taken at the points ρ_i.

Secondly, we consider a spatial discretization of each of the PDEs in (2.83) using the approximations x_i^k defined in (2.84) and replace the spatial derivatives with central difference

approximations. If we introduce the operators

$$L^k c^k = \frac{\partial}{\partial \rho}\left(D(\rho)\frac{\partial c^k}{\partial \rho} + V(\rho)\, c^k\right), \quad k = 1, 2, ..., m, \tag{2.85}$$

then the central difference approximations of (2.85), in terms of x_k^i, at the interior points ρ_i are given by

$$\begin{aligned}\left(L^k c^k\right)_i \approx\ & \frac{1}{h^2}\left(D\left(\rho_{i+\frac{1}{2}}\right) x_{i+1}^k - \left(D\left(\rho_{i+\frac{1}{2}}\right) + D\left(\rho_{i-\frac{1}{2}}\right)\right) x_i^k + D\left(\rho_{i-\frac{1}{2}}\right) x_{i-1}^k\right) + \\ & \frac{1}{2h}\left(V(\rho_{i+1})\, x_{i+1}^k - V(\rho_{i-1})\, x_{i-1}^k\right), \quad i = 2, 3, ..., n-1,\end{aligned} \tag{2.86}$$

where we use the notation

$$\rho_{i\pm\frac{1}{2}} = \frac{1}{2}\left(\rho_i + \rho_{i\pm 1}\right).$$

The forms of $\left(L^k c^k\right)_1$ and $\left(L^k c^k\right)_n$ depend on the values x_0^k and x_{n+1}^k. These values are obtained from the boundary conditions, which brings us to the third step in our discretization process.

Various kinds of boundary conditions may be considerd with (2.83). Here we take the zero normal derivative boundary conditions

$$\frac{\partial c^k}{\partial \rho}(a, t) = \frac{\partial c^k}{\partial \rho}(b, t) = 0, \quad k = 1, 2, ..., m, \tag{2.87}$$

as this seems to be very common in applications. Proceeding as for the interior points we approximate these boundary conditions using central difference approximations. We use at $\rho = a$ the approximations

$$\frac{\partial c^k}{\partial \rho}(a, t) \approx \frac{x_2^k - x_0^k}{2h}, \quad k = 1, 2, ..., m, \tag{2.88}$$

which gives

$$x_0^k = x_2^k, \quad k = 1, 2, ..., m.$$

Similarly, at $\rho = b$ we find that

$$x_{n+1}^k = x_{n-1}^k, \quad k = 1, 2, ..., m.$$

Substitution of these values for x_0^k and x_{n+1}^k into (2.86) gives the forms of $\left(L^k c^k\right)_1$ and $\left(L^k c^k\right)_n$, that is,

$$\begin{aligned}\left(L^k c^k\right)_1 &\approx \frac{2}{h^2}\left(D\left(\rho_{i+\frac{1}{2}}\right)+D\left(\rho_{i-\frac{1}{2}}\right)\right)\left(x_2^k-x_1^k\right)+\frac{1}{2h}\left(V\left(\rho_{i+1}\right)-V\left(\rho_{i-1}\right)\right)x_2^k,\\ \left(L^k c^k\right)_n &\approx \frac{2}{h^2}\left(D\left(\rho_{i+\frac{1}{2}}\right)+D\left(\rho_{i-\frac{1}{2}}\right)\right)\left(x_n^k-x_{n-1}^k\right)+\frac{1}{2h}\left(V\left(\rho_{i+1}\right)-V\left(\rho_{i-1}\right)\right)x_{n-1}^k.\end{aligned} \tag{2.89}$$

Now substituting (2.86) and (2.89) into (2.83) and setting

$$f^k = L^k c^k + R^k + S^k, \quad k = 1, 2, ..., m,$$

results in a system of $(n \times m)$ ODEs of the form

$$\frac{dx^k}{dt} = f^k, \quad k = 1, 2, ..., m, \tag{2.90}$$

where

$$x^k = \left(x_1^k, x_2^k, ..., x_n^k\right), \quad k = 1, 2, ..., m.$$

In the final step we consider a discretization of the initial conditions for (2.83). This gives the initial conditions for (2.90). If the initial conditions for (2.83) are given as

$$c^k(\rho, 0) = q^k, \quad k = 1, 2, ..., m,$$

then by using (2.84) the initial conditions for (2.90) are obtained as

$$x_i^k(0) = q_i^k, \quad i = 1, 2, ...n,$$

or in a vector form

$$x^k(0) = q^k, \quad k = 1, 2, ..., m. \tag{2.91}$$

Combining equations (2.90) and (2.91) gives the system of ODEs

$$\begin{aligned}\frac{dx^k}{dt} &= f^k\\ x^k(0) &= q^k, \qquad k = 1, 2, ..., n.\end{aligned} \tag{2.92}$$

In most cases, (2.92) represents a stiff system of ODEs and poses a challenging numerical task. In the next chapter we propose the amplitude-shape method for solving (2.92).

Chapter 3

Amplitude-Shape Method

3.1 Introduction

In section 2.7 we observed that a description of a chemical kinetics transport problem involves a system of partial differential equations (PDEs). In fact, most physical phenomena occurring in practical applications can be described by PDEs. The solution of the PDE, denoted by $u(\rho, t)$ usually depends on time t and on one or more spatial variables ρ. The simplest method for solving PDEs is the *separation of variables.* Although this method is very effective, it is only applicable to linear homogeneous PDEs which have linear homogeneous boundary conditions. However, if the method is applicable, then the solution $u(\rho, t)$ is written as the product

$$u(\rho, t) = \phi(t)\, v(\rho),$$

of functions $\phi(t)$ and $v(\rho)$ depending on t and ρ, respectively. If such a separation is not possible, then the solution $u(\rho, t)$ is often represented in terms of an infinite sum

$$u(\rho, t) = \sum_{i=0}^{\infty} \phi_i(t)\, v_i(\rho),$$

which in practical computations is truncated to a finite sum. Here we propose to express the solution $u(\rho, t)$ in the form

$$u(\rho, t) = \phi(t)\, v(\rho, t), \tag{3.1}$$

which may be thought of as an *incomplete separation of variables*. The motivation for expressing the solution in this form is based on the following observation: For a physical process undergoing rates of change, inasmuch as the solution $u(\rho, t)$ may be changing with respect to time, its overall change with respect to ρ is only very slightly affected. Thus, the representation (3.1) would be of practical relevance if it is possible to normalize the function $v(\rho, t)$ in such a way that it changes with time much more slowly than the function $\phi(t)$. Since the variable ρ usually represents the spatial variable(s), $v(\rho, t)$ is called the *shape* function. On the other hand, the function $\phi(t)$ may be considered as an *amplitude* function since it describes the essential change of $u(\rho, t)$ with time.

The approach to be presented, which considers representing the solution as a product of an amplitude function and a shape function, will be called the *amplitude-shape method* (ASM). The ASM adapts the notion of representing the solution in the form (3.1) from the *quasistatic method* [20, 5, 17], originally developed in nuclear reactor physics. When the PDEs describing a realistic nuclear reactor model are discretized, by the method of lines or a similar method of approximating the spatial dependence, an extremely large system of ODEs result, and is usually highly stiff. The solution of such a system is usually simplified with the application of the quasistatic method. Originally the method was based on purely heuristic grounds with the mathematical foundations of the method being given much later by Mika [15]. Subsequently, Kozakiewicz and Mika [11] proposed an algorithm, based on the representation (3.1), for the numerical solution of ODEs. That algorithm consisted in using different, and unfortunately, fixed time steps for the amplitude and shape equations. Parumasur [21] considered the application of this algorithm together with a modified version to the solution of ODEs resulting from the MOL. Earlier, Mika and Scribani [16] also reported preliminary results for such ODEs.

3.2 Amplitude-Shape Equations

We consider a system of ODEs in $\mathbf{R}^n, n \geq 1$

$$\begin{aligned} \frac{dx}{dt} &= f(x,t) \\ x(0) &= q, \quad t \geq 0, \end{aligned} \tag{3.2}$$

where the vector x can be written in terms of subvectors x^k, that is

$$x = \left(x^1, x^2, \ldots, x^m\right), \tag{3.3}$$

which have the form

$$x^k = \left(x_1^k, x_2^k, \ldots, x_{n_k}^k\right), \quad k = 1,2,\ldots,m, \tag{3.4}$$

with

$$\sum_{k=1}^{m} n_k = n.$$

Similarly, the initial vector q has the form

$$q = \left(q^1, q^2, \ldots, q^m\right).$$

The norm of x is defined as

$$\|x\| = \max\left\{\|x^1\|, \|x^2\|, \ldots, \|x^m\|\right\}$$

with

$$\left\|x^k\right\| = \max\left\{\left|x_1^k\right|, \left|x_2^k\right|, \ldots, \left|x_{n_k}^k\right|\right\}, \quad k = 1,2,\ldots,m.$$

It is to be noted that, in order to avoid confusion with the norm notation, we use the same notation for the norms in different spaces.

The basic assumption necessary for the validity of the ASM is that each of the subvectors x^k can be represented as a product of a scalar amplitude ϕ^k and a shape vector v^k so that

$$x^k(t) = \phi^k(t)\, v^k(t), \quad k = 1,2,\ldots,m. \tag{3.5}$$

The shape vector is supposed to change slowly with time. We attempt to achieve this by an appropriate normalization. For each k introduce a nonzero weight vector

$$w^k = \left(w_1^k, w_2^k, \ldots, w_{n_k}^k\right), \tag{3.6}$$

and require that the scalar product of the vectors w^k and v^k is independent of time and equal to one, say. It is convenient to express the scalar product of the vectors in $\mathbf{R}^{n_k}$ in terms of the operators W^k

$$\left(W^k x^k\right)(t) := \left(w^k, x^k(t)\right), \quad w^k, x^k \in \mathbf{R}^{n_k}, \quad k = 1, 2, \ldots, m, \tag{3.7}$$

where $W^k : \mathbf{R}^{n_k} \longrightarrow \mathbf{R}$ is defined by

$$\left(W^k x^k\right)(t) := \sum_{i=1}^{n_k} w_i^k x_i^k(t), \quad k = 1, 2, ..., m. \tag{3.8}$$

Thus we require

$$\left(W^k v_k\right)(t) \equiv 1, \quad k = 1, 2, ..., m. \tag{3.9}$$

With this it follows that

$$\begin{aligned} \left(W^k x^k\right)(t) &= \left(W^k \phi^k v^k\right)(t) \\ &= \phi^k(t) \left(W^k v^k\right)(t) \end{aligned}$$

that is

$$\phi^k(t) = \left(W^k x^k\right)(t), \quad k = 1, 2, ..., m, \tag{3.10}$$

and from (3.5)

$$v^k(t) = \frac{x^k(t)}{\phi^k(t)}, \quad k = 1, 2, ..., m. \tag{3.11}$$

We now prove that the representation (3.5) is unique.

Theorem 3.2.1 *The representation (3.5) is unique provided ϕ^k does not vanish on T_a.*

Proof

To prove we assume the contrary. Then for all or some k, x^k will have two different representations of the form (3.5), that is

$$x^k(t) = \phi^k(t)\, v^k(t), \quad k = 1, 2, ..., m, \tag{3.12}$$

and

$$x^k(t) = \bar{\phi}^k(t)\, \bar{v}^k(t), \quad k = 1, 2..., m, \tag{3.13}$$

where ϕ^k, $\bar{\phi}^k$ and v^k, $\bar{v}^k$ are defined by (3.10) and (3.11), respectively. Equations (3.12) and (3.13) imply

$$\phi^k(t)\, v^k(t) = \bar{\phi}^k(t)\, \bar{v}^k(t), \quad k = 1, 2, ..., m. \tag{3.14}$$

Operating on both sides of (3.14) with W^k and using (3.9) we obtain

$$\phi^k(t) = \bar{\phi}^k(t), \quad k = 1, 2, ..., m, \tag{3.15}$$

and from (3.14) we have

$$v^k(t) = \bar{v}^k(t), \quad k = 1, 2, ..., m. \tag{3.16}$$

provided ϕ^k does not vanish on T_a. Equations (3.15) and (3.16) proves the uniqueness of (3.5). □

We will show in the next section that ϕ^k does not vanish, possibly, over the smaller interval T_b, where $0 < b \leq a$, if the initial vector q is appropriately chosen.

We have as a result of (3.9)

$$\left(W^k \frac{dv^k}{dt}\right)(t) = \frac{d\left(W^k v^k\right)(t)}{dt} = \frac{d}{dt}(1) \equiv 0.$$

After rewriting (3.2) in terms of x^k we have

$$\begin{aligned} \frac{dx^k}{dt} &= f^k\left(x^1, x^2, \ldots, x^m, t\right), \\ x^k(0) &= q^k, \quad k = 1, 2, \ldots, m. \end{aligned} \tag{3.17}$$

and by substituting (3.5) into (3.17) we obtain

$$\begin{aligned} \frac{d\phi^k}{dt}v^k + \phi^k\frac{dv^k}{dt} &= f^k\left(\phi^1 v^1, \phi^2 v^2, \ldots, \phi^m v^m, t\right), \\ \phi^k(0)\, v^k(0) &= q^k, \quad k = 1, 2, \ldots . m. \end{aligned} \tag{3.18}$$

By operating with W^k on both sides of the two equations in (3.18) we obtain the IVP describing the amplitude

$$\begin{aligned} \frac{d\phi^k}{dt} &= \gamma^k, \\ \phi^k(0) &= \nu^k, \quad k = 1, 2, ..., m \end{aligned} \tag{3.19}$$

where

$$\begin{aligned} \gamma^k\left(\phi^1, \ldots, \phi^m, v^1, \ldots, v^m, t\right) &:= W^k f^k\left(\phi^1 v^1, \ldots, \phi^m v^m, t\right), \\ \nu^k &:= \left(W^k q^k\right)(t). \end{aligned}$$

Then by using (3.19) in (3.18) we obtain the IVP for the shape

$$\begin{aligned} \frac{dv^k}{dt} &= g^k, \\ v^k(0) &= r^k, \quad k = 1, 2, \ldots . m, \end{aligned} \tag{3.20}$$

where

$$\begin{aligned} g^k\left(\phi^1, \ldots, \phi^m, v^1, \ldots, v^m, t\right) &:= \frac{1}{\phi^k}\left(f^k - \gamma^k v^k\right), \\ r^k &:= \frac{1}{\nu^k} q^k. \end{aligned}$$

The system of ODEs (3.19) for the amplitude and the system of ODEs (3.20) for the shape can be written as the combined system of ODEs

$$\begin{aligned} \frac{dz}{dt} &= m(z, t) \\ z(0) &= s, \end{aligned} \tag{3.21}$$

if we define a new vector function $z : \mathbf{R} \to \mathbf{R}^{n+m}$ as

$$z = \left(\phi^1, \phi^2, \ldots, \phi^m, v_1^1, v_2^1, \ldots, v_{n_1}^1, v_1^2, v_2^2, \ldots v_{n_2}^2, \ldots, v_1^m, v_2^m, \ldots, v_{n_m}^m\right).$$

The function $m : \mathbf{R}^{n+m+1} \to \mathbf{R}^{n+m}$ and the vector $s \in \mathbf{R}^{n+m}$ are similarly defined.

From above we see that for the system of ODEs (3.2) a corresponding amplitude-shape system (3.21) may be derived. Naturally, we cannot expect the ASM to be useful for all systems of ODEs arising in this way. The motivation for using the ASM must be based on physical considerations. For example, if we are dealing with a system of ODEs that represents an evolution of a physical system, for which we expect small spatial oscillations, then the ASM might be a viable solution approach. Such a situation may arise in chemical kinetics, where the effect of diffusion on the spatial distribution of reagents is usually very slow in comparison with the effect of chemical reactions.

Finally, we should be able to choose an appropriate normalization or, in other words, the vectors w^k. The most natural one, and the one which we have found to be adequate in our examples of application, is to take simply $w_i^k = 1$. This would imply that $(\phi^1, \phi^2, \ldots, \phi^m)$ represents the total number of particles, or the total mass, or a similar quantity. There are, however, other possibilities (see [11, 18]).

3.3 Equivalence of Systems

In Theorem 2.1.1 it is stated that if the functions $f(x,t)$ satisfy the smoothness properties (i) and (ii) listed in the theorem then the system of ODEs

$$\begin{aligned} \frac{dx}{dt} &= f(x,t) \\ x(0) &= q \end{aligned} \tag{3.22}$$

will have a unique solution in $E_a = D \times T_a$ where

$$D = \{x \in \mathbf{R}^n : \|x - q\| \leq d, \quad d > 0\}, \tag{3.23}$$

and $T_a = [0, a]$, $a > 0$. On the other hand, with the operators W^k defined by (3.8) we see that the functions $m(z,t)$ in the amplitude-shape system

$$\begin{aligned} \frac{dz}{dt} &= m(z,t), \\ z(0) &= s, \end{aligned} \tag{3.24}$$

will have the same smoothness properties as $f(x,t)$ in (3.22) if $\phi^k(t)$ are bounded away from zero so that dividing by ϕ^k does not introduce singularities. To avoid dividing by zero we have to make an assumption that the initial condition q is such that the values of $\left|\nu^k\right|$ are bounded away from zero. In other words we will assume that there exists a constant $\beta > 0$ such that

$$\left|\nu^k\right| \geq \beta > 0, \quad k = 1, 2, ..., m. \tag{3.25}$$

With the conditions listed in Theorem 2.1.1 the solution of (3.22) is continuous at the initial value. This means that for any value of μ satisfying the inequality $0 < \mu < \beta$, there exists the interval $T_b = [0, b]$, $0 < b \leq a$, such that

$$\left|W^k x^k(t) - \nu^k\right| \leq \mu, \quad k = 1, 2, ..., m, \quad t \in T_b. \tag{3.26}$$

On the other hand, since

$$\left|W^k x^k(t)\right| = \left|\sum_{i=1}^{n_k} w_i^k x_i^k(t)\right| \leq w^k \left|x_i^k(t)\right|, \tag{3.27}$$

where $w^k = \sum_{i=1}^{n_k} w_i^k$, we have

$$\begin{aligned} \left|W^k x^k(t) - \nu^k\right| &= \left|W^k x^k(t) - W^k q^k\right| \\ &= \left|W^k \left(x^k(t) - q^k\right)\right| \\ &\leq w^k \left|x^k(t) - q^k\right|. \end{aligned}$$

That is

$$\left|W^k x^k(t) - \nu^k\right| \leq w^k \left|x^k(t) - q^k\right|, \quad k = 1, 2, ..., m. \tag{3.28}$$

Hence, to satisfy (3.23) we have to take

$$\left|x^k(t) - q^k\right| \leq \frac{\mu}{w^k}, \quad k = 1, 2, ..., m. \tag{3.29}$$

In the norm this becomes

$$\|x(t) - q\| \leq \frac{\mu}{w}, \tag{3.30}$$

where $w = \max\limits_{1 \leq k \leq m} w^k$.

Thus with D in (3.23) replaced by

$$D = \left\{ x \in \mathbf{R}^n : \|x - q\| \leq \frac{\mu}{w} \right\}, \tag{3.31}$$

it is possible to define the region

$$D_1 = \left\{ z \in \mathbf{R}^{n+m} : \|z - s\| \leq d_1,\ d_1 > 0 \right\}, \tag{3.32}$$

and we see that the functions $m(z,t)$ will then satisfy the conditions (i) and (ii) of Theorem 2.1.1 on $E_b = D_1 \times T_b$. From this it follows

Theorem 3.3.1 *The system (3.24) has a unique solution on $E_c = D_1 \times T_c$ with $T_c = [0, c]$, where $c = \min\left\{b, \frac{d_1}{M_1}\right\}$ and*

$$M_1 = \sup_{(z,t) \in E_b} \|m(z,t)\|.$$

Finally, we prove that the systems (3.22) and (3.24) are equivalent on the interval T_c.

Theorem 3.3.2 *The systems (3.22) and (3.24) are equivalent on the interval T_c.*

Proof

From the derivation of the amplitude shape equations in the previous section it follows that if x^k is the solution of (3.17) then ϕ^k and v^k are the solutions of (3.19) and (3.20), respectively.

Conversely, if ϕ^k and v^k are the solutions of (3.19) and (3.20) then we must show that (3.9) and (3.5) are satisfied on T_c. Firstly, to prove (3.9), denote by u^k the functions defined on

the left hand side of (3.9)

$$u^k := \left(W^k v^k\right)(t), \quad k = 1, 2, ..., m, \quad t \in T_c,$$

and apply the operators W^k to (3.20). As a result we obtain for u^k the system of ODEs

$$\begin{aligned} \frac{du^k}{dt} &= \frac{\gamma^k}{\phi^k}\left(1 - u^k\right), \\ u^k(0) &= 1 \quad k = 1, 2, ..., m, \quad t \in T_c, \end{aligned}$$

which has a solution $u^k \equiv 1$ on T_c. This is the unique solution of (3.20) on T_c since once again it follows that the operators W^k do not change the smoothness properties of the functions defined on the right hand side of (3.33). Thus (3.9) is satisfied on T_c.

On the other hand to prove (3.5), define by y^k the functions defined on the right hand side of (3.5)

$$y^k := \phi^k(t)\, v^k(t), \quad k = 1, 2, ..., m, \quad t \in T_c, \tag{3.33}$$

where ϕ^k and v^k are the solutions of (3.19) and (3.20), respectively. That is

$$\frac{d\phi^k}{dt} = \gamma^k, \quad k = 1, 2, ..., m, \tag{3.34}$$

and

$$\frac{dv^k}{dt} = \frac{1}{\phi^k}\left(f^k - \gamma^k v^k\right), \quad k = 1, 2, ..., m. \tag{3.35}$$

By substituting (3.34) into (3.35) and rearranging (3.35) we have

$$v^k \frac{d\phi^k}{dt} + \phi^k \frac{dv^k}{dt} = f^k, \quad k = 1, 2, ..., m,$$

which is the same as

$$\frac{d}{dt}\left(\phi^k v^k\right) = f^k\left(\phi^1 v^1, ..., \phi^m v^m, t\right), \quad k = 1, 2, ..., m. \tag{3.36}$$

By using (3.33) in (3.36) we have

$$\frac{d}{dt}\left(y^k(t)\right) = f^k\left(y^1, y^2, ..., y^m, t\right), \quad k = 1, 2, ..., m,$$

and we see that $y^k(t)$ and $x^k(t)$ satisfy the same equation (3.22). Thus by uniqueness of the solution of (3.22) it follows that $y^k(t) \equiv x^k(t)$ on T_c. □

In the next chapter we describe the numerical procedure for solving the amplitude and shape equations.

Chapter 4

Amplitude-Shape Numerical Procedure

4.1 Introduction

We see that with the ASM the system of ODEs (3.2) is to be transformed into an amplitude system

$$\begin{aligned} \frac{d\phi}{dt} &= \gamma\left(\phi^1, ..., \phi^m, v^1, ..., v^m, t\right) \\ \phi(0) &= \nu, \end{aligned} \tag{4.1}$$

characterized by a rapidly changing solution

$$\phi = \left(\phi^1, \phi^2, ..., \phi^m\right),$$

and a shape system

$$\begin{aligned} \frac{dv}{dt} &= g\left(\phi^1, ..., \phi^m, v^1, ..., v^m, t\right) \\ v(0) &= r, \end{aligned} \tag{4.2}$$

characterized by a relatively smooth solution

$$v = \left(v^1, v^2, ..., v^m\right).$$

The main objective of the ASM is to transform the original system of ODEs so that only a few ODEs remain stiff. With such an approach it will be seen (in the next chapter) that a significant reduction in numerical effort may be realized.

In this chapter, we present two basic numerical schemes for solving (4.1)-(4.2). These schemes combines an implicit method with an explicit one and we refer to the combined method as an Amplitude-Shape Numerical Scheme (ASNS). It is to be noted that these are by no means unique choices and other schemes are also possible (see Hofer [10], for example).

4.2 Amplitude-Shape Numerical Scheme

As a first ASNS for solving (4.1)-(4.2) we propose

$$\begin{aligned} v^*_{n+1} &= v_n + hg_n \\ \phi_{n+1} &= \phi_n + h\gamma_{n+1} \\ v_{n+1} &= v_n + hg_{n+1}, \end{aligned} \tag{4.3}$$

where

$$g_n \equiv g\left(\phi_n, v_n, t_n\right), \qquad \gamma_{n+1} \equiv \gamma\left(\phi_{n+1}, v^*_{n+1}, t_{n+1}\right), \qquad g_{n+1} \equiv g\left(\phi_{n+1}, v^*_{n+1}, t_{n+1}\right).$$

This is based on the application of the implicit Euler method (2.9) to the amplitude subsystem (4.1) and the explicit Euler method (2.8) to the shape subsystem (4.2).

As a second ASNS for solving (4.1)-(4.2) we propose

$$
\begin{aligned}
v_{n+1}^{*} &= v_n + hg_n, \\
\phi_{n+1} &= \phi_n + \frac{1}{2}h\left(\gamma_n + \gamma_{n+1}\right), \\
v_{n+1} &= v_n + \frac{1}{2}h\left(g_n + g_{n+1}\right),
\end{aligned}
\tag{4.4}
$$

where

$$
\begin{aligned}
g_n &\equiv g\left(\phi_n, v_n, t_n\right), \qquad & g_{n+1} &\equiv g\left(\phi_{n+1}, v_{n+1}^{*}, t_{n+1}\right), \\
\gamma_n &\equiv \gamma\left(\phi_n, v_n, t_n\right), \qquad & \gamma_{n+1} &\equiv \gamma\left(\phi_{n+1}, v_{n+1}^{*}, t_{n+1}\right).
\end{aligned}
$$

This is based on the implicit RK2 scheme (2.23) applied to (4.1) and the explicit RK2 scheme (2.23) applied to (4.2).

We prove the convergence of (4.4) only, since the proof of (4.3) would be completely analogous. We begin by proving that (4.4) is consistent of order 2.

4.3 Order and Consistency

We write (4.4) as

$$
\begin{aligned}
v_1^{*} - v_0 &= hg_0, \\
\phi_1 - \phi_0 &= \frac{1}{2}h\left(\gamma_0 + \gamma_1\right), \\
v_1 - v_0 &= \frac{1}{2}h\left(g_0 + g_1\right).
\end{aligned}
\tag{4.5}
$$

where we use the simplified notation

$$
\begin{aligned}
&v_0 := v_n, \quad && \phi_0 := \phi_n, \quad && g_0 := g_n, \quad && \gamma_0 := \gamma_n, \\
&v_1^{*} := v_{n+1}^{*}, \quad && \phi_1 := \phi_{n+1}, \quad && v_1 := v_{n+1}, \quad && g_1 := g_{n+1}, \quad && \gamma_1 := \gamma_{n+1}.
\end{aligned}
$$

By assuming that γ and g are sufficiently smooth, by expanding γ_1 and g_1 in a Taylor series about (ϕ_0, v_0, t_0), and by using the notation

$$
\Delta\phi_1 = \phi_1 - \phi_0,
$$

$$\Delta v_1 = v_1 - v_0,$$

we can write (4.5) as

$$\begin{aligned}\Delta\phi_1 &= \frac{1}{2}h\left(2\gamma_0 + \gamma_{\phi,0}\Delta\phi_1 + h\gamma_{v,0}g_0 + h\gamma_{t,0}\right) + O\left(h^3\right),\\ \Delta v_1 &= \frac{1}{2}h\left(2g_0 + g_{\phi,0}\Delta\phi_1 + hg_{v,0}g_0 + hg_{t,0}\right) + O\left(h^3\right),\end{aligned} \tag{4.6}$$

where

$$\gamma_0 \equiv \frac{\partial\gamma}{\partial\phi}(\phi_0, v_0, t_0), \quad \gamma_{v,0} \equiv \frac{\partial\gamma}{\partial v}(\phi_0, v_0, t_0), \quad \gamma_{t,0} \equiv \frac{\partial\gamma}{\partial t}(\phi_0, v_0, t_0),$$
$$g_{\phi,0} \equiv \frac{\partial g}{\partial\phi}(\phi_0, v_0, t_0), \quad g_{v,0} \equiv \frac{\partial g}{\partial v}(\phi_0, v_0, t_0), \quad g_{t,0} \equiv \frac{\partial g}{\partial t}(\phi_0, v_0, t_0).$$

Solving for $\Delta\phi_1$ from the first equation in (4.6) yields

$$\begin{aligned}\Delta\phi_1 &= \left(1 - \frac{1}{2}h\gamma_{\phi,0}\right)^{-1}\left(h\gamma_0 + \frac{1}{2}h^2\gamma_{v,0}g_0 + \frac{1}{2}h^2\gamma_{t,0} + O\left(h^3\right)\right)\\ &= \left(1 + \frac{1}{2}h\gamma_{\phi,0}\right)\left(h\gamma_0 + \frac{1}{2}h^2\gamma_{v,0}g_0 + \frac{1}{2}h^2\gamma_{t,0}\right) + O\left(h^3\right)\end{aligned}$$

where we used

$$\left(1 - \frac{1}{2}h\gamma_{\phi,0}\right)^{-1} = 1 + \frac{1}{2}h\gamma_{\phi,0} + \frac{1}{4}h^2\gamma_{\phi,0}^2 + \frac{1}{8}h^3\gamma_{\phi,0}^3 + \ldots.$$

which holds for h sufficiently small.

Hence,

$$\Delta\phi_1 = h\gamma_0 + \frac{1}{2}h^2\gamma_{\phi,0}\gamma_0 + \frac{1}{2}h^2\gamma_{v,0}g_0 + \frac{1}{2}h^2\gamma_{t,0} + O\left(h^3\right) \tag{4.7}$$

and substituting (4.7) into the second equation (4.6) leads to

$$\Delta v_1 = hg_0 + \frac{1}{2}h^2g_{\phi,0}\gamma_0 + \frac{1}{2}h^2g_{v,0}g_0 + \frac{1}{2}h^2g_{t,0} + O\left(h^3\right) \tag{4.8}$$

Now expanding $\Delta\phi(t_1) = \phi(t_1) - \phi(t_0)$ in the Taylor series about (ϕ_0, v_0, t_0) shows that

$$\begin{aligned}\Delta\phi(t_1) &= \phi(t_1) - \phi(t_0)\\ &= h\phi_0' + \frac{1}{2}h^2\phi_0'' + O\left(h^3\right)\\ &= h\gamma_0 + \frac{1}{2}h^2\gamma_0' + O\left(h^3\right)\\ &= h\gamma_0 + \frac{1}{2}h^2\left(\gamma_{\phi,0}\gamma_0 + \gamma_{v,0}g_0 + \gamma_{t,0}\right) + O\left(h^3\right),\end{aligned}$$

where the prime represents differentiation with respect to t and we have used the chain rule.

Hence, we have

$$\Delta\phi(t_1) = h\gamma_0 + \frac{1}{2}h^2\gamma_{\phi,0}\gamma_0 + \frac{1}{2}h^2\gamma_{v,0}g_0 + \frac{1}{2}h^2\gamma_{t,0} + O\left(h^3\right). \tag{4.9}$$

Similarly, by expanding $\Delta v(t_1) = v(t_1) - v(t_0)$ about (ϕ_0, v_0, t_0) we find

$$\Delta v(t_1) = hg_0 + \frac{1}{2}h^2 g_{\phi,0}\gamma_0 + \frac{1}{2}h^2 g_{v,0}g_0 + \frac{1}{2}h^2 g_{t,0} + O\left(h^3\right). \tag{4.10}$$

Comparing (4.7)-(4.8) with (4.9)-(4.10) shows that (4.4) is of order 2. Furthermore, it defines a consistent method and by Theorem 2.2.1 it is zero-stable. Next we prove the convergence of (4.4).

4.4 Convergence

From (4.4) we see that the final approximations are given by

$$\phi_{n+1} = \phi_n + \frac{1}{2}h(\gamma_n + \gamma_{n+1}), \tag{4.11}$$

$$v_{n+1} = v_n + \frac{1}{2}h(g_n + g_{n+1}), \tag{4.12}$$

where

$$\gamma_n \equiv \gamma(\phi_n, v_n, t_n), \qquad g_n \equiv g(\phi_n, v_n, t_n),$$
$$\gamma_{n+1} \equiv \gamma(\phi_{n+1}, v_n + hg_n, t_{n+1}) \qquad g_{n+1} \equiv \gamma(\phi_{n+1}, v_n + hg_n, t_{n+1}).$$

By replacing the approximate values ϕ_n , v_n , ϕ_{n+1}, v_{n+1} by the exact values $\phi(t_n)$, $v(t_n)$, $\phi(t_{n+1})$, $v(t_{n+1})$ and by adding the remainder terms τ^1_{k+1}, τ^2_{k+1} corresponding to $\phi(t_{n+1})$ and $v(t_{n+1})$, respectively, we have

$$\phi(t_{n+1}) = \phi(t_n) + \frac{1}{2}h(\gamma(t_n) + \gamma(t_{n+1})) + \tau^1_{n+1}, \tag{4.13}$$

$$v(t_{n+1}) = v(t_n) + \frac{1}{2}h(g(t_n) + g(t_{n+1})) + \tau^2_{n+1}, \tag{4.14}$$

where

$$\gamma(t_n) \equiv \gamma(\phi(t_n), v(t_n), t_n), \qquad \gamma(t_{n+1}) \equiv \gamma(\phi(t_{n+1}), v(t_n) + hg(t_n), t_{n+1}),$$
$$g(t_n) \equiv g(\phi(t_n), v(t_n), t_n), \qquad g(t_{n+1}) \equiv g(\phi(t_{n+1}), v(t_n) + hg(t_n), t_{n+1}).$$

Subtracting the corresponding equations in (4.11)-(4.12) and (4.13)-(4.14), taking the respective norms[1] and using the notation

$$\begin{aligned} \left\|e_j^1\right\| &= \|\phi_j - \phi(t_j)\|, \\ \left\|e_j^2\right\| &= \|v_j - v(t_j)\|, \quad j = n, n+1. \end{aligned}$$

results in the following set of equations

$$\begin{aligned} \left\|e_{n+1}^1\right\| &\leq \left\|e_n^1\right\| + \frac{1}{2}h\left(\|\gamma_n - \gamma(t_n)\| + \|\gamma_{n+1} - \gamma(t_{n+1})\|\right) + \left\|\tau_{n+1}^1\right\|, \\ \left\|e_{n+1}^2\right\| &\leq \left\|e_n^2\right\| + \frac{1}{2}h\left(\|g_n - g(t_n)\| + \|g_{n+1} - g(t_{n+1})\|\right) + \left\|\tau_{n+1}^2\right\|. \end{aligned}$$

Since the functions γ and g are Lipschitz continuous, they satisfy the inequalities of the form

$$\begin{aligned} \|\gamma(\phi_n, v_n, t_n) - \gamma(\phi(t_n), v(t_n), t_n)\| &\leq L_1\left(\|\phi_n - \phi(t_n)\| + \|v_n - v(t_n)\|\right), \\ \|g(\phi_n, v_n, t_n) - g(\phi(t_n), v(t_n), t_n)\| &\leq L_2\left(\|\phi_n - \phi(t_n)\| + \|v_n - v(t_n)\|\right), \end{aligned}$$

with Lipschitz constants L_1 and L_2. By applying these conditions with $L = \max\{L_1, L_2\}$ on the right hand side of the preceeding set of equations we have from the first inequality

$$\begin{aligned} \left\|e_{n+1}^1\right\| &\leq \left\|e_n^1\right\| + \frac{1}{2}hL\left(\left\|e_n^1\right\| + \left\|e_n^2\right\|\right) + \frac{1}{2}hL\left(\left\|e_{n+1}^1\right\| + \left\|e_n^2\right\| + h\|g_n - g(t_n)\|\right) + \left\|\tau_{n+1}^1\right\| \\ &\leq \left(1 + \frac{1}{2}hL\right)\left\|e_n^1\right\| + hL\left\|e_n^2\right\| + \frac{1}{2}h^2L^2\left(\left\|e_n^1\right\| + \left\|e_n^2\right\|\right) + \frac{1}{2}hL\left\|e_{n+1}^1\right\| + \left\|\tau_{n+1}^1\right\|, \end{aligned}$$

that is,

$$\left\|e_{n+1}^1\right\| \leq \left(1 + \frac{1}{2}hL + \frac{1}{2}h^2L^2\right)\left\|e_n^1\right\| + \left(hL + \frac{1}{2}h^2L^2\right)\left\|e_n^2\right\| + \frac{1}{2}hL\left\|e_{n+1}^1\right\| + \left\|\tau_{n+1}^1\right\|. \quad (4.15)$$

[1]We use the same notation for the norms in different spaces

Similarly, from the second inequality we have

$$\begin{aligned}\left\|e_{n+1}^2\right\| &\leq \left\|e_n^2\right\| + \frac{1}{2}hL\left(\left\|e_n^1\right\| + \left\|e_n^2\right\|\right) + \frac{1}{2}hL\left(\left\|e_{n+1}^1\right\| + \left\|e_n^2\right\| + h\left\|g_n - g\left(t_n\right)\right\|\right) + \left\|\tau_{n+1}^2\right\| \\ &\leq (1+hL)\left\|e_n^2\right\| + \frac{1}{2}hL\left\|e_n^1\right\| + \frac{1}{2}h^2L^2\left(\left\|e_n^1\right\| + \left\|e_n^2\right\|\right) + \frac{1}{2}hL\left\|e_{n+1}^1\right\| + \left\|\tau_{n+1}^2\right\|,\end{aligned}$$

that is,

$$\left\|e_{n+1}^2\right\| \leq \left(1 + hL + \frac{1}{2}h^2L^2\right)\left\|e_n^2\right\| + \left(\frac{1}{2}hL + \frac{1}{2}h^2L^2\right)\left\|e_n^1\right\| + \frac{1}{2}hL\left\|e_{n+1}^1\right\| + \left\|\tau_{n+1}^2\right\|. \quad (4.16)$$

Equations (4.15)-(4.16) can be written in the form

$$\begin{bmatrix} \left\|e_{n+1}^1\right\| \\ \left\|e_{n+1}^2\right\| \end{bmatrix} \leq A(h)\begin{bmatrix} \left\|e_n^1\right\| \\ \left\|e_n^2\right\| \end{bmatrix} + \begin{bmatrix} \frac{1}{2}hL \\ \frac{1}{2}hL \end{bmatrix}\left\|e_{n+1}^1\right\| + \begin{bmatrix} \left\|\tau_{n+1}^1\right\| \\ \left\|\tau_{n+1}^2\right\| \end{bmatrix},$$

where

$$A(h) = \begin{bmatrix} 1 + \frac{1}{2}hL + \frac{1}{2}h^2L^2 & hL + \frac{1}{2}h^2L^2 \\ \frac{1}{2}hL + \frac{1}{2}h^2L^2 & 1 + hL + \frac{1}{2}h^2L^2 \end{bmatrix}.$$

Taking norms on both sides of the preceeding inequality gives

$$\begin{aligned}\left\|e_{n+1}\right\| &\leq \left\|A(h)\right\|\left\|e_n\right\| + \frac{1}{2}hL\left\|e_{n+1}\right\| + \left\|\tau_{n+1}\right\| \\ &\leq \left(1 + \frac{3}{2}hL + h^2L^2\right)\left\|e_n\right\| + \frac{1}{2}hL\left\|e_{n+1}\right\| + \left\|\tau_{n+1}\right\|,\end{aligned}$$

where

$$e_j = \begin{bmatrix} e_j^1 \\ e_j^2 \end{bmatrix}, \quad j = n, n+1,$$

and

$$\tau_{n+1} = \begin{bmatrix} \tau_{n+1}^1 \\ \tau_{n+1}^2 \end{bmatrix}.$$

Hence,

$$\left\|e_{n+1}\right\| \leq \left(\frac{1 + \frac{3}{2}hL + h^2L^2}{1 - \frac{1}{2}hL}\right)\left\|e_n\right\| + \frac{1}{1 - \frac{1}{2}hL}\left\|\tau_{n+1}\right\|. \quad (4.17)$$

Since (4.11)-(4.12) is of order 2, $\tau_{n+1} = O\left(h^3\right)$, that is, there exists a positive constant B such that

$$\left\|\tau_{n+1}\right\| \leq Bh^3, \quad (4.18)$$

and we may write (4.17) as

$$\|e_{n+1}\| \leq R(h)\|e_n\| + S(h), \tag{4.19}$$

where

$$R(h) = \frac{1 + \frac{3}{2}hL + h^2L^2}{1 - \frac{1}{2}hL}, \qquad S(h) = \frac{Bh^3}{1 - \frac{1}{2}hL}.$$

Assuming $e_0 = 0$, that is no initial error, we compare (4.19) with the difference equation

$$E_{n+1} = RE_n + S, \tag{4.20}$$

with initial value E_0. We will seek a solution E_n of (4.20) which will dominate $\|e_n\|$, that is, $\|e_n\| \leq E_n$ for integers $n \geq 0$. Proceeding by induction, we assume $\|e_n\| \leq E_n$ and prove that $\|e_{n+1}\| \leq E_{n+1}$. From (4.19) we have

$$\begin{aligned} \|e_{n+1}\| &\leq R\|e_n\| + S \\ &\leq RE_n + S \\ &= E_{n+1}. \end{aligned}$$

Since $e_0 = E_0$ the induction is complete and guarantees $\|e_n\| \leq E_n$ for all integers $n \geq 0$.

Next we solve (4.20). By direct computation we find

$$\begin{aligned} E_1 &= RE_0 + S = S \\ E_2 &= RE_1 + S = S(R+1) \\ E_3 &= RE_2 + S = S\left(R^2 + R + 1\right) \end{aligned}$$

and in general it follows that

$$\begin{aligned} E_n = RE_{n-1} + S &= S\left(R^{n-1} + R^{n-2} + \ldots + R + 1\right) \\ &= S\left(\frac{R^n - 1}{R - 1}\right); \end{aligned}$$

which is the solution of (4.20) as can be verified by substitution.

Hence,

$$\begin{aligned}\|e_n\| \leq E_n &= \left(\frac{Bh^3}{1-\frac{1}{2}hL}\right)\left(\frac{1-\frac{1}{2}hL}{2hL+h^2L^2}\right)\left[\left(\frac{1+\frac{3}{2}hL+h^2L^2}{1-\frac{1}{2}hL}\right)^n-1\right] \\ &= \left(\frac{Bh^2}{2L\left(1+\frac{1}{2}hL\right)}\right)\left[\left(\frac{1+\frac{3}{2}hL+h^2L^2}{1-\frac{1}{2}hL}\right)^n-1\right].\end{aligned}$$

As $h \to 0$ the first factor tends to zero but we note that $n \to \infty$ as $h \to 0$ in the second factor. To prove convergence we consider the expression in square brackets and take the limit as $h \to 0$ and $n \to \infty$ simultaneously in such a way that $t = nh$ remains fixed;

$$\begin{aligned}\lim_{\substack{h\to 0 \\ t=nh}} \left(\frac{1+\frac{3}{2}hL+h^2L^2}{1-\frac{1}{2}hL}\right)^n &= \lim_{\substack{h\to 0 \\ t=nh}} \exp\left[n\ln\left(\frac{1+\frac{3}{2}hL+h^2L^2}{1-\frac{1}{2}hL}\right)\right] \\ &= \lim_{\substack{h\to 0 \\ t=nh}} \exp\left[\frac{t}{h}\ln\left(\frac{1+\frac{3}{2}hL+h^2L^2}{1-\frac{1}{2}hL}\right)\right] \\ &= \mathrm{e}^{2Lt}\end{aligned}$$

where we have applied L'Hôpital's rule to the expression in square brackets.

Since this holds $\forall t \in [0, a]$ we have

$$\lim_{\substack{h\to 0 \\ t=nh}} z_n = z(t) \qquad \forall t \in [0, a]$$

where $z(t)$ is the solution of (3.21) and the convergence of (4.11)-(4.12) is established.

4.5 Amplitude-Shape Program

We present a computer program, which we call ASP, for performing the integration of the amplitude and shape systems. The program implements the ASNS ((4.3) or (4.4)) and

uses one of three options for adjusting the stepsize during the integration process. The first option uses a fixed stepsize during the integration process. The second option is based on Richardson's extrapolation technique, discussed in Section 2.4, to monitor the error and automatically adjust the stepsize. The formula

$$h_{next} \cong 0.9 \left(\frac{\epsilon}{E}\right)^{\frac{1}{3}} h, \tag{4.21}$$

is used to adjust the stepsize, where ϵ is the user defined accuracy and E is the estimate of the local error. If $|E| \leq \epsilon$ then (4.21) is used to estimate the next stepsize. On the other hand, if $|E| > \epsilon$ then h is reduced and (4.21) is used to obtain another estimate of h_{next}. As a measure of precaution, we multiply by a safety factor 0.9, which is slightly smaller then unity, since the error estimate is only accurate to the third order in h. Following (2.29) we use

$$z(t+h) = z\left(t, \frac{h}{2}\right) + \frac{E}{15}, \tag{4.22}$$

as the improved approximation of the solution at each step.

The third option also adjusts the stepsize automatically. It is based on polynomial extrapolation and uses the stepsize monitor discussed in Section 2.4.

For simplicitly we restrict the number of amplitude equations to two. A description of the numerical programming strategy is outlined and a set of algorithms are provided. The programs are included in the appendix. At each step the amplitude equations are solved implicitly by means of Newton iteration and the shape equations solved explicitly. In the Newton iteration the Jacobi matrix is generated by finite differences and the matrix inversion is performed directly. The overall structure and calling sequence of the routines used in ASP is illustrated in Fig. (4.1). ASP is the main program in which

(i) the user defined parameters (e.g. the number of amplitude and shape equations,

error tolerance, initial time step, etc) are set,

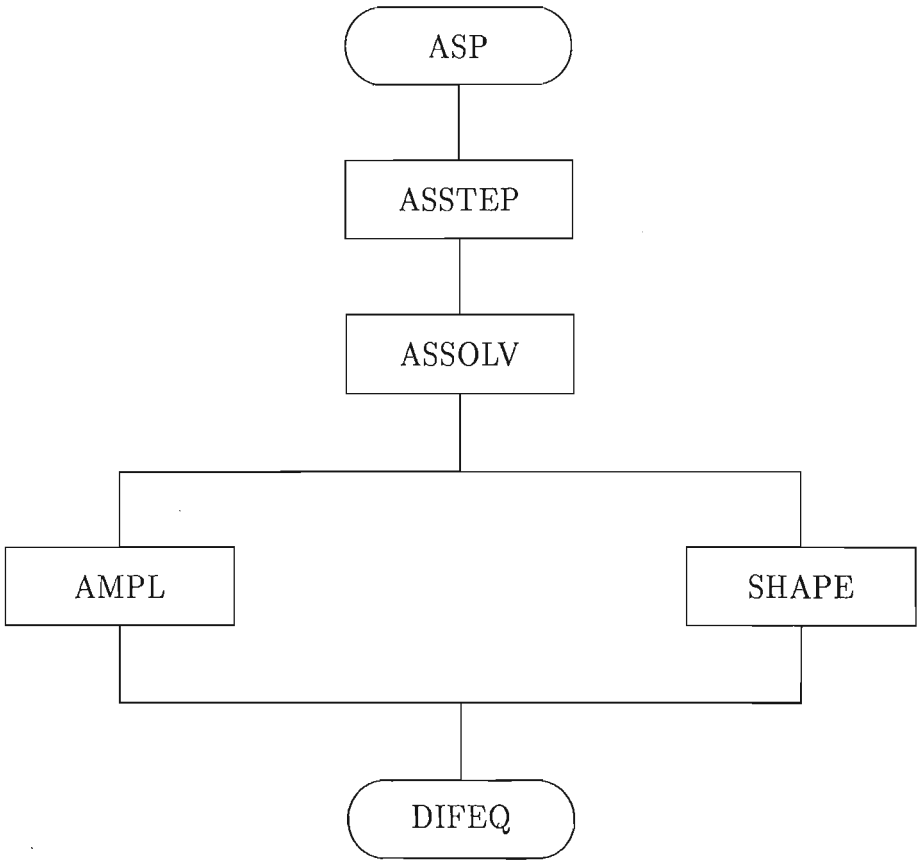

Figure 4.1: Structure of ASP: ASP and DIFEQ are the user-defined routines.

(ii) the weight vectors ω^k are evaluated,

(iii) the initial conditions for the amplitude and shape systems computed, and

(iv) the computing time calculated and output of results given.

The routines ASSTEP, ASSOLV, AMPL and SHAPE form a built-in component of the ASP. A description of these routines together with their algorithms are given below.

ASSTEP[2] is used to advance the solution, estimate the error and adjust the stepsize in the integration process. This is done by making a call to the core integrator routine *assolv*. At the start of the integration step the following are input: time t; arrays fi and v holding the vectors ϕ_n and v_n, respectively; the stepsize h to be attempted; the error tolerance *eps*; and the array *zscl* against which the error of the solution is scaled. At the end of the integration step the arrays fi and v are replaced by the improved solution values (4.22) and the estimated next stepsize, *hnext*, returned. The arrays *fisav*, *vsav*, *fitemp*, and *vtemp* are used for temporary storage and *rnorm* is a user defined function for evaluating the maximum norm.

ASSOLV[3] implements the ASNS for performing the actual integration of the amplitude and shape systems (4.1)-(4.2). The variables *ts* and h represent the time and stepsize to be tried at the start of the interval. The arrays fi and v hold the vectors ϕ_n and v_n when the algorithm is called, and are replaced by ϕ_{n+1}, v_{n+1} on return. The arrays *dfi* and *dv* hold the derivatives of fi and v and are supplied by calling the routines *ampl* and *shape*. The parameters *Ftol*, *xtol*, *delta*, and *maxits* and the arrays F, *rp*, and *dx* are used in the Newton iteration (refer to programs in appendix). The arrays *fisav*, *vsav*, *dfisav*, and *dvsav* are used for temporary storage and the function *norm* has the same meaning as above.

[2]The stepsize strategy described in the algorithm in Figure 4.2 is for Richardson's extrapolation.

[3]The algorithm in Figure 4.3 implements (4.4).

Finally, the routines AMPL and SHAPE are used to compute the derivatives of fi and v, respectively. In both algorithms, input at $t = t_n$ are the arrays fi, and v holding the vectors ϕ_n and v_n, respectively. The array w stores the weight vectors and the array x contains the solution vector. Whenever required a call is made to a user supplied routine DIFEQ to supply the derivative, dx, of x.

In the next chapter we solve various examples by using the amplitude-shape numerical procedure described above.

```
tsav := t;
for i := 1 to 2 do fisav[i] := fi[i];
for j := 1 to m do vsav[j] := v[j];
901 :
hh = 0.5 * h;
assolv(t, hh, fi, v);
t = tsav + hh;
assolv(t, hh, fi, v);
for i := 1 to 2 do
begin
   temp[i] := fisav[i];
end
for j := 1 to m do
begin
   vtemp[j] := vsav[j];
end
assolv(tsav, h, fitemp, vtemp);
for i := 1 to 2 do fitemp[i] := fi[i] - fitemp[i];
for j := 1 to m do vtemp[j] := v[j] - vtemp[j];
rnorm1 := rnorm(fiscl, fitemp);
rnorm2 := rnorm(vscl, vtemp);
err := max(rnorm1, rnorm2)/eps;
if err <= eps then
begin
   hnext := 0.9 * exp((1./3.) * ln(eps/err)) * h;
   for i := 1 to 2 do fi[i] := fi[i] + fitemp[i]/3.;
   for j := 1 to m do v[j] := v[j] + vtemp[j]/3.
end  else
   begin
      h := 0.9 * exp((1./3.) * ln(eps/err)) * h;
      t := tsav;
      for i := 1 to 2 do fi[i] := fisav[i];
      for j := 1 to m do v[j] := vsav[j];
       goto 901
   end
end
```

Figure 4.2: Algorithm : ASSTEP

```
tsav := t;
for i := 1 to 2 do fisav[i] := fi[i];
for j := 1 to m do vsav[j] := v[j];
shape(tsav, fisav, vsav, dvsav);
for j := 1 to m do v[j] := vsav[j] + h * dvsav[j];
ampl(tsav, fisav, vsav, dfisav);
for i := 1 to 2 do fi[i] := fisav[i] + h * dfisav[i];
tsav := tsav + h;
repeat
   begin
      ampl(tsav, fi, v, dfi);
      for i := 1 to 2 do F[i] = -fi[i] + fisav[i] + 0.5 * h * (dfisav[i] + dfi[i]);
      if norm(F) < Ftol then  goto 911
      for i := 1 to 2 do
      begin
         fisav[i] := fi[i];    dfisav[i] := dfi[i]
      end
      for k := 1 to 2 do
      begin
         fi[k] := fi[k] + delta;
         ampl(tsav, fi, v, dfi);
         for i := 1 to 2 do
         if i = k then rp(i, k) := 1.0 - 0.5 * h * (dfi[i] - dfisav[i])/delta else
            rp(i, k) := -0.5 * h * (dfi[i] - dfisav[i])/delta
         fi[k] := fisav[k];
      end
      for i := 1 to 2 do dfi[i] := dfisav[i];
      dx[1] := (F[1] * rp[2,2] - F[2] * rp[1,2])/(rp[1,1] * rp[2,2] - rp[1,2] * rp[2,1]);
      dx[2] := (F[2] * rp[1,1] - F[1] * rp[2,1])/(rp[1,1] * rp[2,2] - rp[1,2] * rp[2,1]);
      for 1 := 1 to 2 do fi[i] := fi[i] + dx[i];
      if norm(dx) < xtol then  goto 911
   end
until iter = maxits;
911 :
shape(tsav, fi, v, dv);
for j := 1 to m do v[j] := vsav[j] + 0.5 * (dvsav[j] + dv[j]);
```

Figure 4.3: Algorithm : ASSOLV

```
for i := 1 to 2 do
begin
   l1 := (i − 1) ∗ m + 1;
   l2 := i ∗ m;
   for j := l1 to l2 do x[j] := fi[i] ∗ v[j];
end
difeq(t, x, dx);
for i := 1 to 2 do
begin
   l1 := (i − 1) ∗ m + 1;
   l2 := i ∗ m;
   dfi[i] := 0.;
   for j := l1 to l2 do dfi[i] := dfi[i] + w[j] ∗ dx[j];
end
```

Figure 4.4: Algorithm : AMPL

```
for i := 1 to 2 do
begin
   l1 := (i - 1) * m + 1;
   l2 := i * m;
   for j := l1 to l2 do x[j] := fi[i] * v[j];
end
difeq(t, x, dx);
for i := 1 to 2 do
begin
   l1 := (i - 1) * m + 1;
   l2 := i * m;
   finv := 1./fi[i];
   s := 0.;
   for j := l1 to l2 do s := s + w[j] * dx[j];
   for j := l1 to l2 do dv[j] := finv * (dx[j] - s * v[j]);
end
```

Figure 4.5: Algorithm : SHAPE

Chapter 5

Numerical Experiments

To evaluate the performance of the ASM we present six examples which will be solved with ASP. The first three examples are based on the simple one-dimensional linear diffusion equation and variations of it. These examples serve to demonstrate the applicability of the ASM. Thereafter, we consider two systems of quasilinear diffusion equations describing the spatial effects in chemical kinetics. These examples are chosen as representative examples of the class of problems (2.83) in Section 2.7. The final example describes a system of nonlinear PDEs whose exact solution is known. This example demonstrates the robustness of the ASM. In all the examples we choose the components of the weight vectors as follows. If the component of the solution vector is positive then we choose the corresponding component in the weight vector as equal to one. On the other hand, if the component of the solution vector is negative then we choose the corresponding component in the weight vector as equal to negative one.

All calculations were performed on a PC/Pentium (75MHz) using Fortran double precision arithmetic. To facilitate the referencing of the programs we adopt the following nomenclature. We use ASP1 and ASP2 to refer to the programs which implement (4.3) and (4.4),

respectively. This is followed by the use of the numbers 0, 1, 2 to indicate the type of stepsize selection strategy used as follows :

0 - fixed stepsize,

1 - variable stepsize based on Richardson's extrapolation,

2 - variable stepsize based on polynomial extrapolation.

Thus, for example, ASP10 refers to the program which implements (4.3) and uses a fixed stepsize, etc.

In the examples below, we use the method of lines discussed in Section 2.5 to discretize the PDEs.

Example 1

We consider the linear one-dimensional diffusion equation

$$\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial \rho^2}, \qquad 0 \leq \rho \leq 1, \tag{5.1}$$

with boundary conditions

$$u(0,t) = u(1,t) = 0, \tag{5.2}$$

and initial conditions

$$u(\rho,0) = \sin(\pi\rho). \tag{5.3}$$

The exact solution is

$$u(\rho,t) = e^{-(\pi\alpha)^2 t} \sin(\pi\rho) \tag{5.4}$$

We solved this problem with a uniform mesh of 51 points and $\alpha = 1$. The results using ASP10 and ASP20 with $h = 1 \times 10^{-4}$ and the maximum relative error $\|e^1\|_\infty$, are shown in Tables 5.1-5.2. The solution is compared with the exact solution. Figures 5.1-5.3 illustrates the exact solution, amplitude solution and the shape solution, respectively.

Diffusion Equation						
		EXACT	ASP10			
ρ	t	$x^1 \times 10^1$	$\phi^1 \times 10^{-1}$	$v^1 \times 10^2$	$x^1 \times 10^1$	$\|e^1\|_\infty \times 10^{-3}$
0.00	0.00	0.000000	3.182	0.000	0.000000	0.0000
	1.00	0.000000	0.000164	0.000	0.000000	1.6227
	2.00	0.000000	0.000000	0.000	0.000000	3.2427
0.20	0.00	5.877853	3.182	1.847	5.877853	0.0000
	1.00	0.000304	0.000164	1.847	0.000303	1.6227
	2.00	0.000000	0.000000	1.847	0.000000	3.2427
0.40	0.00	9.510565	3.182	2.989	9.510565	0.0000
	1.00	0.000491	0.000164	2.989	0.000490	1.6227
	2.00	0.000000	0.000000	2.989	0.000000	3.2427
0.60	0.00	9.510565	3.182	2.989	9.510565	0.0000
	1.00	0.000491	0.000164	2.989	0.000490	1.6227
	2.00	0.000000	0.000000	2.989	0.000000	3.2427
0.80	0.00	5.877853	3.182	1.847	5.877853	0.0000
	1.00	0.000304	0.000164	1.847	0.000303	1.6227
	2.00	0.000000	0.000000	1.847	0.000000	3.2427
1.00	0.00	0.000000	3.182	0.000	0.000000	0.0000
	1.00	0.000000	0.000164	0.000	0.000000	1.6227
	2.00	0.000000	0.000000	0.000	0.000000	3.2427

Table 5.1: [ASP10] Numerical Results for Example 1.

Diffusion Equation						
		EXACT	ASP20			
ρ	t	$x^1 \times 10^1$	$\phi^1 \times 10^{-1}$	$v^1 \times 10^2$	$x^1 \times 10^1$	$\|e^1\|_\infty \times 10^{-3}$
0.00	0.00	0.000000	3.182	0.000	0.000000	0.0000
	1.00	0.000000	0.000165	0.000	0.000000	3.2489
	2.00	0.000000	0.000000	0.000	0.000000	6.5088
0.20	0.00	5.877853	3.182	1.847	5.877853	0.0000
	1.00	0.000304	0.000165	1.847	0.000304	3.2489
	2.00	0.000000	0.000000	1.847	0.000000	6.5088
0.40	0.00	9.510565	3.182	2.989	9.510565	0.0000
	1.00	0.000491	0.000165	2.989	0.000493	1.6227
	2.00	0.000000	0.000000	2.989	0.000000	3.2427
0.60	0.00	9.510565	3.182	2.989	9.510565	0.0000
	1.00	0.000491	0.000165	2.989	0.000493	1.6227
	2.00	0.000000	0.000000	2.989	0.000000	3.2427
0.80	0.00	5.877853	3.182	1.847	5.877853	0.0000
	1.00	0.000304	0.000165	1.847	0.000304	3.2489
	2.00	0.000000	0.000000	1.847	0.000000	6.5088
1.00	0.00	0.000000	3.182	0.000	0.000000	0.0000
	1.00	0.000000	0.000165	0.000	0.000000	3.2489
	2.00	0.000000	0.000000	0.000	0.000000	6.5088

Table 5.2: [ASP20] Numerical Results for Example 1.

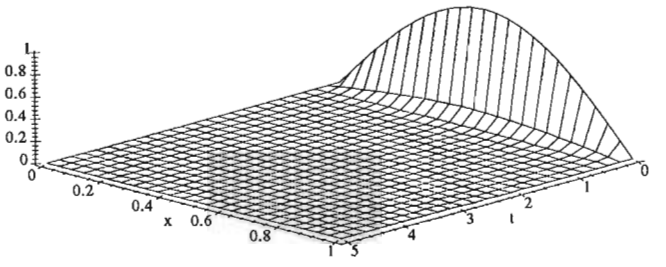

Figure 5.1: 3D-Plot: Exact Solution of Example 1.

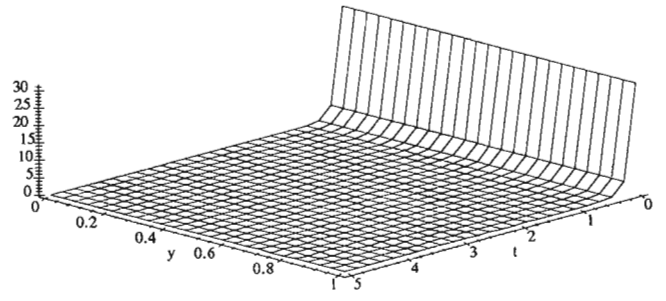

Figure 5.2: 3D-Plot: Amplitude Solution of Example 1.

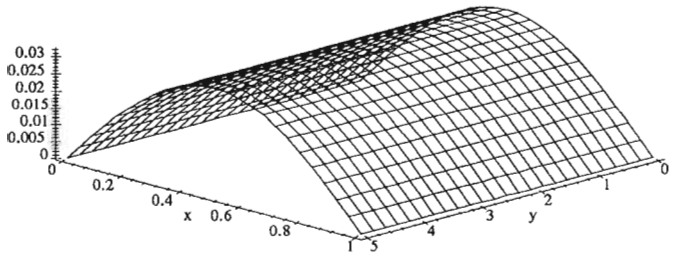

Figure 5.3: 3D-Plot: Shape Solution of Example 1.

Example 2

We consider (5.1) with a reaction term

$$\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial \rho^2} - \beta u, \qquad 0 \leq \rho \leq 1, \tag{5.5}$$

with the boundary conditions

$$u(0,t) = u(1,t) = 0, \tag{5.6}$$

and the initial condition

$$u(\rho,0) = \sin(\pi\rho). \tag{5.7}$$

The exact solution is

$$e^{-(\beta+\pi^2\alpha^2)t} \sin(\pi\rho). \tag{5.8}$$

We solved this problem with $\alpha = \beta = 1$ in exactly the same way as the previous example. The results are shown in Tables 5.3-5.4. Figures 5.4-5.6 depicts the exact solution, amplitude solution and shape solution, respectively.

Reaction-Diffusion Equation						
		EXACT	ASP10			
ρ	t	$x^1 \times 10^1$	$\phi^1 \times 10^{-1}$	$v^1 \times 10^2$	$x^1 \times 10^1$	$\|e^1\|_\infty \times 10^{-3}$
-1.00	0.00	0.000000	3.179	0.000	0.000000	0.0000
	0.20	0.000000	0.3617	0.000	0.000000	1.4184
	0.50	0.000000	0.01390	0.000	0.000000	3.5487
	0.80	0.000000	0.000534	0.000	0.000000	5.6835
	1.00	0.000000	0.000060	0.000	0.000000	7.1093
-0.60	0.00	-9.510565	3.179	-2.992	-9.510565	0.0000
	0.20	-1.081644	0.3617	-2.992	-1.082002	1.4184
	0.50	-0.041486	0.01390	-2.992	-0.041588	3.5487
	0.80	-0.001591	0.000534	-2.992	-0.001598	5.6835
	1.00	-0.000180	0.000060	-2.992	-0.000182	7.1093
0.60	0.00	9.510565	3.179	2.992	9.510565	0.0000
	0.20	1.081644	0.3617	2.992	1.082002	1.4184
	0.50	0.041486	0.01390	2.992	0.041588	3.5487
	0.80	0.001591	0.000534	2.992	0.001598	5.6835
	1.00	0.000180	0.000060	2.992	0.000182	7.1093
1.00	0.00	0.000000	3.179	0.000	0.000000	0.0000
	0.20	0.000000	0.3617	0.000	0.000000	1.4184
	0.50	0.000000	0.01390	0.000	0.000000	3.5487
	0.80	0.000000	0.000534	0.000	0.000000	5.6835
	1.00	0.000000	0.000060	0.000	0.000000	7.1093

Table 5.3: [ASP10] Numerical Results for Example 2.

Reaction-Diffusion Equation						
		EXACT	ASP20			
ρ	t	$x^1 \times 10^1$	$\phi^1 \times 10^{-1}$	$v^1 \times 10^2$	$x^1 \times 10^1$	$\|e^1\|_\infty \times 10^{-2}$
-1.00	0.00	0.000000	3.179	0.000	0.000000	0.0000
	0.20	0.000000	0.3621	0.000	0.000000	0.2601
	0.50	0.000000	0.01394	0.000	0.000000	0.6514
	0.80	0.000000	0.000536	0.000	0.000000	1.0441
	1.00	0.000000	0.000061	0.000	0.000000	1.3069
-0.60	0.00	-9.510565	3.179	-2.992	-9.510565	0.0000
	0.20	-1.081644	0.3621	-2.992	-1.083280	0.2601
	0.50	-0.041486	0.01394	-2.992	-0.041710	0.6514
	0.80	-0.001591	0.000536	-2.992	-0.001606	1.0441
	1.00	-0.000180	0.000061	-2.992	-0.000183	1.3069
0.60	0.00	9.510565	3.179	2.992	9.510565	0.0000
	0.20	1.081644	0.3621	2.992	1.083280	0.2601
	0.50	0.041486	0.01394	2.992	0.041710	0.6514
	0.80	0.001591	0.000536	2.992	0.001606	1.0441
	1.00	0.000180	0.000061	2.992	0.000183	1.3069
1.00	0.00	0.000000	3.179	0.000	0.000000	0.0000
	0.20	0.000000	0.3621	0.000	0.000000	0.2601
	0.50	0.000000	0.01394	0.000	0.000000	0.6514
	0.80	0.000000	0.000536	0.000	0.000000	1.0441
	1.00	0.000000	0.000061	0.000	0.000000	1.3069

Table 5.4: [ASP20] Numerical Results for Example 2.

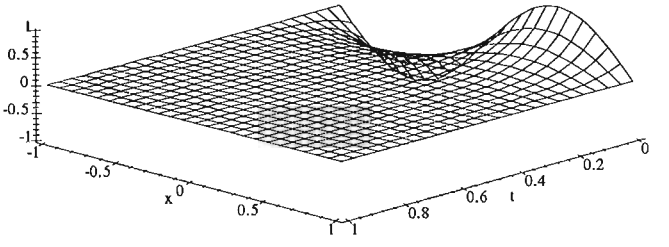

Figure 5.4: 3D-Plot: Exact Solution of Example 2.

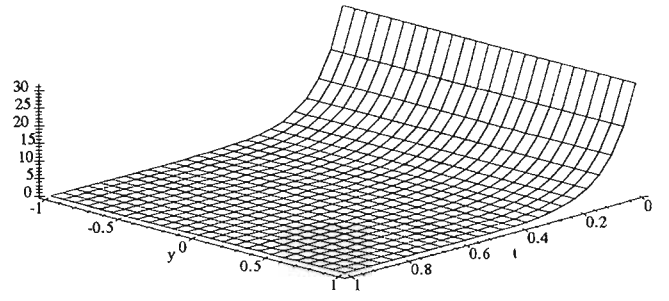

Figure 5.5: 3D-Plot: Amplitude Solution of Example 2.

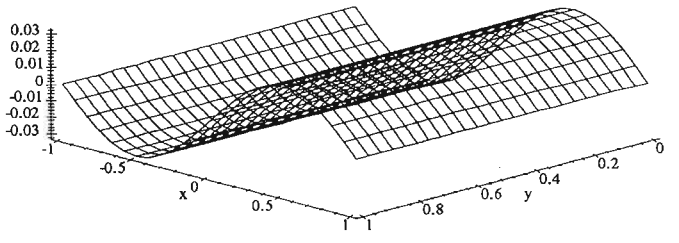

Figure 5.6: 3D-Plot: Shape Solution of Example 2.

Example 3

We consider (5.1) with a convection term

$$\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial \rho^2} - \nu \frac{\partial u}{\partial \rho}, \qquad 0 \leq \rho \leq 1, \tag{5.9}$$

with the boundary conditions

$$u(0,t) = u(1,t) = 0, \tag{5.10}$$

and the initial condition

$$u(\rho,0) = e^{\frac{\rho}{2}} \sin(\pi\rho). \tag{5.11}$$

The exact solution is

$$u(\rho,t) = e^{-\left(\frac{\nu^2}{4\alpha^2}+\pi^2\alpha^2\right)t} e^{\frac{\nu}{2\alpha^2}\rho} \sin(\pi\rho). \tag{5.12}$$

We solved this problem with $\alpha = \nu = 1$ in exactly the same way as the previous two examples. The results are shown in Tables 5.5-5.6. Figures 5.7-5.9 shows the exact solution, amplitude solution and the shape solution, respectively.

As expected the ASM is well suited for the solution of the above examples. The 3D plots demonstrate the advantage of using the ASM. In particular, Figures 5.3-5.9 illustrate a relatively smooth shape solution as compared to the exact solution, in Figures 5.1-5.7. This means when integrating along the t axis, the integration of the shape system is clearly favored to the integration of the original system.

Convection-Diffusion Equation						
		EXACT	ASP10			
ρ	t	$x^1 \times 10^1$	$\phi^1 \times 10^{-1}$	$v^1 \times 10^2$	$x^1 \times 10^1$	$\|e^1\|_\infty \times 10^{-3}$
0.00	0.00	0.000000	4.110	0.000	0.000000	0.0000
	0.20	0.000000	0.5424	0.000	0.000000	0.2853
	0.50	0.000000	0.02604	0.000	0.000000	0.7010
	0.80	0.000000	0.001250	0.000	0.000000	1.1166
	1.00	0.000000	0.0001652	0.000	0.000000	1.3935
0.20	0.00	6.496032	4.110	1.581	6.496032	0.0000
	0.20	0.858362	0.5424	1.581	0.857251	0.2853
	0.50	0.041229	0.02604	1.581	0.041158	0.7010
	0.80	0.001980	0.001250	1.581	0.001976	1.1166
	1.00	0.000261	0.0001652	1.581	0.000261	1.3935
0.80	0.00	8.768726	4.110	2.133	8.768726	0.0000
	0.20	1.158667	0.5424	2.133	1.157180	0.2853
	0.50	0.055653	0.02604	2.133	0.055558	0.7010
	0.80	0.002673	0.001250	2.133	0.002667	1.1166
	1.00	0.000353	0.0001652	2.133	0.000352	1.3935
1.00	0.00	0.000000	4.110	0.000	0.000000	0.0000
	0.20	0.000000	0.5424	0.000	0.000000	0.2853
	0.50	0.000000	0.02604	0.000	0.000000	0.7010
	0.80	0.000000	0.001250	0.000	0.000000	1.1166
	1.00	0.000000	0.0001652	0.000	0.000000	1.3935

Table 5.5: [ASP10] Numerical Results for Example 3

Convection-Diffusion Equation						
		EXACT	ASP20			
ρ	t	$x^1 \times 10^1$	$\phi^1 \times 10^{-1}$	$v^1 \times 10^2$	$x^1 \times 10^1$	$\|e^1\|_\infty \times 10^{-3}$
0.00	0.00	0.000000	4.110	0.000	0.000000	0.0000
	0.20	0.000000	0.5427	0.000	0.000000	0.2907
	0.50	0.000000	0.02606	0.000	0.000000	0.2266
	0.80	0.000000	0.001252	0.000	0.000000	0.0016
	1.00	0.000000	0.0001654	0.000	0.000000	0.1753
0.20	0.00	6.496032	4.110	1.581	6.496032	0.0000
	0.20	0.858362	0.5427	1.581	0.857732	0.2907
	0.50	0.041229	0.02606	1.581	0.041196	0.2266
	0.80	0.001980	0.001252	1.581	0.001978	0.0016
	1.00	0.000261	0.0001654	1.581	0.000261	0.1753
0.80	0.00	8.768726	4.110	2.133	8.768726	0.0000
	0.20	1.158667	0.5427	2.133	1.157829	0.2907
	0.50	0.055653	0.02606	2.133	0.055609	0.2266
	0.80	0.002673	0.001252	2.133	0.002670	0.0016
	1.00	0.000353	0.0001654	2.133	0.000352	0.1753
1.00	0.00	0.000000	4.110	0.000	0.000000	0.0000
	0.20	0.000000	0.5427	0.000	0.000000	0.2907
	0.50	0.000000	0.02606	0.000	0.000000	0.2266
	0.80	0.000000	0.001252	0.000	0.000000	0.0016
	1.00	0.000000	0.0001654	0.000	0.000000	0.1753

Table 5.6: [ASP20] Numerical Results for Example 3

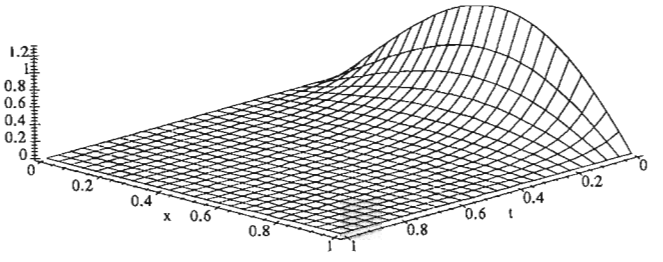

Figure 5.7: 3D-Plot: Exact Solution of Example3.

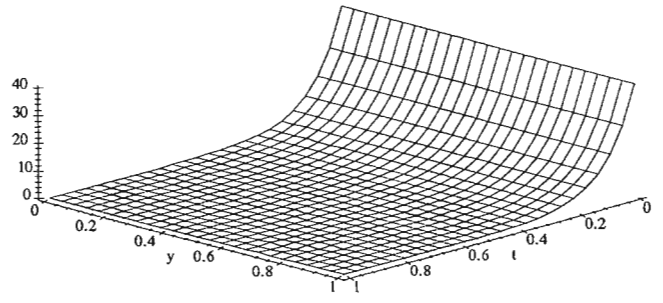

Figure 5.8: 3D-Plot: Amplitude Solution of Example3.

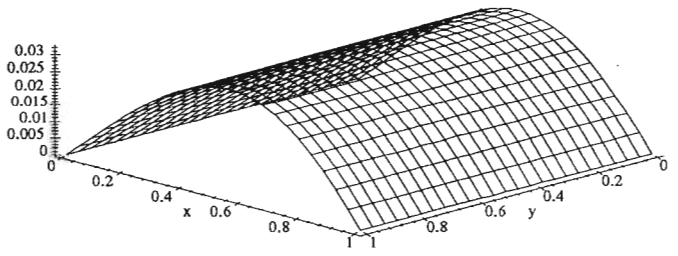

Figure 5.9: 3D-Plot: Shape Solution of Example3.

Next we consider two examples from chemical kinetics. We used ASP21 and ASP22 to solve these problems. To check the efficiency of these programs we compared them with Gear's Solver used to solve the original system of ODEs .

Example 4

We consider the system of equations [2]

$$\frac{\partial c^{(k)}}{\partial t} = \frac{\partial}{\partial \rho}\left[K(z)\frac{\partial c^{(k)}}{\partial \rho}\right] + R^k\left(c^{(1)}, c^{(2)}, t\right) \quad k = 1, 2.$$

where $c^{(1)}$ is the concentration of the oxygen singlet O_1 and $c^{(2)}$ is the concentration of ozone O_3, ρ denotes the elevation above the earth in kilometers ($30 \leq \rho \leq 50$), $K(z) = e^{\frac{z}{5}} 10^{-8}$ and $R^{(k)}$ are the reaction terms given by

$$\begin{aligned} R^1\left(c^{(1)}, c^{(2)}, t\right) &= -k_1 c c^{(1)} - k_2 c^{(1)} c^{(2)} + 2 c k_3(t) + k_4(t) c^{(2)}, \\ R^2\left(c^{(1)}, c^{(2)}, t\right) &= k_1 c c^{(1)} - k_2 c^{(1)} c^{(2)} - k_4(t) c^{(2)}, \end{aligned}$$

with $c = 3.7 \times 10^{16}$, $k_1 = 1.63 \times 10^{-16}$, $k_2 = 4.66 \times 10^{-16}$. The functions k_3 and k_4 are given by the formula

$$k_j = \left\{ \begin{array}{ll} \exp^{-(a_j/\sin \omega t)}, & \sin \omega t > 0 \\ 0, & \sin \omega t \leq 0 \end{array} \right\}, \qquad j = 3, 4,$$

where $a_3 = 22.62$, $a_4 = 7.601$ and $\omega = \pi/43200$.

The initial conditions are

$$c^{(1)}(\rho, 0) = 10^6 \gamma(\rho), \qquad c^{(2)}(\rho, 0) = 10^{12} \gamma(\rho),$$

where

$$\gamma(\rho) = 1 - \left(\frac{\rho - 40}{10}\right)^2 + \frac{1}{2}\left(\frac{\rho - 40}{10}\right)^4,$$

and the boundary conditions are

$$\frac{\partial c^{(k)}}{\partial \rho}(30, t) = \frac{\partial c^{(k)}}{\partial \rho}(50, t) = 0, \qquad k = 1, 2.$$

We solved this problem with a uniform mesh of 100 points which resulted in a system of 200 ordinary differential equations. The initial time step and the error tolerance were both set equal to 10^{-4} in ASP21 and ASP22. The error tolerance in the implicit Gear Solver (IGS) was also set equal to 10^{-4}. The reference solution was obtained with the IGS and the error tolerance of 10^{-8}. The programmes were allowed to run until the final value of $t = 1s$ was reached. The results for the solution subvectors $c^{(1)}$ and $c^{(2)}$ at various spatial points and output times are displayed in Tables 5.7-5.10

Two Species Diffusion - 1D System						
		REF	IGS	ASP21		
z	t	$x^1 \times 10^{-5}$	$x^1 \times 10^{-5}$	$\phi^1 \times 10^{-7}$	$v^1 \times 10^2$	$x^1 \times 10^{-5}$
30.00	0.00	5.000000	5.000000	7.63999	0.6545	5.000000
	1.00	0.012012	0.011733	0.01835	0.6545	0.012012
	2.00	0.000028	0.000052	0.00004	0.6545	0.000028
	3.00	0.000000	0.000000	0.00000	0.6545	0.000000
34.84	0.00	7.698325	7.698325	7.63999	1.008	7.698325
	1.00	0.018493	0.018063	0.01835	1.008	0.018493
	2.00	0.000044	0.000042	0.00004	1.008	0.000044
	3.00	0.000000	0.000000	0.00000	1.008	0.000000
44.94	0.00	7.850314	7.850314	7.63999	1.028	7.850314
	1.00	0.018858	0.018419	0.01835	1.028	0.018858
	2.00	0.000045	0.000082	0.00004	1.028	0.000045
	3.00	0.000000	0.000000	0.00000	1.028	0.000000
50.00	0.00	5.000000	5.000000	7.63999	0.6545	5.000000
	1.00	0.012013	0.017340	0.01835	0.6545	0.012013
	2.00	0.000028	0.000052	0.00004	0.6545	0.000028
	3.00	0.000000	0.000000	0.00000	0.6545	0.000000
CPU TIME			47			11

Table 5.7: [ASP21] Numerical Results for Example 4. (c^1 vector)

Two Species Diffusion - 1D System						
		REF	IGS	ASP21		
z	t	$x^1 \times 10^{-11}$	$x^1 \times 10^{-11}$	$\phi^1 \times 10^{-13}$	$v^1 \times 10^2$	$x^1 \times 10^{-11}$
30.00	0.00	5.000000	5.000000	7.640	0.6545	5.000000
	1.00	5.000008	5.000008	7.640	0.6545	5.000008
	2.00	5.000012	5.000012	7.640	0.6545	5.000012
	3.00	5.000015	5.000015	7.640	0.6545	5.000015
34.84	0.00	7.698325	7.698325	7.640	1.008	7.698325
	1.00	7.698334	7.698334	7.640	1.008	7.698334
	2.00	7.698335	7.698335	7.640	1.008	7.698335
	3.00	7.698336	7.698336	7.640	1.008	7.698336
44.94	0.00	7.850314	7.850314	7.640	1.028	7.850314
	1.00	7.850305	7.850305	7.640	1.028	7.850305
	2.00	7.850288	7.850288	7.640	1.028	7.850288
	3.00	7.850271	7.850271	7.640	1.028	7.850271
50.00	0.00	5.000000	5.000000	7.640	0.6545	5.000000
	1.00	5.000184	5.000184	7.640	0.6545	5.000184
	2.00	5.000360	5.000360	7.640	0.6545	5.000360
	3.00	5.000535	5.000534	7.640	0.6545	5.000535
CPU TIME			47			11

Table 5.8: [ASP21] Numerical Results for Example 4. (c^2 vector)

Two Species Diffusion - 1D System						
		REF	IGS	ASP22		
z	t	$x^1 \times 10^{-5}$	$x^1 \times 10^{-5}$	$\phi^1 \times 10^{-7}$	$v^1 \times 10^2$	$x^1 \times 10^{-5}$
30.00	0.00	5.000000	5.000000	7.63999	0.6545	5.000000
	1.00	0.012012	0.011733	0.01835	0.6545	0.012010
	2.00	0.000028	0.000052	0.00004	0.6545	0.000028
	3.00	0.000000	0.000000	0.00000	0.6545	0.000000
34.84	0.00	7.698325	7.698325	7.63999	1.008	7.698325
	1.00	0.018493	0.018063	0.01835	1.008	0.018496
	2.00	0.000044	0.000042	0.00004	1.008	0.000044
	3.00	0.000000	0.000000	0.00000	1.008	0.000000
44.94	0.00	7.850314	7.850314	7.63999	1.028	7.850314
	1.00	0.018858	0.018419	0.01835	1.028	0.018854
	2.00	0.000045	0.000082	0.00004	1.028	0.000045
	3.00	0.000000	0.000000	0.00000	1.028	0.000000
50.00	0.00	5.000000	5.000000	7.63999	0.6545	5.000000
	1.00	0.012013	0.017340	0.01835	0.6545	0.012010
	2.00	0.000028	0.000052	0.00004	0.6545	0.000028
	3.00	0.000000	0.000000	0.00000	0.6545	0.000000
CPU TIME			47			17

Table 5.9: [ASP22] Numerical Results for Example 4. (c^1 vector)

Two Species Diffusion - 1D System						
		REF	IGS	ASP22		
z	t	$x^2 \times 10^{-11}$	$x^2 \times 10^{-11}$	$\phi^2 \times 10^{-13}$	$v^2 \times 10^2$	$x^2 \times 10^{-11}$
30.00	0.00	5.000000	5.000000	7.640	0.6545	5.000000
	1.00	5.000008	5.000008	7.640	0.6545	5.000008
	2.00	5.000012	5.000012	7.640	0.6545	5.000012
	3.00	5.000015	5.000015	7.640	0.6545	5.000015
34.84	0.00	7.698325	7.698325	7.640	1.008	7.698325
	1.00	7.698334	7.698334	7.640	1.008	7.698334
	2.00	7.698335	7.698335	7.640	1.008	7.698335
	3.00	7.698336	7.698336	7.640	1.008	7.698336
44.94	0.00	7.850314	7.850314	7.640	1.028	7.850314
	1.00	7.850305	7.850305	7.640	1.028	7.850305
	2.00	7.850288	7.850288	7.640	1.028	7.850288
	3.00	7.850271	7.850271	7.640	1.028	7.850271
50.00	0.00	5.000000	5.000000	7.640	0.6545	5.000000
	1.00	5.000184	5.000184	7.640	0.6545	5.000184
	2.00	5.000360	5.000360	7.640	0.6545	5.000360
	3.00	5.000535	5.000534	7.640	0.6545	5.000535
CPU TIME			47			17

Table 5.10: [ASP22] Numerical Results for Example 4. (c^2 vector)

Example 5

We consider the pair of coupled reaction-diffusion equations involving two chemical species undergoing diurnal kinetics and transport in two dimensions [2]

$$\frac{\partial c^{(k)}}{\partial t} = K_h \frac{\partial^2 c^{(k)}}{\partial z^2} + \frac{\partial}{\partial \rho}\left(K_v(\rho)\frac{\partial c^{(k)}}{\partial \rho}\right) + R^k\left(c^{(1)}, c^{(2)}, t\right), \quad k = 1, 2,$$

with the diffusion coefficients given as

$$K_h = 4 \times 10^{-6}, \quad K_v(\rho) = 10^{-8}\exp^{\rho/5}.$$

As in the previous example $c^{(1)}$ is the concentration of the oxygen singlet O_1 and $c^{(2)}$ the concentration of ozone O_3, depending now on two spatial variables: ρ - the elevation above the earth ($30 \leq \rho \leq 50$) and z - the horizontal position ($0 \leq z \leq 20$), both in kilometers.

The reaction terms R^k are the same as in the previous example. The initial conditions are given by polynomials that are slightly peaked in the centre and consistent with the boundary conditions

$$\begin{aligned} c^{(1)}(\rho, z, 0) &= 10^6 \alpha(\rho)\beta(z), \\ c^{(2)}(\rho, z, 0) &= 10^{12}\alpha(\rho)\beta(z), \end{aligned}$$

where

$$\alpha(\rho) \equiv 1 - (0.1\rho - 4)^2 + (0.1\rho - 4)^4/2,$$

$$\beta(z) \equiv 1 - (0.1z - 1)^2 + (0.1z - 1)^4/2.$$

The boundary conditions are taken in the form

$$\begin{aligned} \frac{\partial c^{(k)}}{\partial z}(0, \rho, t) &= \frac{\partial c^{(k)}}{\partial z}(20, \rho, t) = 0, \\ \frac{\partial c^{(k)}}{\partial \rho}(z, 30, t) &= \frac{\partial c^{(k)}}{\partial \rho}(z, 50, t) = 0. \end{aligned}$$

We solved this problem using 20×20 grid consisting of 400 uniformly spaced grid points which resulted in a system of 800 ordinary differential equations. The parameters of computations were set the same as the previous problem. The results for $c^{(1)}$ and $c^{(2)}$ are displayed in Tables 5.11- 5.14

In both the above examples the solution obtained by ASP21 and ASP22 compares favorably with the reference solution and a significant difference in computing time over IGS is observed. The reduction of computing time could prove extremely beneficial when solving realistic systems comprizing of thousands of ordinary differential equations on a PC.

Two Species Diffusion - 2D System							
			REF	IGS	ASP21		
ρ	z	t	$x^1 \times 10^{-5}$	$x^1 \times 10^{-5}$	$\phi^1 \times 10^{-8}$	$v^1 \times 10^3$	$x^1 \times 10^{-5}$
0.000	30.00	0.00	2.500000	2.500000	1.525726	1.639	2.500000
		1.00	0.006007	0.005867	0.003666	1.639	0.006007
		2.00	0.000014	0.000026	0.000008	1.639	0.000014
		3.00	0.000000	0.000000	0.000000	1.639	0.000000
4.210	34.21	0.00	3.611221	3.611221	1.525726	2.367	3.611221
		1.00	0.008676	0.008475	0.003666	2.367	0.008676
		2.00	0.000020	0.000038	0.000008	2.367	0.000020
		3.00	0.000000	0.000000	0.000000	2.367	0.000000
14.73	44.73	0.00	4.084646	4.084646	1.525726	2.677	4.084646
		1.00	0.009813	0.009585	0.003666	2.677	0.009813
		2.00	0.000023	0.000043	0.000008	2.677	0.000023
		3.00	0.000000	0.000000	0.000000	2.677	0.000000
20.00	50.00	0.00	2.590250	2.590250	1.525726	1.698	2.590250
		1.00	0.006224	0.006079	0.003666	1.698	0.006224
		2.00	0.000014	0.000014	0.000008	1.698	0.000014
		3.00	0.000000	0.000000	0.000000	1.698	0.000000
CPU TIME				1667			50

Table 5.11: [ASP21] Numerical Results for Example 5. (c^1 vector)

Two Species Diffusion - 2D System							
			REF	IGS	ASP21		
ρ	z	t	$x^2 \times 10^{-11}$	$x^2 \times 10^{-11}$	$\phi^2 \times 10^{-14}$	$v^2 \times 10^3$	$x^2 \times 10^{-11}$
0.000	30.00	0.00	2.500000	2.500000	1.526	1.639	2.500000
		1.00	2.500005	2.500005	1.526	1.639	2.500005
		2.00	2.500008	2.500008	1.526	1.639	2.500008
		3.00	2.500010	2.500010	1.526	1.639	2.500010
4.210	34.21	0.00	3.611221	3.611221	1.526	2.367	3.611221
		1.00	3.611227	3.611227	1.526	2.367	3.611227
		2.00	3.611229	3.611229	1.526	2.367	3.611229
		3.00	3.611231	3.611231	1.526	2.367	3.611231
14.73	44.73	0.00	4.084646	4.084646	1.526	2.677	4.084646
		1.00	4.084641	4.084641	1.526	2.677	4.084641
		2.00	4.084631	4.084631	1.526	2.677	4.084631
		3.00	4.084622	4.084622	1.526	2.677	4.084622
20.00	50.00	0.00	2.590250	2.590250	1.526	1.698	2.590250
		1.00	2.590329	2.590329	1.526	1.698	2.590329
		2.00	2.590405	2.590405	1.526	1.698	2.590405
		3.00	2.590481	2.590481	1.526	1.698	2.590481
CPU TIME				1667			50

Table 5.12: [ASP21] Numerical Results for Example 5. (c^2 vector)

Two Species Diffusion - 2D System							
			REF	IGS	ASP22		
ρ	z	t	$x^1 \times 10^{-5}$	$x^1 \times 10^{-5}$	$\phi^1 \times 10^{-8}$	$v^1 \times 10^3$	$x^1 \times 10^{-5}$
0.000	30.00	0.00	2.500000	2.500000	1.525726	1.639	2.500000
		1.00	0.006007	0.005867	0.003668	1.639	0.006010
		2.00	0.000014	0.000026	0.000008	1.639	0.000014
		3.00	0.000000	0.000000	0.000000	1.639	0.000000
4.210	34.21	0.00	3.611221	3.611221	1.525726	2.367	3.611221
		1.00	0.008676	0.008475	0.003668	2.367	0.008681
		2.00	0.000020	0.000038	0.000008	2.367	0.000020
		3.00	0.000000	0.000000	0.000000	2.367	0.000000
14.73	44.73	0.00	4.084646	4.084646	1.525726	2.677	4.084646
		1.00	0.009813	0.009585	0.003668	2.677	0.009819
		2.00	0.000023	0.000043	0.000008	2.677	0.000023
		3.00	0.000000	0.000000	0.000000	2.677	0.000000
20.00	50.00	0.00	2.590250	2.590250	1.525726	1.698	2.590250
		1.00	0.006224	0.006079	0.003668	1.698	0.006227
		2.00	0.000014	0.000014	0.000008	1.698	0.000014
		3.00	0.000000	0.000000	0.000000	1.698	0.000000
CPU TIME				1667			31

Table 5.13: [ASP22] Numerical Results for Example 5. (c^1 vector)

Two Species Diffusion - 2D System							
			REF	IGS	ASP22		
ρ	z	t	$x^1 \times 10^{-11}$	$x^1 \times 10^{-11}$	$\phi^1 \times 10^{-14}$	$v^1 \times 10^3$	$x^1 \times 10^{-11}$
0.000	30.00	0.00	2.500000	2.500000	1.526	1.639	2.500000
		1.00	2.500005	2.500005	1.526	1.639	2.500005
		2.00	2.500008	2.500008	1.526	1.639	2.500008
		3.00	2.500010	2.500010	1.526	1.639	2.500010
4.210	34.21	0.00	3.611221	3.611221	1.526	2.367	3.611221
		1.00	3.611227	3.611227	1.526	2.367	3.611227
		2.00	3.611229	3.611229	1.526	2.367	3.611229
		3.00	3.611231	3.611231	1.526	2.367	3.611231
14.73	44.73	0.00	4.084646	4.084646	1.526	2.677	4.084646
		1.00	4.084641	4.084641	1.526	2.677	4.084641
		2.00	4.084631	4.084631	1.526	2.677	4.084631
		3.00	4.084622	4.084622	1.526	2.677	4.084622
20.00	50.00	0.00	2.590250	2.590250	1.526	1.698	2.590250
		1.00	2.590329	2.590329	1.526	1.698	2.590329
		2.00	2.590405	2.590405	1.526	1.698	2.590405
		3.00	2.590481	2.590481	1.526	1.698	2.590481
CPU TIME				1667			31

Table 5.14: [ASP22] Numerical Results for Example 5. (c^2 vector)

As a final numerical example we have.

Example 6

We consider a pair of nonlinear PDEs with exact solution [14].

$$\begin{aligned}\frac{\partial u}{\partial t} &= \frac{\partial}{\partial \rho}\left((v-1)\frac{\partial u}{\partial \rho}\right) + (16\rho t - 2t - 16(v-1))(u-1) + 10\rho e^{-4\rho} \\ \frac{\partial v}{\partial t} &= \frac{\partial^2 v}{\partial \rho^2} + \frac{\partial u}{\partial \rho} + 4u - 4 + \rho^2 - 2t - 10te^{-4\rho},\end{aligned}$$

with the initial conditions

$$u(\rho,0) = 1, \qquad v(\rho,0) = 1,$$

and the boundary conditions

$$u(0,t) = v(0,t) = 1,$$

$$3u(1,t) + u_\rho(1,t) = 3, \qquad 5v_\rho(1,t) = e^4(u(1,t) - 1).$$

The exact solution is given by

$$\begin{aligned}u(\rho,t) &= 1 + 10\rho te^{-4\rho}, \\ v(\rho,t) &= 1 + \rho^2 t.\end{aligned}$$

We used a uniform mesh of 50 points on the spatial interval $(0 \leq \rho \leq 1)$ which resulted in a system of 100 ordinary differential equations. We used ASP21 to solve this problem. The initial time step and error tolerance were both set equal to 10^{-3} in ASP21. The error tolerance in IGS was also set equal to 10^{-3}. The results for the solution subvectors u and v are shown in Tables 5.15-5.16 As a reference solution we use the exact solution. From this example we see that even in the case of nonlinear equations in which the shape equations depend strongly on the amplitudes our approach is quite robust. It competes reasonably well with the implicit Gear's solver and is much faster than the explicit Gear's solver.

Nonlinear System							
		REF	IGS	EGS	ASP21		
ρ	t	x^1	x^1	x^1	$\phi^1 \times 10^{-1}$	$v^1 \times 10^2$	x^1
0.00	0.00	1.0000	1.0000	1.0000	5.000	2.000	1.0000
	0.20	1.0000	1.0000	1.0000	5.558	1.799	1.0000
	0.50	1.0000	1.0000	1.0000	6.394	1.564	1.0000
	0.80	1.0000	1.0000	1.0000	7.226	1.384	1.0000
	1.00	1.0000	1.0000	1.0000	7.784	1.285	1.0000
0.38	0.00	1.0000	1.0000	1.0000	5.000	2.000	1.0000
	0.20	1.1644	1.1644	1.1644	5.558	2.095	1.1644
	0.50	1.4110	1.4108	1.4108	6.394	2.207	1.4108
	0.80	1.6577	1.6568	1.6570	7.226	2.293	1.6565
	1.00	1.8221	1.8210	1.8210	7.784	2.340	1.8215
0.79	0.00	1.0000	1.0000	1.0000	5.000	2.000	1.0000
	0.20	1.0659	1.0659	1.0659	5.558	1.918	1.0659
	0.50	1.1648	1.1648	1.1648	6.394	1.822	1.1648
	0.80	1.2638	1.2635	1.2636	7.226	1.749	1.2634
	1.00	1.3297	1.3294	1.3294	7.784	1.708	1.3298
1.00	0.00	1.0000	1.0000	1.0000	5.000	2.000	1.0000
	0.20	1.0366	1.0366	1.0366	5.558	1.865	1.0366
	0.50	1.0915	1.0915	1.0915	6.394	1.707	1.0915
	0.80	1.1465	1.1464	1.1464	7.226	1.587	1.1463
	1.00	1.1831	1.1830	1.1830	7.784	1.520	1.1882
CPU TIME			3	20			3

Table 5.15: [ASP21] Numerical Results for Example 6. (u vector)

Nonlinear System							
		REF	IGS	EGS	ASP21		
ρ	t	x^2	x^2	x^2	$\phi^2 \times 10^{-1}$	$v^2 \times 10^2$	x^2
0.00	0.00	1.0000	1.0000	1.0000	5.000	2.000	1.0000
	0.20	1.0000	1.0000	1.0000	5.337	1.874	1.0000
	0.50	1.0000	1.0000	1.0000	5.843	1.712	1.0000
	0.80	1.0000	1.0000	1.0000	6.341	1.577	1.0000
	1.00	1.0000	1.0000	1.0000	6.680	1.497	1.0000
0.38	0.00	1.0000	1.0000	1.0000	5.000	2.000	1.0000
	0.20	1.0300	1.0301	1.0301	5.337	1.930	1.0301
	0.50	1.0751	1.0753	1.0753	5.843	1.841	1.0753
	0.80	1.1202	1.1205	1.1205	6.341	1.766	1.1200
	1.00	1.1503	1.1505	1.1505	6.680	1.722	1.1500
0.79	0.00	1.0000	1.0000	1.0000	5.000	2.000	1.0000
	0.20	1.1266	1.1267	1.1267	5.337	2.111	1.1267
	0.50	1.3167	1.3168	1.3168	5.843	2.254	1.3168
	0.80	1.5067	1.5068	1.5067	6.341	2.373	1.5047
	1.00	1.6334	1.6332	1.6337	6.680	2.443	1.6318
1.00	0.00	1.0000	1.0000	1.0000	5.000	2.000	1.0000
	0.20	1.2000	1.2000	1.2000	5.337	2.294	1.2000
	0.50	1.5000	1.5000	1.5000	5.843	2.567	1.5000
	0.80	1.8000	1.7999	1.7998	6.341	2.835	1.7974
	1.00	2.0000	1.9995	1.9996	6.680	2.991	1.9982
CPU TIME			3	20			3

Table 5.16: [ASP21] Numerical Results for Example 6. (v vector)

Chapter 6

Conclusions

The solution of ODEs is frequently encountered in practical applications. In most cases, however, the system of ODEs cannot be solved analytically and numerical methods have to be used. On the other hand, the system of ODEs describing a physically realistic problem is often very large and stiff and the numerical solution of such a system is difficult. A typical example is the class of chemical problems presented in Section 2.7. When the method of lines is applied to this class of problems a large stiff system of ODEs results. As observed in Section 2.6, the solution of a stiff system of ODEs requires an implicit method. At the same time it is seen that implicit methods can be extremely prohibitive in both cost and storage requirements, sometimes to such an extent that the method may not be applicable at all.

The amplitude-shape method, investigated in this thesis, is intended to reduce the computational effort involved with solving certain systems of stiff ODEs occuring in practical applications. With this method the original system of ODEs is transformed into a new system which is easier to manage from a computational point of view. To be precise, the stiffness present in the original system is confined to only a few equations described by a

complicated amplitude system while the majority of the equations are incorporated into a properly behaved shape system. It is then possible to treat the amplitude system with an implicit method suitable for solving stiff systems, while the shape system can be treated with an inexpensive explicit method. For this purpose we used a mixed implicit-explicit scheme, together with extrapolation to monitor the error and adjust the stepsize.

The amplitude-shape method was applied to various examples including two practical examples from chemical kinetics. The examples were chosen to demonstrate the effectiveness and robustness of the method. The method proved to be particularly effective for the solution of the chemical kinetics systems and resulted in a significant gain in efficiency. Such an improvement in efficiency may be especially useful for those who utilize PCs to solve extremely large systems of stiff ODEs and have to repeat the computations many times identifying the parameters.

We observe that when the amplitude-shape method is applicable, it becomes substantially advantangeous in terms of storage requirement and solution effort. It is evident that the application of the amplitude-shape method requires a fair amount of physical intuition into the prognosis of a given physical system. However, we are of the opinion that the time spent in this way will be duely compensated by the substantial gain in computational efficiency when the amplitude-shape method is applicable.

Finally, we have chosen as our field of application the area of chemical kinetics. It is envisaged that nuclear reactor kinetics maybe be another area of application.

Bibliography

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

[2] G D Byrne and A C Hindmarsh, Stiff ODE Solvers: A Review of Current and Coming Attractions, *Journal of Computational Physics*, **70**, 1–62 (1987)

[3] P Deuflhard, Order and Stepsize Control in Extrapolation Methods, *Numerische Mathematik*, **41**, 399–422 (1983)

[4] P Deuflhard, Recent Progress in Extrapolation Methods for Ordinary Differential Equations, *SIAM Review*, **27**, 505–535 (1985)

[5] H L Dodds, Jr., Accuracy of the quasistatic method for two-dimensional thermal reactor transients with feedback, *Nuclear Science and Engineering* **59**, 271–281 (1976)

[6] C W Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, New Jersey, 1971

[7] E Hairer and CH Lubich, Asymptotic expansions of fixed- stepsize methods, *Numerische Mathematik*, **45**, 345–360 (1984)

[8] P Hartman, *Ordinary Differential Equations*, Birkhäuser, Boston, 1982

[9] A C Hindmarsh, ODE Solvers for Use with Method of Lines, pp. 312–316 in *Advances in Computer Methods for Partial Differential Equations IV*, I M A C S, 1981 (Editors: R Vichnevetsky and R S Stepleman)

[10] E Hofer, Partially implicit method for large stiff systems of ODEs with only few equations introducing small time constants, *SIAM Journal on Numerical Analysis*, **5**, 645–663 (1976)

[11] J M Kozakiewicz and J R Mika, Multigrid method for numerical solution of ordinary differential equations, *Matematyka Stosowana*, **25**, 37–45 (1992)

[12] J D Lambert, *Computational Methods in Ordinary Differential Equations*, John Wiley and Sons, Chichester, 1990

[13] J D Lambert, *Numerical Methods for Ordinary Differential Systems*, John Wiley and Sons, Chichester, 1995

[14] N K Madsen and R F Sincovec, Software for Partial Differential Equations, pp. 229–242 in *Numerical Methods for Differential Systems*, Academic Press, New York, 1976 (Editors: L Lapidus and W E Schiesser)

[15] J R Mika, Mathematical foundations of the quasistatic approximation in reactor physics, *Annals of Nuclear Energy*, **9**, 585–589 (1982)

[16] J R Mika and I Scribani, Multigrid Line Method for Partial Differential Equations, SAMS Congress, Mmabatho, October 1990

[17] J R Mika, K Andrzejewski and N Parumasur, Amplitude-Shape Method: The Quasistatic Method Revisited, *Transport Theory and Statistical Physics*, to appear

[18] J R Mika and N Parumasur, Amplitude-Shape Method for Numerical Solution of Ordinary Differential Equations, NUMDIFF-8, Alexisbad, 1-5 September 1997

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

[20] K O Ott and D A Meneley, Accuracy of the quasistatic treatment of spatial reactor kinetics, *Nuclear Science and Engineering*, **36**, 402–411 (1969)

[21] N Parumasur, The application of the multigrid algorithm to the solution of stiff ordinary differential equations resulting from partial differential equations, M Sc thesis, University of Natal, Durban, South Africa, 1992

[22] W H Press, S A Teukolsky, W T Vetterling and B P Flannery, *Numerical Recipes in Fortran: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1992

[23] W E Schiesser, *The Numerical Method of Lines*, Academic Press, San Diego, New York, 1991

[24] L F Shampine and C W Gear, A User's View of Solving Stiff Differential Equations, *SIAM Review*, **21**, 1–17 (1979)

[25] W Werner, Kinetics of nuclear system, pp. 313–364 in *Advances in Nuclear Science and Technology, Vol. 10*, Plenum Press, New York, 1977, (Editors: E J Henley, J Lewins and M Becker)

APPENDIX

```
      PROGRAM ASP
c     Main program in which the user defined parameters are set,
c     weight vectors evaluated, initial conditions for the amplitude
c     and shape systems computed, computing time calculated and output
c     of results done
c     The user defined parameters are :
c            n1 - number of amplitude equations
c            n2 - number of shape equations
c            t0 - initial time
c          tmax - final time
c          hout - time interval at which solution is to be printed
c            h0 - initial time step
c           eps - user defined error tolerance
c         ncol1 - first column number of solution array to be printed
c                 ncol2 is the second column number, etc.
c     INIT is a user defined routine which supplies the initial
c     solution vector Y and ASSTEP is the driver routine used to
c     advance  the solution over a single step.
c     tnext is the next output time at which the solution is desired
c     hnext is the stepsize to be tried next
c     The arrays used are :
c             w - holds the weight vector
c            fi - contains both the initial values and the values at
c                 each step of the amplitude solution
c         fiscl - array against which the error in the amplitude
c                 solution is scaled
```

```
c             v - contains both the initial values and the values at
c                 each step of the shape solution
c          vscl - array against which the error in the shape
c                 solution is scaled
c             y - contains the solution to the original system of ODEs
      implicit double precision (a-h,o-z)
      PARAMETER (N1MAX=2,N2MAX=800)
      COMMON /inter/ w(800),n1,n2,n
      DIMENSION fi(N1MAX),fiscl(N1MAX)
      DIMENSION v(N2MAX),vscl(N2MAX)
      DIMENSION y(N2MAX)
      call getdat(iyr,imon,iday)
      call gettim(ihr0,imin0,isec0,i100th0)
      write(*,'('' date'',5x,i2,''-'',i2,''-'',i4)') iday,imon,iyr
      write(*,'('' time'',5x,i2,'':'',i2,'':'',i2,''.'',i2)')
     $  ihr0,imin0,isec0,i100th0
      open(1,file='INPUT',status='OLD')
      open(2,file='OUT',status='NEW')
      read(1,*) n1
      read(1,*) n2
      read(1,*) t0
      read(1,*) tmax
      read(1,*) hout
      read(1,*) h0
      read(1,*) eps
      read(1,*) ncol1,ncol2,ncol3,ncol4
      t=t0
      h=h0
```

```
      n=n1*n2
      if (n.gt.N2MAX) goto 100
c  call subroutine init for the initial solution vector
      call init(n1,n2,n,y)
c  evaluate weight vector w
      do i=1,n
        w(i)=dsign(1.d0,y(i))
      enddo
c  compute initial values of fi
      do i=1,n1
        fi(i)=0.d0
        k=(i-1)+1
        do j=(i-1)*n2+1,k*n2
          fi(i)=fi(i)+w(j)*y(j)
        enddo
      enddo
c  compute initial values of v
      do i=1,n1
        k=(i-1)+1
        do j=(i-1)*n2+1,k*n2
          v(j)=y(j)/fi(i)
        enddo
      enddo
      write(2,*)'    T          FI(1)    FI(2)     V(ncol1)   V(ncol2)
     $V(ncol3)   V(ncol4) '
      write(2,*)
      write(2,60)t,fi(1),fi(2),v(ncol1),v(ncol2),v(ncol3),v(ncol4)
      write(*,10)
```

```
10    format(/15x,'AMPLITUDE-SHAPE PROGRAM (ASP21)')
      write(*,*)
      write(*,'('' N='',i4,''  H0='',1pe8.1,''    EPS='',1pe8.1,   )')
     $ n,h0,eps
      write(*,20) ncol1,ncol2,ncol3,ncol4
      write(*,30) t0,y(ncol1),y(ncol2),y(ncol3),y(ncol4)
20     format(/5x,' T',10x,'Y(' i3 ')',8x,'Y(',i3,')',8x,'Y(',i3,')'
     $ 8x,'Y(',i3,')'/)
30     format(1pe10.3,5(1x,1pe13.6))
      tnext=tnext+hout
c  main loop starts here
40     continue
c  scale components of fi
      do i=1,n1
        fiscl(i)=dmax1(1.d0,dabs(fi(i)))
      enddo
c  scale components of v (no scaling needed - dummy value substituted)
      do j=1,n
        vscl(j)=1.d0
      enddo
c  make a call to the routine ASSTEP
      call asstep(t,h,n1,n2,n,fi,v,fiscl,vscl,eps,hnext)
c  calculate solution vector y
      do i=1,n1
        k=(i-1)+1
        do j=(i-1)*n2+1,k*n2
          y(j)=fi(i)*v(j)
        enddo
```

```
      enddo
c  evaluate weight vector w for next step
      do i=1,n
        w(i)=dsign(1.d0,y(i))
      enddo
      h=hnext
      if(t.ge.tnext)then
      write(*,50)t,y(ncol1),y(ncol2),y(ncol3),y(ncol4)
50     format(1pe10.3,5(1x,1pe13.6))
      write(2,60)t,fi(1),fi(2),v(ncol1),v(ncol2),v(ncol3),v(ncol4)
60     format(1pe10.3,6(1x,1pe10.3))
c  calculate tnext and if the present step overshoots tnext decrease h
      t2=tnext
      tnext=tnext+hout
      endif
      t1=t+h
      if((t1-tnext)*(t1-t2).gt.0.d0) h=tnext-t
      if(t.lt.tmax) goto 40
c  main loop ends here
      call gettim(ihr1,imin1,isec1,i100th1)
      ihr=ihr1-ihr0
      imin=imin1-imin0
      isec=isec1-isec0
      i100th=i100th1-i100th0
      ictime=3600*(ihr1-ihr0)+60*(imin1-imin0)+(isec1-isec0)+
     $ .01*(i100th1-i100th0)
      write(*,'('' COMPUTING TIME WAS  '',i5,'' S'')') ictime
      stop' OK!'
```

```
100   WRITE(*,110)
110   FORMAT(//,30H ERROR - VALUE OF N EXCEEDED,//)
      stop
      end


      SUBROUTINE asstep(t,h,n1,n2,m,fi,v,fiscl,vscl,eps,hnext)
c     Driver routine for advancing the solution over a single step
c.    by making a call to the core integrator routine ASSOLV.
c     The accuracy is controlled by monitoring the local truncation
c     error and the stepsize is adjusted automatically. The arrays
c     fi,fiscl,v,vscl have the same meaning as in the main program
c     ASP and fisav,vsav,fitemp,vtemp are arrays used for temporary
c     storage.
c     RNORM is a user written function for evaluating the scaled
c     maximum vector norm.
c     At the end of the integration step the improved solution
c     values for fi and v are returned and the next stepsize to be
c     attempted returned.
      implicit double precision (a-h,o-z)
      PARAMETER (N1MX=2,N2MX=800)
      DIMENSION fi(n1),fiscl(n1),fisav(N1MX)
      DIMENSION v(m),vscl(m),vsav(N2MX)
      DIMENSION fitemp(N1MX),vtemp(N2MX)
c  Save initial data
      tsav=t
      do i=1,n1
        fisav(i)=fi(i)
      enddo
```

```
      do j=1,m
        vsav(j)=v(j)
      enddo
c  Cut stepsize in half
901   hh=0.5d0*h
c  Perform the integration using two half steps
      call assolv(t,hh,n1,n2,m,fi,v)
      t=tsav+hh
      call assolv(t,hh,n1,n2,m,fi,v)
      t=tsav+h
      if(t.eq.tsav) pause 'ERROR- value of h infintesimal'
c  Save current solution values
      do i=1,n1
        fitemp(i)=fisav(i)
      enddo
      do j=1,m
        vtemp(j)=vsav(j)
      enddo
c  Perform the integration using a single step
      call assolv(tsav,h,n1,n2,m,fitemp,vtemp)
c  Estimate the local truncation error and use it to monitor the
c  accuracy. The arrays fitemp and vtemp are used to store the error.
      do i=1,n1
        fitemp(i)=fi(i)-fitemp(i)
      enddo
      do j=1,m
        vtemp(j)=v(j)-vtemp(j)
      enddo
```

```
      rnorm1=rnorm(n1,fiscl,fitemp)
      rnorm2=rnorm(m,vscl,vtemp)
      err=dmax1(rnorm1,rnorm2)/eps
c  If error criterion is satisfied estimate the next step and return
c  the improved solution values
      if(err.le.1.d0)then
        r=(err**(-1.d0/3.d0))
c   increase step by at most a factor of 2
        r=dmin1(2.d0,r)
        hnext=0.9d0*r*h
        do i=1,n1
          fi(i)=fi(i)+fitemp(i)*(1.d0/3.d0)
        enddo
        do j=1,m
          v(j)=v(j)+vtemp(j)*(1.d0/3.d0)
        enddo
      return
c  Else if the current step failed reduce the stepsize and try again
      else
        h=0.9d0*(err**(-1.d0/3.d0))*h
        t=tsav
        do i=1,n1
          fi(i)=fisav(i)
        enddo
        do j=1,m
          v(j)=vsav(j)
        enddo
        goto 901
```

```
      endif
      end

      FUNCTION rnorm(nz,zscl,z)
c     User written function for evaluating the scaled maximum vector
c     norm
c     nz is the dimension of the vector, zscl is an array against
c     which the vector is to be scaled and z is the array containing
c     the vector whose norm is required.
      implicit double precision (a-h,o-z)
      DIMENSION zscl(nz),z(nz)
      rnorm=0.d0
      do i=1,nz
        rnorm=dmax1(rnorm,dabs(z(i)/zscl(i)))
      enddo
      return
      end

      SUBROUTINE assolv(ts,h,n1,n2,m,fi,v)
c     Core integrator routine which implements the Amplitude-Shape
c     numerical scheme (equations (4.1)-(4.2)). The arrays fi and v
c     have the same meaning as in ASP and the arrays dfi and dv
c     contain the derivatives of fi and v, respectively. AMPL and
c     SHAPE are built-in routines which evaluates the derivatives
c     of fi and v, respectively. The parameters used in the Newton
c     iteration are
c          delta  - used to perturb solution when computing partial
c                   derivatives.
```

```
c          maxits - maximum number of iterations
c            Ftol - tolearnce for function values
c            Xtol - tolerance for solution values
c     The arrays fisav,vsav,dfisav,dvsav are used for temporary
c     storage and the array u is the unit vector. The function rnorm
c     has the same meanong as before.
      implicit double precision (a-h,o-z)
      PARAMETER(N1MX=2,N2MX=800)
      DIMENSION fi(n1),fi0(N1MX),dfi(N1MX),dfi0(N1MX)
      DIMENSION fip(N1MX),dfip(N1MX),F(N1MX),dx(N1MX)
      DIMENSION rp(N1MX,N1MX),u(N1MX)
      DIMENSION v(m),v0(N2MX),dv(N2MX),dv0(N2MX)
c  Initialize parameters for Newton iteration and vector u
      delta=1.d-2
      maxits=3
      Ftol=1.d-3
      Xtol=1.d-3
      do i=1,n1
        u(i)=1.d0
      enddo
c  Save initial values of fi and v
      do i=1,n1
        fi0(i)=fi(i)
      enddo
      do j=1,m
        v0(j)=v(j)
      enddo
c  Euler estimate of fi and v
```

```
      call difeq1(ts,n1,n2,m,fi0,v0,dfi0)
      do i=1,n1
        fi(i)=fi0(i)+h*dfi0(i)
      enddo
      call difeq2(ts,n1,n2,m,fi0,v0,dv0)
      do j=1,m
        v(j)=v0(j)+h*dv0(j)
      enddo
      ts=ts+h
c  Begin Newton iteration
      do iter=1,maxits
        call difeq1(ts,n1,n2,m,fi,v,dfi)
        do i=1,n1
          F(i)=-fi(i)+fi0(i)+0.5d0*h*(dfi0(i)+dfi(i))
        enddo
c  Check for convergence in function values
        errf=rnorm(n1,u,F)
        if (errf.le.Ftol) goto 911
c  Compute partial derivatives
        do i=1,n1
          fip(i)=fi(i)
          dfip(i)=dfi(i)
        enddo
        do k=1,n1
          fi(k)=fi(k)+delta
          call difeq1(ts,n1,n2,m,fi,v,dfi)
          do i=1,n1
            if(i.eq.k) then
```

```
            rp(i,k)=1.d0-0.5d0*h*(dfi(i)-dfip(i))/delta
          else
            rp(i,k)=-0.5d0*h*(dfi(i)-dfip(i))/delta
          endif
        enddo
        fi(k)=fip(k)
      enddo
      do i=1,n1
        dfi(i)=dfip(i)
      enddo
c  Solve system of equations for increment directly
      D=rp(1,1)*rp(2,2)-rp(1,2)*rp(2,1)
      dx(1)=(F(1)*rp(2,2)-F(2)*rp(1,2))/D
      dx(2)=(F(2)*rp(1,1)-F(1)*rp(2,1))/D
c  Add increment to solution to obtain second order estimate of fi
      do i=1,n1
        fi(i)=fi(i)+dx(i)
      enddo
c  Check for convergence in solution values
      errx=rnorm(n1,u,dx)
      if (errx.le.Xtol) goto 911
    enddo
911   call difeq2(ts,n1,n2,m,fi,v,dv)
c  Second order estimate of v
    do j=1,m
      v(j)=v0(j)+0.5d0*h*(dv0(j)+dv(j))
    enddo
    return
```

```
      end


      subroutine difeq1(t,n1,n2,m,fi,v,dfi)
c     Built-in routine for evaluating the right hand side of the
c     amplitude equation. The arrays fi and v have the same
c     meaning as in ASP and dfi is an array which contains the
c     derivative of fi. The array w contains the weight vector and
c     yy contains the solution vector. DIFEQ is a user written
c     routine which for evaluating the right hand side of the
c     original system of equations. The array dydx contains the
c     derivative of yy.
      implicit double precision (a-h,o-z)
      PARAMETER (N1MX=2, N2MX=800)
      common/inter/w(N2MX)
      dimension fi(n1),dfi(N1MX)
      dimension v(m)
      dimension yy(N2MX),dydx(N2MX)
c  Compute solution by taking the product of fi and v
      do i=1,n1
        l1=(i-1)*n2+1
        l2=i*n2
        do j=l1,l2
          yy(j)=fi(i)*v(j)
        enddo
      enddo
      call difeq(t,n1,n2,m,yy,dydx)
c   Evaluate right hand side of amplitude system
      do i=1,n1
```

```
        l1=(i-1)*n2+1
        l2=i*n2
        dfi(i)=0.d0
        do j=l1,l2
          dfi(i)=dfi(i)+w(j)*dydx(j)
        enddo
      enddo
      return
      end

      subroutine difeq2(t,n1,n2,m,fi,v,dv)
c     Built-in routine for evaluating the right hand side of the
c     shape equation. The arrays fi and v have the same meaning
c     as in ASP and dv is an array which contains the derivative
c     of v. The array w contains the weight vector and yy contains
c     the solution vector. DIFEQ is a user written routine which
c     for evaluating the right hand side of the original system of
c     equations. The array dydx contains the derivative of yy.
      implicit double precision (a-h,o-z)
      PARAMETER (N1MX=2, N2MX=800)
      common/inter/w(N2MX)
      dimension fi(n1)
      dimension v(m),dv(N2MX)
      dimension yy(N2MX),dydx(N2MX)
c  Compute solution by taking the product of fi and v
      do i=1,n1
        l1=(i-1)*n2+1
        l2=i*n2
```

```
        do j=l1,l2
          yy(j)=fi(i)*v(j)
        enddo
      enddo
      call difeq(t,n1,n2,m,yy,dydx)
c  Evaluate right hand side of shape system
      do i=1,n1
        l1=(i-1)*n2+1
        l2=i*n2
        fiinv=1.d0/fi(i)
        s=0.d0
        do j=l1,l2
          s=s+w(j)*dydx(j)
        enddo
        do j=l1,l2
          dv(j)=fiinv*(dydx(j)-s*v(j))
        enddo
      enddo
      return
      end


      Subroutine difeq(t,n1,n2,n,y,dy)
c     User defined routine to evaluate the right hand side of the
c     original system of ODEs. This routine is for the Two Species
c     Diffusion - 1D System. The arrays y and dy contain the
c     solution vector and its derivative, respectively.
      implicit double precision (a-h,o-z)
      PARAMETER (NMAX=800)
```

```
      DIMENSION y(n),dy(n)
      COMMON /MESH/hz,z(NMAX)
      d=1.d0/hz**2
c  discretization at first mesh point
      m1=n2+1
      m2=n2+2
      g=rk(1.d0-0.5d0)
      h=rk(1.d0+0.5d0)
      dy(1)=2.d0*d*(g+h)*(y(2)-y(1))+r1(y(1),y(m1),t)
      dy(m1)=2.d0*d*(g+h)*(y(m2)-y(m1))+r2(y(1),y(m1),t)
c  discretization at interior mesh points
      do i=2,n2-1
        g=rk(dble(i)-0.5d0)
        h=rk(dble(i)+0.5d0)
        im=i+n2
        dy(i)=d*(g*y(i-1)-(g+h)*y(i)+h*y(i+1))+r1(y(i),y(im),t)
        dy(im)=d*(g*y(im-1)-(g+h)*y(im)+h*y(im+1))+r2(y(i),y(im),t)
      enddo
c  discretization at final mesh point
      g=rk(n2-0.5d0)
      h=rk(n2+0.5d0)
      dy(n2)=2.d0*d*(g+h)*(y(n2-1)-y(n2))+r1(y(n2),y(n),t)
      dy(n)=2.d0*d*(g+h)*(y(n-1)-y(n))+r2(y(n2),y(n),t)
      return
      end

      FUNCTION rk(v)
c     Function for calculating diffusion coefficients
```

```
      implicit double precision (a-h,o-z)
      PARAMETER (NMAX=800)
      COMMON/MESH/hz,z(NMAX)
      rk=1.d-8*dexp((30.d0+v*hz)/5.d0)
      return
      end

      FUNCTION r1(y1,y2,t)
c     Function for calculating reaction term 1
      implicit double precision (a-h,o-z)
      COMMON/FUNC/c3,c4
      DATA y3,c1,c2/3.7d16,1.63d-16,4.66d-16/
      pi=4.d0*atan(1.d0)
      w=pi/43200.d0
      a3=22.62d0
      a4=7.601d0
      if (dsin(w*t).gt.0.d0) then
        c3=dexp(-a3/dsin(w*t))
        c4=dexp(-a4/dsin(w*t))
      else
        if (dsin(w*t).le.0.d0) then
        c3=0.d0
        c4=0.d0
        endif
      endif
      r1=-c1*y1*y3-c2*y1*y2+2.d0*c3*y3+c4*y2
      return
      end
```

```
      FUNCTION r2(y1,y2,t)
c     Function for calculating reaction term 2
      implicit double precision (a-h,o-z)
      COMMON/FUNC/c3,c4
      DATA y3,c1,c2/3.7d16,1.63d-16,4.66d-16/
      pi=4.d0*atan(1.d0)
      w=pi/43200.d0
      a3=22.62d0
      a4=7.601d0
      if (dsin(w*t).gt.0.d0) then
        c3=dexp(-a3/dsin(w*t))
        c4=dexp(-a4/dsin(w*t))
      else
        if (dsin(w*t).le.0.d0) then
        c3=0.d0
        c4=0.d0
        endif
      endif
      r2=c1*y1*y3-c2*y1*y2-c4*y2
      return
      end

      SUBROUTINE init(n1,n2,n,y)
c     User written subroutine for evaluating the initial
c     conditions of the original system of equations and
c     is returned in the array z. GM is a function which
c     returns the values of gamma.
```

```
      implicit double precision (a-h,o-z)
      PARAMETER (NMAX=800)
      COMMON/MESH/hz,z(NMAX)
      DIMENSION y(n)
      pi=4.d0*atan(1.d0)
      hz=20.d0/dble(n2-1)
c  Initialize mesh points
      do i=1,n2
        z(i)=30.d0+dble(i-1)*hz
      enddo
c  Calculate initial vector
      do i=1,n2
        y(i)=1.d6*gm(z(i))
        y(i+n2)=1.d12*gm(z(i))
      enddo
      return
      end

      FUNCTION gm(z)
c  Function for calculating gamma.
      implicit double precision (a-h,o-z)
      gm=1.d0-(z/10.d0-4.d0)**2+0.5d0*(z/10.d0-4.d0)**4
      return
      end
```

```
    *** Input File for Two Species Diffusion - 1D System ***

2                       M
100                     K
0.D0                    T0
4.D0                    TMAX
1.D0                    DTOUT
1.D-4                   DT
1.0D-4                  EPS
1,25,75,100             NCOL
```

*** Output of Results for Two Species Diffusion - 1D System ***

AMPLITUDE-SHAPE PROGRAM (ASP21)

N= 200 H0= 1.0E-04 EPS= 1.0E-04

T	Y(1)	Y(25)	Y(75)	Y(100)
0.000E+00	5.000000E+05	7.698325E+05	7.850314E+05	5.000000E+05
1.000E+00	1.201269E+03	1.849318E+03	1.885811E+03	1.201311E+03
2.000E+00	2.886093E+00	4.442495E+00	4.530119E+00	2.886294E+00
3.000E+00	6.933947E-03	1.067191E-02	1.088230E-02	6.934668E-03
4.000E+00	1.666625E-05	2.564749E-05	2.615287E-05	1.666855E-05

COMPUTING TIME WAS 11 S

T	FI(1)	FI(2)	V(ncol1)	V(ncol2)	V(ncol3)
0.000E+00	7.640E+07	7.640E+13	6.545E-03	1.008E-02	1.028E-02
1.000E+00	1.835E+05	7.640E+13	6.545E-03	1.008E-02	1.028E-02
2.000E+00	4.409E+02	7.640E+13	6.546E-03	1.008E-02	1.028E-02
3.000E+00	1.059E+00	7.640E+13	6.547E-03	1.008E-02	1.028E-02
4.000E+00	2.545E-03	7.640E+13	6.548E-03	1.008E-02	1.028E-02