# The development and testing of a new integration method for the solution of stiff differential equations arising in the digital simulation of fluid power systems

Caplen, Mark J. S.

*Award date:*
1988

*Awarding institution:*
University of Bath

Link to publication

# THE DEVELOPMENT AND TESTING OF A NEW INTEGRATION METHOD FOR THE SOLUTION OF STIFF DIFFERENTIAL EQUATIONS ARISING IN THE DIGITAL SIMULATION OF FLUID POWER SYSTEMS

Submitted by

Mark J. S. Caplen

for the degree of Ph.D.

of the University of Bath

1988

[illegible]

## SUMMARY

The main portion of the work presented in this thesis is concerned with the development and testing of a new integration method, which is specifically aimed at solving stiff ordinary differential equations arising in the digital simulation of Fluid Power systems. The method has been tested on a variety of problems arising from hydraulic systems, and has been satisfactorily incorporated as an alternative integration method in the Hydraulic Automatic Simulation Package (HASP), a package designed within the School of Engineering at the University of Bath, specifically for the simulation of Fluid Power systems. The performance of the package using this new integrator, with selected Fluid Power circuits, is investigated.

The thesis highlights the mathematical difficulties that occur in Fluid Power simulation, and discusses in detail the ways in which these problems are overcome. Mathematical stiffness is the major problem that arises, and is the main reason for studying the new integration method. The method developed to cope with this problem has shown improved performance over conventional integration methods in certain application areas, and the advantages and disadvantages of the integrator are identified in detail. The analysis of the method shows that it possesses good stability properties, which are essential if a method is to be used for solving stiff differential equations. The method is particularly suited for some types of problems, an example being diagonally dominant systems, and these arise in the discretisation of parabolic partial differential equations. This application of the method is also investigated.

In general, the integration method provides an alternative to conventional integration methods, and is worth considering for the simulation of Fluid Power systems. The work presented has also helped to clarify the direction in which numerical integration methods should be headed, particularly when implemented as general purpose, automatic differential equation solvers. The latter portion of this thesis studies the idea of using methods which are particularly suited to individual problems, and examines the potential of an expert systems approach to the automatic selection of methods within a numerical integration algorithm.

**Acknowledgements**

The author would like to thank Professor D. E. Bowns and Dr. R. E. Dorey for their supervision and guidance throughout the project, and to Mr K. Blackmore for his valuable advice. The financial support given by the Science and Engineering Research Council is gratefully acknowledged. Finally, the author would like to thank his family for their support and encouragement throughout the duration of the work.

## CONTENTS

## List Of Tables

**List Of Figures**

## Notation

| | | |
|---|---|---|
| $a(t)$ | - | coefficient of the differential equation $y' = a(t)y + b(t)$ |
| $a_n$ | - | value of coefficient $a(t)$ at time $t_n$ |
| $a_{in}$ | - | value of coefficient ai at time $t_n$ |
| $a_{in+1}$ | - | value of coefficient ai at time $t_{n+1}$ |
| $A$ | - | system matrix |
| $A_1$ | - | area of an actuator - piston end |
| $A_2$ | - | area of an actuator - rod end |
| $b(t)$ | - | forcing function of the differential equation $y' = a(t)y + b(t)$ |
| $b_n$ | - | value of coefficient $b(t)$ at time $t_n$ |
| $b_{in}$ | - | value of coefficient bi at time $t_n$ |
| $b_{in+1}$ | - | value of coefficient bi at time $t_{n+1}$ |
| $B$ | - | bulk modulus of hydraulic oil |
| $d_o$ | - | fraction of air dissolved in hydraulic oil at S.T.P. |
| det | - | determinant |
| $D$ | - | diagonal matrix |
| $D_f$ | - | finite region |
| $e_n$ | - | global error of a numerical scheme at time $t_n$ |
| $E_{n+1}$ | - | local error made in one step of a numerical scheme |
| $f$ | - | viscous friction coefficient |
| $f(t,y)$ | - | derivative of variable y at time t |
| $f_w$ | - | windage loss coefficient |
| $F$ | - | force |
| $F_c$ | - | coulomb friction force |
| $G$ | - | matrix governing the error propogation of the solution obtained by the new method |

| | | |
|---|---|---|
| $h$ | - | integration time step |
| $I$ | - | identity matrix |
| $J$ | - | moment of inertia |
| $J_{ac}$ | - | Jacobian matrix |
| $k$ | - | spring rate |
| $k_r$ | - | flow coefficient of a relief valve |
| $k_o$ | - | flow coefficient of an orifice |
| $K_{1,2,3}$ | - | coefficients of a second order differential equation |
| $L$ | - | Lipschitz constant |
| $L_1$ | - | local error function |
| $M$ | - | mass of load |
| $M_e$ | - | error matrix |
| $M_y$ | - | Lipschitz constant with respect to $y_{n+k}$ |
| $n$ | - | polytropic index of a gas |
| $p$ | - | order of accuracy of a numerical method |
| $P$ | - | oil pressure in a pipe |
| $P_m$ | - | modal matrix |
| $Q$ | - | flow rate |
| $s$ | - | variable of integration |
| sup | - | supremum |
| $S$ | - | stiffness ratio |
| Sgn(z) | - | sign function of z |
| $t$ | - | independent variable |
| $t_n$ | - | integration time corresponding to step n |
| $T_l$ | - | load torque |
| $T_m$ | - | motor torque |
| $u$ | - | velocity of an actuator piston |

| | | |
|---|---|---|
| $v_i$ | - | volume of pipe i |
| $x$ | - | displacement of an actuator piston |
| $y$ | - | dependent variable |
| $\underline{y}$ | - | vector notation for dependent variable |
| $y'$ | - | first derivative of the dependent variable with respect to time |
| $y''$ | - | second derivative of the dependent variable |
| $y(t_n)$ | - | true value of dependent variable at time $t_n$ |
| $y_n$ | - | approximation to $y(t_n)$ |
| $z_n$ | - | modified solution to a difference equation |
| $\Delta$ | - | differential operator |
| $\varepsilon_n$ | - | vector matrix of errors at time $t_n$ |
| $\lambda_k$ | - | eigenvalue of the system matrix |
| $\mu$ | – | viscosity of hydraulic oil |
| $\mu_k$ | - | real part of $\lambda_k$ |
| $\rho(\xi)$ | – | first characteristic polynomial of a multi-step method |
| $\sigma(\xi)$ | - | second characteristic polynomial of a multi-step method |
| $\phi(t,y,h)$ | - | function defining a single step method |
| $\psi(t,y,h)$ | - | function defining new method |
| $\omega_k$ | - | imaginary part of $\lambda_k$ |
| $\omega$ | - | angular velocity |
| $[..]^T$ | - | transposed matrix |

# CHAPTER 1

**DETAILED CONTENTS** **Page**

# CHAPTER 1

# INTRODUCTION

## The Simulation Of Engineering Systems

**1.1** Many physical situations can be represented, after appropriate modification or simplification, by a set of algebraic or differential equations. The solution of these mathematical systems, either by analytical or numerical methods, accompanied by the appropriate analysis of the results, can give an accurate indication of the behaviour of the original system. Both the mathematical modelling and the solution of the resulting equations can be automated by the use of a digital computer, with software playing the role of both the modeller and solver. Consequently, it is possible to simulate on a digital computer the behaviour of an engineering system, without the actual construction of that system.

**1.2** Simulation allows the accurate design and analysis of many engineering systems to be undertaken in the safe and relatively cheap environment of the computer. Consequently, a great deal of research has been carried out in the field of simulation, and at the centre of this work lies the problem of solving the equations that arise in the mathematical modelling of the system components. Most simulations are carried out in a time domain, and require the solution of sets of ordinary differential equations. Although algebraic equations can present difficulties in their solution, it is the difficulties that arise with the differential equations that are studied here. This thesis is concerned with the problems involved in solving the sets of differential equations that arise in the simulation of engineering, and in particular Fluid Power, systems. It presents a new numerical integration method aimed at dealing with these difficulties. Some of the work that has preceeded this thesis [1] [2] [3] [4] provides a useful insight into the major problem areas that must be analysed and overcome.

## History Of Computer Simulation

**1.3** Computer simulation began over 30 years ago, following the advent of the analogue computer. After the advent of the digital computer, a large number of digital analogue simulators were written, which were programs using digital representations of sets of analogue elements, with these elements appearing as subroutines or functions. The users of these simulators were expected to write main segments linking the subroutines and functions in a specified manner. As time progressed, the sophistication of these simulators increased, and the systems that could be simulated became more complex. The integration methods employed also had to be capable of solving the more complicated differential equations arising, and, whereas very early simulators employed single low order methods such as Simpson's rule [3], later packages such as DYSAC (Digitally Simulated Analog Computer) [2] and HYBLOC [5] used fourth order Runge-Kutta and fifth order predictor-corrector methods respectively.

**1.4** Since digital simulations are repetitive in nature, the need for a general purpose simulation language arose. A number of these languages have been developed, and one of the first was termed MIMIC [6], which translates digital commands into appropriate machine code. The language incorporates a translator which automatically sorts the commands in a program into a particular calling sequence, suitable for the model subroutines which describe individual components of the system being simulated. FORTRAN statements are also acceptable in the language. MIMIC preceded other general purpose simulation languages which were created with the same principles in mind. In the 1970's, I.B.M. developed CSMP (Continuous System Modelling Program) [7] [8], which allows the user to enter coding statements for different operations such as numerical integration and differentiation. A specific example of one of these statements, termed a structure statement, is:

Y = INTGRL(IC,X)

which states that the output , Y, is obtained by integrating X, with Y at the initial time equal to IC. This name, INTGRL, defines the particular DEVICE, i.e. the particular function to be performed on the variable X, and is calling a specific subroutine. ACSL (Advanced Computer Simulation Language) [9] is similar to CSMP in its structure and operation, and both of these languages accept FORTRAN statements and employ a translator which sorts the calling statements for the complete simulation process. ACSL has a large library of subroutines which model commonly occurring engineering effects, with the user able to expand this library if this is necessary. These two languages also allow the user to decide which integration method is required, and a number of different algorithms are available. Table 1.1 presents an overview of the integration methods used by different simulation languages.

**1.5** Although the integration and other stages are taken out of the hands of the user with these, and other simulation languages, such as CSSL [10] and ISIM [11], it is still necessary for the user to design and code the simulation programs. To do this skilfully requires a specialist knowledge of both the language and the area in which it is to be applied, which is undesirable if the language is to be implemented as a practical and convenient tool to aid the user. The generality of these simulation languages is partially to blame for this.

**1.6** To take the computer programming and simulation coding out of the hands of the user, packages with automatic code generation facilities have been developed for particular industries. An example is a package that has been developed specifically for the simulation of chemical reactions which is called KISS (KInetics Simulation System) [12]. This interactively running package requires no code to be written by the user, although he must naturally have a knowledge of chemical terminology. The integrator employed by the system is a modified form of a generalised Runge-Kutta method developed by Kaps and Pentrop [13].

## Fluid Power Simulation

**1.7** Since computer simulation can save considerable time and money in the design and development of engineering systems, it was soon adopted into the field of hydraulics. The McDonnel Aircraft Corporation released a package for simulating hydraulic systems in 1977. The package consists of several programs which simulate different aspects of several generalised hydraulic systems, e.g. the program HYTRAN [14] which analyses hydraulic transients. Although these programs give more flexibility to the user, in that it is not necessary to write actual coding, they still require a knowledge of the program structure if additional information is to be included. A program called DSH (Digital Simulation of Hydraulics) [15] has been developed in West Germany, which is intended to have the versatility and degree of user-friendliness which allows an inexperienced user to simulate any hydraulic system. The user of the package defines the circuit to be simulated in terms of 'macro' or 'micro' words. The macro word defines a mathematical model which already exists in the package. A micro word defines a single mathematical operation. By defining a sequence of micro words, it is possible to represent a model not catered for by the basic package. The package consists of five programs controlled by a co-ordinating program. However there are major drawbacks with this package, the main ones being the static nature of the programs which constitute DSH, which tend to limit the types of models that can be written, and the user interface; users have to define information in terms of data fields, and this can lead to a long and complicated procedure, particularly for sophisticated hydraulic circuits. The work in this thesis is concerned with another special purpose simulation package, called HASP (Hydraulic Automatic Simulation Package), which has been designed in the Fluid Power Centre at the University of Bath. This package requires the user to construct a circuit diagram of the hydraulic system which is being investigated, and convert this to a computer block diagram representing the individual models in the system. This diagram defines the component models to be used in the simulation and is subsequently used to provide simple alphanumeric input to a program generator to form a computer simulation of the system.

**Hydraulic Automatic Simulation Package (HASP)**

**1.8** The Hydraulic Automatic Simulation Package has been developed to simulate the dynamic performance of hydraulic systems arising in the Fluid Power field. The package consists of a library of mathematical models, each representing discrete physical components of a Fluid Power system, together with a program generator. The component models form individual blocks that are placed in the required position by a program generator to represent a hydraulic circuit. The aims of HASP are to allow a user to specify an hydraulic circuit, and to simulate the system without any specialised knowledge of mathematical modelling, numerical methods, or of complex computational techniques. A detailed description of the HASP package is given in chapter 5

**1.9** At present, HASP considers only the solution of ordinary differential equations where the independent variable is time. This is termed 'lumped parameter' theory, where the parameters computed in any model are assumed uniform throughout. Although work has been done with the consideration of spatial variance in such parameters as pressure, flow or velocity by Skarbeck-Wazynski [16], partial differential equations present more problems than their o.d.e. counterparts. These problems generally lead to excessive computation times in the solution of the equations, and hence can severely restrict the use of a package modelling spatial variance.

## Mathematical Problems In Fluid Power Simulation

**1.10** Many difficulties arise in the simulation of Fluid Power systems, and the majority of these are mathematical in nature. In particular, difficulties are caused by four main branches of mathematical problems, these being:

- i) Mathematical stiffness, where the eigenvalues determining the solution of the differential equations differ greatly in magnitude.
- ii) Discontinuities, both in the solution variables and their derivatives.
- iii) Non-linearities in the differential equations formulated to describe model behaviour.
- iv) The oscillatory behaviour of the solution variables.

Physical non-linearities such as those presented by cavitation, stiction and actuators reaching the limits of their travel demonstrate how these problems can arise. Non-linear orifice equations and circuits with small pipe volumes can also lead to differential equations that must be dealt with carefully. A full investigation of the mathematical problems and some examples of when they arise are found in chapter 2.

**1.11** This thesis approaches the problems arising in the simulation of Fluid Power systems from a mathematical viewpoint. The HASP package originally employed Gear's integration method [49] to solve the systems of differential equations arising from the mathematical models. This is a routine which can use any of several multi-step methods, and which uses an automatic time step control procedure. This method is adequate for solving the majority of equations, but does not prove to be suitable for coping with problems that are either very stiff in their mathematical formulation, discontinuous or highly oscillatory. When applied to these problems, the integrator reverts to a low order method, and a very small time step, leading to excessive computer run-times, a factor that should be avoided with a viable interactive simulation package. Also, special modelling techniques must be employed in order to use Gear's method, when the difficiulties that have been outlined here arise. It is not only Gear's method that has difficulties with working in this enviroment; numerical methods are still being sought and developed that are better able to cope with the problems presented above. The failings of presently available numerical methods are discussed in detail in the next chapter.

## A New Integration Method

**1.12** The main work presented in this thesis is dedicated to the development and testing of a new integration method, and its subsequent implementation inside the HASP package as a versatile integrator. The new method is based on the analytical solution of a linear first order ordinary differential equation, an approach which was first suggested by Professor D.E. Bowns at the University of Bath [2]. It is a single step method, and differs from the classical integration techniques, such as Adam's-type methods. The new method was originally formulated as an explicit method, and from this followed an implicit form. Both the explicit and the implicit forms have good stability[1] properties, that make them immediately applicable to stiff systems of differential equations. These stability properties also ensure that the method enjoys considerable advantages over existing methods in solving problems which have diagonally dominant system matrices, a point which will be fully clarified. The implementation of the method inside of HASP is described, and this work involves special treatment of the mathematical models that describe the individual components of the hydraulic system being simulated. This is because the package has been designed for use with a classical integration method, and the information that a classical method requires for the solution of a differential equation differs from the data that the new method needs; this will be investigated in chapter 5.

## Existing Numerical Methods

**1.13** A large amount of research has been done in the field of numerical methods, particularly with regard to the solution of stiff ordinary differential equations. Some of the relevant methods are investigated, since it is important that the main requirements of a robust and competitive integrator are known before attempting to develop a new integrator. These methods include Adam's methods, Euler's method, Runge-Kutta methods and Gear's method. The necessary requirements, which will be discussed in chapter 2, include good stability properties and high accuracy. Also, an automatic time step control is essential. A package developed by Enright [17] also enables a user to compare constructively numerical methods, and work with this package has been undertaken in order to gain more information about the new method

1 The implication of a stable numerical method is that any error arising in the computed solution of an inherently stable system should decay as the solution advances in time. These errors inevitably arise, both from truncation error and round-off error, and so it is essential for a practical numerical method to have good stability properties.

being developed.

## Runge-Kutta And Switching Methods

**1.14** The study of the mathematical difficulties arising in Fluid Power simulation has led to several integration methods being used with the package, each of which is applied for a specific purpose. The application of Runge-Kutta methods to the HASP simulation package is studied in this thesis. In particular, various types of implicit Runge-Kutta methods are examined, since these are more applicable to stiff systems of differential equations. Work done by Butcher [18] and Norsett [19] has brought Runge-Kutta methods back to the forefront of modern numerical techniques, in the attempt to solve problems that require methods with very stringent stability properties, as well as high accuracy. The methods investigated have been termed diagonally implicit Runge-Kutta methods by Alexander [20]. Quite recently, work has been done by Petzold [21] with the automatic selection of methods for solving stiff and non-stiff systems of ordinary differential equations.

**1.15** The work presented by Petzold introduces the idea that a scheme which automatically determines whether to employ a class of methods suitable for non-stiff problems, or a class of methods suitable for stiff problems, throughout the integration of a problem, is more efficient in its implementation than simply using one single class of methods. The scheme that is discussed is able to switch from one class of methods to the other, depending on which it decides is the most suitable for the problem being solved. This idea has been used by Robertson [22] with explicit Runge-Kutta methods, and the ideas presented in chapter 8 suggest that it may be worthwhile to employ diagonally implicit Runge-Kutta methods as well.

**1.16** The implementation of a suitable integration method is very important for the HASP package, and this integrator may not be in the form of one method, but of several methods, together with a decision making routine that decides which method is most applicable for each particular problem. For full flexibility, it should be possible to change integration methods during a simulation in order to ensure that the most efficient integration method is always being employed.

## Programming And Software

**1.17** As well as the mathematical issues, the development of good, well-structured and well-written software is of great importance. At this point it should be stated that the HASP package is written exclusively in FORTRAN, and so the new integrator is also, not just to homogenise the system, but because FORTRAN is a high-level language particularly well-suited to numerical computation. The software must also be transportable, and so no machine-dependent code should be used. Furthermore, it is essential to be familiar with the machine on which the work is carried out, to ensure that the computer is a tool aiding the work, and not a handicap.

**1.18** It is important to emphasise further the way in which the programs are written. Since the integration method must be computationally efficient, it is essential that the software written to implement the method is also efficient. Consequently, the algorithms for the software must be well-structured and designed, and the software comprehensively tested. The programs written for the work presented in this thesis have been produced by a 'top-down' design [23]. Coding requires great care, but is a natural progression from the design stage, whereas the testing stage must ensure that the program is able to deal satisfactorily with all the data presented to it. Consequently, appropriate data must be devised in an attempt to ensure that all situations are handled correctly.

## Plan And Scope Of Thesis

The remainder of this chapter describes the course taken and the work covered by this thesis. The objectives of each chapter are presented, and a summary of the contents is given.

**1.18 Chapter 2** describes in detail the mathematical difficulties that arise in hydraulic simulation, and analyses some of the numerical methods that have been developed to cope with these difficulties. A definition of mathematical stiffness is given, and the reasons why it presents problems to numerical integration methods are explained. The ideas presented are used and expanded upon throughout the thesis, particularly in the development of the new method. The object of the chapter is to present an overview of the mathematical background behind Fluid Power simulation.

**1.19 Chapter 3** introduces the new integration method that is to be studied. The method is developed and subsequently tested on a set of test problems, each of which arises from a engineering circuit. The performance of the method in solving these problems is analysed, and compared with that of other numerical methods which have also been used to solve the same problems. The problems have been chosen since they demonstrate the mathematical difficulties described in chapter 2, and are hence useful examples on which to test a new numerical method.

**1.20 Chapter 4** gives an analysis of the new method. The local error of the method is examined, and from this is constructed a time step control, which ensures the higher computational efficiency of the method. The stability properties of the method are also investigated. From this analysis, a more robust, general purpose integration method is constructed. After performing the theoretical analysis, the method is applied to a set of problems, which have been chosen to illustrate the properties that are introduced earlier in the chapter.

**1.21 Chapter 5** introduces and explains the structure and workings of the HASP simulation package, with particular emphasis on the implementation of a numerical integration method inside the package. The evaluation of a Jacobian matrix using a perturbation technique is explained, and the generalisation of the method to allow its implementation is discussed. Having placed the integrator inside the package, dynamic simulations of Fluid Power systems are performed, and the method compared with the previous integrator employed by the package.

**1.22 Chapter 6** describes a testing package which has been designed to aid in the assessment of Initial Value methods for stiff systems of ordinary differential equations. The package has been used to test the new integration method, and has helped to determine the problem domain over which the method is suitable. The package consists of a collection of FORTRAN subroutines, combined with a canonical set of test problems. The problems have been chosen from different fields of Science and Technology, and cover all the mathematical problem areas that have been discussed in the previous chapters of this thesis, with a particular emphasis on stiffness.

**1.23 Chapter 7** presents a particular application of the new method. Since the method has been found to have beneficial stability properties when used to solve problems which have diagonally dominant system matrices, one area which will lead to this type of problem is examined. When parabolic partial differential equations are discretised to form sets of ordinary differential equations, then the resultant system matrices are often diagonally dominant, and the equations themselves very stiff. Present numerical methods that are used to solve these equations are examined, and compared with the new method which is also applied to the problem. The possibility of extending the new method is also discussed.

**1.24 Chapter 8** broadens the scope of the thesis, and introduces Runge-Kutta methods. These are an alternative set of single-step integration methods which have some desirable properties which may prove to be beneficial when used to solve the problems arising in Fluid Power. In particular, implicit Runge-Kutta methods are studied, and their potential in acting as an alternative integrator is discussed. As well as introducing Runge-Kutta methods, switching methods are also examined, and their possible application to the HASP package is contemplated. Suggestions are made which may prove to be rewarding if carried out. The work presented is a result of the author's study of Fluid Power simulation, and provides up to date methods which have been applied with some considerable success to certain problem areas similar to those found in hydraulics.

| Year | Simulation Language | Integration Methods |
|---|---|---|
| 1961 | DYSAC | FIXED STEP LENGTH:<br>RUNGE-KUTTA 4TH ORDER |
| 1965 | MIMIC | FIXED STEP LENGTH:<br>RUNGE-KUTTA 4TH ORDER |
| 1969 | HYBLOC | VARIABLE STEP LENGTH:<br>ADAMS ( 1ST,3RD, 5TH ORDER) |
| 1975 | ACSL | FIXED STEP LENGTH:<br>RUNGE-KUTTA (1ST, 2ND, 4TH ORDER)<br>VARIABLE STEP LENGTH:<br>ADAMS MOULTON, GEAR |
| 1976 | CSSL | FIXED STEP LENGTH:<br>EULER, TRAPEZOIDAL, ADAMS<br>VARIABLE STEP LENGTH:<br>RUNGE-KUTTA ( 4TH ORDER), ADAMS |
| 1978 | CSMP III | FIXED STEP LENGTH:<br>EULER, ADAMS, TRAPEZOIDAL,<br>SIMPSON, RUNGE-KUTTA (4TH ORDER)<br>VARIABLE STEP LENGTH:<br>RUNGE-KUTTA ( 4TH ORDER ), GEAR |
| 1981 | KISS | VARIABLE STEP LENGTH:<br>RUNGE-KUTTA ( 4TH ORDER ) |
| 1983 | ISIM | FIXED STEP LENGTH:<br>RUNGE-KUTTA (2ND AND 4TH ORDER)<br>VARIABLE STEP LENGTH:<br>RUNGE-KUTTA SARAFYAN |

**TABLE 1.1 INTEGRATION METHODS USED IN SIMULATION**

# CHAPTER 2

**DETAILED CONTENTS** **Page**

# CHAPTER 2

# MATHEMATICAL BACKGROUND TO THE SIMULATION OF FLUID POWER SYSTEMS

## Introduction

**2.1** The objective of this chapter is to view in detail the possible mathematical difficulties that arise in hydraulic simulation, and to analyse some of the numerical methods that have been developed to cope with these difficulties in the HASP package, prior to the work covered by this thesis. The ideas met form a basis for the work covered in the subsequent chapters. In particular, an expository review of the problem of stiffness in the numerical solution of ordinary differential equations is presented, with special emphasis on stability aspects.

2.2 The work covered is as follows:

Mathematical stiffness is rigorously defined and two hydraulic circuits where stiffness occurs are examined. Some of the present numerical techniques that are available for solving stiff differential equations are discussed and an explanation of the derivation of multi-step and single-step methods is given. An accompanying analysis of the stability, consistency and accuracy of these methods is made, since this work will be relevant when developing the integration method described in chapters 3 and 4. A practical definition of stiffness is also shown and an explanation of the problems it causes is given. The difficulties caused by discontinuities, non-linearities and oscillatory problems are discussed and practical examples of their occurrences in Fluid Power systems are shown. Finally, the application of a predictor-corrector pair as a numerical integrator is explained, and the resultant iteration schemes suitable for stiff systems of equations are examined.

## Mathematical Stiffness

**2.3** At this stage, a mathematically stiff system is defined; later a more in-depth approach to the problems it causes will be taken. The problem of stiffness has been known for some time and was first investigated by Curtiss & Kirschfelder [24]. Briefly, a stiff system is one whose dynamic behaviour is described by a set of coupled differential equations which have solutions with widely differing decay rates. The rates at which the solutions decay are determined by the eigenvalues of the appropriate system matrix.

Consequently, for the Initial Value Problem

$$\underline{y}' = A\underline{y} + \underline{b}(t) \quad y_k(0) = \beta_k \quad k=1,\dots,m$$

where A is a constant (mxm) matrix with constant eigenvalues given by:

$$\lambda_k = \mu_k + j\omega_k \quad k=1,\dots,m$$

and the solution, if the eigenvalues are distinct, is given by:

$$y_k(t) = \alpha_{1_k}e^{\lambda_1 t}+\alpha_{2_k}e^{\lambda_2 t}+\cdots+\alpha_{m_k}e^{\lambda_m t}+F_k(t) \quad k=1,\dots,m \tag{2.1}$$

then the system is said to be stiff if

i) $\mu_k<0 \quad k=1,\dots,m$

ii) $\max_k|\mu_k| \quad >> \quad \min_k|\mu_k|$

The stiffness ratio, S is defined by:

$$S = \frac{\max_k|\mu_k|}{\min_k|\mu_k|} \tag{2.2}$$

**2.4** The real parts of the eigenvalues are taken as it is these that govern the decay rate of the components. The imaginary parts of the eigenvalues are associated with the oscillatory (non-decaying) part of the solution. For the problem

$$\underline{y}' = A(t)\underline{y} + \underline{b}(t) \quad y_k(0)=\beta_k \quad k=1,\dots,m$$

where the eigenvalues of A(t) will be time-dependent, and the solution is given by a similar expression to that in eqn (2.1), then the system is said to be stiff in a time interval, if both i) and ii) apply for t in that interval.

**2.5** Most stability requirements for numerical methods place constraints on a combination of the time step used and the eigenvalues involved, e.g. Euler's method, dealt with later, requires that $|1 + h\lambda| < 1$ to ensure stability. As a consequence, for a stiff system, although interest may not be with the components in the system corresponding to the eigenvalues largest in magnitude, i.e. the smallest time-constants, in order to satisfy the stability requirements for a classical integration method, it may be imperative to choose a very small time step because of these large eigenvalues.

2.6 The temptation is to think that the time step need only be restricted to a small value whilst the component corresponding to the eigenvalue largest in magnitude is significant. This, however, is not the case, the stability restriction must be observed throughout the computation whilst using one particular integration method. It is this property that creates the problem with mathematical stiffness, the time steps required by classical methods for the solution of stiff systems can be extremely small, resulting in long computer execution times.

## Practical Stiffness

2.7 Practically, stiffness occurs as a problem in the solution of a system of differential equations when less computational effort is required to use an implicit method than an explicit method. Implicit methods, which will in general allow a larger time step than explicit methods in order to ensure stability, normally require more work at each step to form a solution than explicit methods. Explicit methods, however, require small time steps if the problem being solved is stiff, so as to satisfy the stability requirements. Mathematical stiffness occurs in the modelling of many areas of Science and Technology; Physics and Chemical Engineering being two such fields. Although in this present context the author is confining the study to Fluid Power systems, the work is applicable in many other areas.

Two practical examples are now given to illustrate how mathematical stiffness can occur in hydraulic system simulation.

**Two Examples To Show The Occurrence Of Mathematical Stiffness**

**2.8 Example 1 - Open loop transmission system.** Consider the open loop hydrostatic system, shown in Figure 2.1, in which the hydraulic motor is supplied with fluid to drive a rotational load comprising an inertia, with viscous friction and a constant applied force. There is a relief valve in the circuit that is used to limit the system pressure. The pump and motor are assumed to exhibit no slip flow or torque losses in order to simplify the analysis.

The differential equations governing the behaviour of the system are:

$$\frac{dP}{dt} = \frac{B}{v}(Q_p - Q_r - Q_m) \tag{2.3}$$

and

$$\frac{d\omega_m}{dt} = \frac{(T_m - f\omega_m - T_l)}{J} \tag{2.4}$$

where:

P is the fluid pressure in the pipe
$Q_p$ is the pump flow rate

$Q_m$ is the motor flow rate
$Q_r$ is the relief valve flow rate
B is the effective bulk modulus of the system
v is the pipe volume
J is the moment of inertia
$\omega_m$ is the angular velocity of the motor
f is the viscous friction coefficient
$T_m$ is the motor torque
$T_l$ is the load torque

2.9 The pump flow rate is given by:

$$Q_p = D_p\omega_p \tag{2.5}$$

where:

$D_p$ is the pump displacement
$\omega_p$ is the pump shaft angular velocity

The motor flow rate is given by:

$$Q_m = D_m\omega_m \tag{2.6}$$

where $D_m$ is the motor displacement

The torque developed by the hydraulic motor is given by :

$$T_m = D_mP \tag{2.7}$$

If the flow rate through the relief valve is non-zero, owing to the system pressure exceeding the relief valve cracking pressure, then if the relief valve is assumed to open instantaneously, the flow rate through the valve is given by:

$$Q_r = k_r(P - P_c) \tag{2.8}$$

where:

$P_c$ is the cracking pressure
$k_r$ is the flow coefficient of the relief valve.

Consequently, the differential equations for the rates of change of pipe pressure and angular velocity of the rotary load, can be written, with the relevant substitutions, as:

$$\frac{dP}{dt} = \frac{B}{V}(D_p\omega_p - k_r(P - P_c) - D_m\omega_m) \tag{2.9}$$

$$\frac{d\omega_m}{dt} = \frac{1}{J}(D_mP - f\omega_m - T_l) \tag{2.10}$$

which leads to the 2x2 matrix equation

$$\begin{bmatrix} \frac{dP}{dt} \\ \frac{d\omega_m}{dt} \end{bmatrix} = \begin{bmatrix} -k_r\frac{B}{V} & \frac{-BD_m}{V} \\ \frac{D_m}{J} & \frac{-f}{J} \end{bmatrix} \begin{bmatrix} P \\ \omega_m \end{bmatrix} + \begin{bmatrix} \frac{B}{V}(D_p\omega_p + k_rP_c) \\ \frac{-T_l}{J} \end{bmatrix} \tag{2.11}$$

This is a matrix differential equation for which the general form is

$$\underline{y}' = A\underline{y} + \underline{b} \tag{2.12}$$

where $\underline{b}$ is the forcing function.

2.10 In order to see how this system leads to the problem of mathematical stiffness it is necessary to examine the eigenvalues of the A matrix. The eigenvalues, the reciprocals of the time-constants in the real case, of the set of differential equations given by equation (2.12), are the solutions of the equation

$$(A - \lambda I)\underline{x} = 0 \tag{2.13}$$

where I is the unit matrix. The eigenvalues can be evaluated from the determinant equation [25].

$$| A - \lambda I | = 0$$

The characteristic equation for the system given by equation (2.11) is:

$$\det \begin{vmatrix} \dfrac{-k_rB}{v} - \lambda & \dfrac{-BD_m}{v} \\ \dfrac{D_m}{J} & -\dfrac{f}{J} - \lambda \end{vmatrix} = 0 \tag{2.14}$$

which leads to a quadratic in λ of the form:

$$\left(\frac{-k_rB}{v} - \lambda\right)\left(-\frac{f}{J} - \lambda\right) + \frac{BD_m}{v}\left(\frac{D_m}{J}\right) = 0 \tag{2.15}$$

which simplifies to:

$$\lambda^2 + \left(\frac{f}{J} + \frac{k_rB}{v}\right)\lambda + \frac{B}{vJ}\left(D_m^2 + k_rf\right) \tag{2.16}$$

Applying the formula for the solution of a quadratic yields:

$$\lambda = -\left(\frac{f}{2J} + \frac{k_rB}{2v}\right) \pm \frac{1}{2}\left(\left(\frac{f}{J} + \frac{k_rB}{v}\right)^2 - \frac{4B}{vJ}(D_m^2 + k_rf)\right)^{0.5}$$

which simplifies to:

$$\lambda = -\left(\frac{f}{2J} + \frac{k_rB}{2v}\right) \pm \frac{1}{2}\left(\left(\frac{k_rB}{v}\right)^2 - \frac{B}{vJ}(2k_rf + 4D_m^2) + \frac{f^2}{J^2}\right)^{0.5} \tag{2.17}$$

If

$$\frac{f^2}{J^2} + \left(\frac{k_rB}{v}\right)^2 >> \frac{B}{2J}(2k_rf + 4D_m^2)$$

or equivalently

$$\frac{f^2v}{J^2B} + \frac{k_r^2B}{v} >> \frac{v}{J}(2k_rf + 4D_m^2)$$

and provided that

$$\left|\frac{f}{2J} + \frac{k_rB}{2v}\right| > \frac{1}{2}\left(\frac{k_r^2B^2}{v^2} - \frac{B}{vJ}(2k_rf + 4D_m^2) + \frac{f^2}{J^2}\right)^{0.5} \tag{2.18}$$

then the eigenvalues of the system will be real, negative and widely separated in magnitude. Consequently a stiff system is apparent, with the stiffness ratio S given by:

$$S = \frac{-\left[\frac{f}{2J} + \frac{k_rB}{2v}\right] - \frac{1}{2}\left[\frac{k_r^2B^2}{v^2} - \frac{B}{vJ}(2k_rf + 4D_m^2) + \frac{f^2}{J^2}\right]^{0.5}}{-\left[\frac{f}{2J} + \frac{k_rB}{2v}\right] + \frac{1}{2}\left[\frac{k_r^2B^2}{v^2} - \frac{B}{vJ}(2k_rf + 4D_m^2) + \frac{f^2}{J^2}\right]^{0.5}} \qquad (2.19)$$

**2.11** To further study this expression, the relative magnitudes of the quantities appearing in this expression for typical hydraulic circuits can be considered. The bulk modulus, B, and the inertia, J, are usually very large. The viscous friction, f, the motor displacement, $D_m$, and the pipe volume, v, are usually small quantities. Typical data are:

$B = 1.4 \times 10^9$ N/m$^2$
$f = 1$ N/sec/rad
$v = 1 \times 10^{-4}$ m$^3$
$D_m = 2.5 \times 10^1$ cc/rev
$J = 0.5$ kgm$^2$
$k_r = 5. \times 10^{-11}$ m$^3$/sec/N/m$^2$

This will lead to a stiffness ratio of

$S \approx 4.0 \times 10^3$

**2.12 Example 2 - Hydraulic actuator circuit.** The circuit shown in Figure 2.2 consists of a fixed displacement pump supplying fluid to a linear actuator by a ' meter in ' flow control orifice. The directional control valve is manually operated and allows the actuator to be extended, retracted or held stationary. When the valve is centred, flow returns to tank through a relief valve. Mathematical stiffness occurs in the circuit at the start of a simulation since the pressure differential across the orifice is small and there is a high flow gain of the orifice under this condition.

**2.13 Orifice restrictor.** The flow rate through an orifice with potential flow is given by the relationship :

$$Q_0 = k_0\sqrt{\Delta P} \tag{2.20}$$

where:

$k_0$ is the orifice flow coefficient
$\Delta P$ is the differential pressure across the orifice

The gradient of this function is :

$$\frac{dQ_0}{d\Delta P} = \frac{k_0}{2\sqrt{\Delta P}} \tag{2.21}$$

When the differential pressure is zero, the gradient of this function is infinite. As the orifice connects two sections of pipe, the zero differential pressure condition can give rise to infinite stiffness. No classical integration method can provide an adequate simulation of this condition as can be seen by considering the equation set for that part of the linear actuator circuit shown in Figure 2.3.

**2.14** Considering the actuator system in this simplified form, the model relationships to describe the system may be written as:

$$\begin{bmatrix} \dfrac{dP_1}{dt} \\ \dfrac{du}{dt} \end{bmatrix} = \begin{bmatrix} \dfrac{-Bk_0}{v\sqrt{\Delta P}} & -\dfrac{A_rB}{v} \\ \dfrac{A_r}{M} & \dfrac{-f}{M} \end{bmatrix} \begin{bmatrix} P_1 \\ u \end{bmatrix} + \begin{bmatrix} \dfrac{Bk_0P_s}{v\sqrt{\Delta P}} \\ 0 \end{bmatrix} \tag{2.22}$$

where:

$A_r$ is the area of the actuator piston
B is the effective bulk modulus of the system
f is the viscous friction coefficient
M is the mass of the load
$P_1$ is the pressure downstream of the orifice
$P_s$ is a constant pressure upstream of the orifice
u is the actuator velocity
v is the volume

The eigenvalues of the system are given by:

$$\det \begin{bmatrix} \dfrac{-Bk_0}{v\sqrt{\Delta P}} - \lambda & \dfrac{-A_r B}{v} \\ \dfrac{A_r}{M} & -\dfrac{f}{M} - \lambda \end{bmatrix} = 0 \tag{2.23}$$

which becomes:

$$\left( \frac{Bk_0}{v\sqrt{\Delta P}} + \lambda \right) \left( \frac{f}{M} + \lambda \right) + \frac{A_r^2 B}{Mv} = 0 \tag{2.24}$$

$\lambda$ is given by:

$$\lambda = -\left( \frac{f}{2M} + \frac{Bk_0}{2v\sqrt{\Delta P}} \right) \pm \frac{1}{2} \left( \left( \frac{f}{M} + \frac{Bk_0}{v\sqrt{\Delta P}} \right)^2 - \frac{4B}{vM} \left( A_r^2 + \frac{k_0 f}{\sqrt{\Delta P}} \right) \right)^{0.5} \tag{2.25}$$

If

$$\left( \frac{f}{M} + \frac{Bk_0}{v\sqrt{\Delta P}} \right)^2 >> \frac{4B}{vM} \left( A_r^2 + \frac{k_0 f}{\sqrt{\Delta P}} \right)$$

and

$$\left| \frac{f}{2M} + \frac{Bk_0}{v\sqrt{\Delta P}} \right| > \frac{1}{2} \left( \left( \frac{f}{M} + \frac{Bk_0}{v\sqrt{\Delta P}} \right)^2 - \frac{4B}{v} M \left( A_r^2 + \frac{k_0 f}{\sqrt{\Delta P}} \right) \right)^{0.5}$$

then the eigenvalues will be real, negative and widely separated in magnitude. The stiffness ratio, S, is given by:

$$S = \frac{-\left(\frac{f}{2M} + \frac{Bk_0}{2v\sqrt{\Delta P}}\right) - \frac{1}{2}\left(\left(\frac{f}{M} + \frac{Bk_0}{v\sqrt{\Delta P}}\right)^2 - \frac{4B}{vM}\left(A_r^2 + \frac{k_0 f}{\sqrt{\Delta P}}\right)\right)^{0.5}}{-\left(\frac{f}{2M} + \frac{Bk_0}{2v\sqrt{\Delta P}}\right) + \frac{1}{2}\left(\left(\frac{f}{M} + \frac{Bk_0}{v\sqrt{\Delta P}}\right)^2 - \frac{4B}{vM}\left(A_r^2 + \frac{k_0 f}{\sqrt{\Delta P}}\right)\right)^{0.5}}$$

For the data:

$B = 1.4 \times 10^9$ N/m$^2$
$f = 1 \times 10^2$ N/sec/m
$v = 1 \times 10^{-2}$ m$^3$
$M = 10$ Kg
$k_0 = 5 \times 10^{-2}$ m$^3$/sec/N/m$^2$
$\Delta P = 1 \times 10^{-6}$ N/m$^2$
$A_r = 5 \times 10^{-2}$ m$^2$

then the stiffness ratio is of the order

$S \approx 1.0 \times 10^9$

**2.15** These two examples, each comprising of a (2x2) set of coupled differential equations, demonstrate how stiffness can be introduced into a system simulation. In these two examples it was introduced by the model of a hydraulic relief valve and the model of an orifice. Although this type of problem is a recurrent one, the modelling of the effects that can cause stiffness must be included to ensure that the equations accurately represent the hydraulic circuit.

## Numerical Integration Methods

**2.16** A brief explanation of currently available numerical integration methods and the mathematical theory behind them is now given, since the issues discussed will be relevant when developing a new integration method.

**Initial Value Problems For First Order Ordinary Differential Equations**

**2.17** A first order differential equation $y' = f(t,y)$ can possess an infinite number of solutions. For example $y(t) = Ce^{\lambda t}$ is, for any value of the constant C, a solution of the differential equation $y' = \lambda y$, where $\lambda$ is a given constant. Under certain conditions, $y' = f(t,y)$ has a unique solution if an initial condition is specified. For example, with the problem $y' = \lambda y$, if $y(\alpha) = \beta$, then the particular solution satisfying the initial condition is given by $y(t) = \beta e^{\lambda(t-\alpha)}$. The differential equation, together with an initial condition, is said to constitute an initial value problem, which is:

$$y' = f(t,y) \qquad y(\alpha) = \beta \tag{2.26}$$

The following theorem, the proof of which can be found in Henrici [26], states conditions on $f(t,y)$ which guarantee the existence of a unique solution of the initial value problem given in equation (2.26).

**2.18 Theorem 1.** Let $f(t,y)$ be defined and continuous for all points $(t,y)$ in the region $D_f$ defined by $\alpha \leqslant t \leqslant \gamma$, $-\infty < y < \infty$, $\alpha$ and $\gamma$ finite, and let there exist a constant L such that, for every $t$, $y$, $y^*$ such that $(t,y)$ and $(t,y^*)$ are both in $D_f$

$$| f(t,y) - f(t,y^*) | \leqslant L | y - y^* | \tag{2.27}$$

Then, if $\beta$ is any given number, there exists a unique solution $y(t)$ of the initial value problem (2.26), where $y(t)$ is continuous and differentiable for all $(t,y)$ in $D_f$

**2.19 Lipschitz condition.** The requirement in equation (2.27) is known as a Lipschitz condition, and the constant L as a Lipschitz constant. Two properties follow from $f(t,y)$ being continuously differentiable [27], viz:

$f(t,y)$ being continuously differentiable with respect to y for all $(t,y)$ in $D_f$
implies that $f(t,y)$ satisfies a Lipschitz condition w.r.t. y for all $(t,y)$ in $D_f$

and this implies that $f(t,y)$ is continuous w.r.t. $y$ for all $(t,y)$ in $D_f$

In particular, if $f(t,y)$ possesses a continuous derivative with respect to $y$ for all $(t,y)$ in $D_f$, then, using the Mean Value theorem,

$$f(t,y) - f(t,y^*) = \frac{\partial f(t,y)}{\partial y}(y - y^*)$$

where $y$ is a point in the interior of the interval whose end points are $y$ and $y^*$, and $(t,y)$ and $(t,y^*)$ are both in $D_f$. The Lipschitz condition in equation (2.27) is satisfied if $L$ is chosen as

$$L = \sup\left|\frac{\partial f(t,y)}{\partial y}\right| \quad \text{for } (t,y) \text{ in } D_f \tag{2.28}$$

**Initial Value Problems For Systems Of First Order Differential Equations**

**2.20** In Fluid Power simulation, then, rather than a single differential equation arising, systems of m simultaneous first order equations in m dependent variables $y_1, y_2, y_3, y_4, \dots, y_m$ arise. If each of these variables satisfies a given condition at the same value $\alpha$ of time, then an initial value problem for a first order system arises. This problem can be written as:

$$y_1 = f_1(t, y_1, y_2, y_3, \dots, y_m) \quad y_1(\alpha) = \beta_1$$
$$y_2 = f_2(t, y_1, y_2, y_3, \dots, y_m) \quad y_2(\alpha) = \beta_2$$
$$y_3 = f_3(t, y_1, y_2, y_3, \dots, y_m) \quad y_3(\alpha) = \beta_3$$

. .

. .

. .

$$y_m = f_m(t, y_1, y_2, y_3, \dots, y_m) \quad y_m(\alpha) = \beta_m$$

Introducing the vector notation

$$\underline{y} = [y_1, y_2, y_3, \ldots, y_m]^T$$

$$\underline{f} = [f_1, f_2, f_3, \ldots, f_m]^T$$

$$\underline{\beta} = [\beta_1, \beta_2, \beta_3, \ldots, \beta_m]^T$$

then the I.V.P. can be written in the form:

$$\underline{y}' = \underline{f}(t, \underline{y}) \quad \underline{y}(\alpha) = \underline{\beta} \tag{2.29}$$

The reason for looking at systems is to show how the theorem of uniqueness and existence extends from a scalar equation, through to a set of equations. In order for theorem 1 to be applicable and give the necessary conditions for the existence of unique solution to equation (2.29) then two changes to the conditions of the theorem must be made. These are:

i) The region $D_f$ must be defined by $\alpha \leqslant t \leqslant \gamma$, $-\infty < y_i < \infty$, $i = 1,2,\ldots,m$

ii) Condition (2.27) must be replaced by:

$$\| \underline{f}(t,\underline{y}) - \underline{f}(t,\underline{y}^*) \| \leqslant L \| \underline{y} - \underline{y}^* \| \tag{2.30}$$

where $(t,\underline{y})$ and $(t,\underline{y}^*)$ are in $D_f$, and $\| \, . \, \|$ denotes a vector norm [47].

In the case where each of the $f_i(t,y_1,y_2,y_3,\ldots,y_m)$, $i = 1,2,\ldots,m$, possesses a continuous derivative with respect to each of the $y_j$, $j = 1,2,\ldots,m$ then, analogous to equation (2.28)

$$L = \sup \left\| \frac{\partial \underline{f}}{\partial \underline{y}} \right\| \quad \text{for } (t,\underline{y}) \text{ in } D_f$$

where $\frac{\partial \underline{f}}{\partial \underline{y}}$ is the Jacobian of $\underline{f}$ with respect to $\underline{y}$ [29].

**2.21 Reduction of high order differential equations to first order.** Frequently, the ordinary differential equations that are formulated to describe physical situations are of second order or higher. In this case, in their solution, either an integration method specifically designed for solving higher order equations must be used, or the differential equations must be reduced to a set of first order equations. This is done by making a substitution into the higher order set of equations, and solving for a new set of unknowns. To reduce the second order differential equation

$$\frac{d^2y}{dt^2} + K_1\frac{dy}{dt} + K_2y = K_3 \tag{2.31}$$

to two first order differential equations, the substitution $\frac{dy}{dt} = r$ can be made, then equation (2.31) will become the system

$$\begin{aligned} \frac{dy}{dt} &= r \\ \frac{dr}{dt} + K_1r + K_2y &= K_3 \end{aligned} \tag{2.32}$$

and now a classical integration technique can be applied . This method of reducing equations can be used on higher order differential equations.

## Direction Fields

2.22 The function y(t), the solution to equation (2.26), is a curve in the ty-plane,and, although it may not be possible to find y(t) exactly, the slope of y(t) at every point on its solution curve is known. If the solution passes through the point (t,y), then since $y' = f(t,y)$, the slope of the tangent line to the curve y(t) at the point (t,y) is given by f(t,y). Consequently, the direction of the solution curve y(t) at any point in the ty-plane is known. The set of all these directions in the plane is called the Direction Field of the differential equation $y' = f(t,y)$. In many cases the solution to a differential equation, although not computed, can be sketched via its direction field.

**2.23 Example to demonstrate the occurrence of direction fields.** Considering the initial value problem

$$y' = 2ty \quad y(0) = 1$$

then

$$y'(t) > 0 \text{ if } ty > 0$$

and

$$y'(t) < 0 \text{ if } ty < 0$$

Hence $y'(t) > 0$ in the first and third quadrants, and $y'(t) < 0$ in the second and fourth quadrants. The direction field is sketched in Figure 2.4 . Since t and y are positive in the first quadrant, the slopes of the tangent lines to any solution curve are positive, so that the solution curves increase and become steeper as t and y become larger. Along the axes the solution is flat because the derivative is zero. In the second quadrant the slopes of the tangent lines are negative since $y' < 0$ . Similar conditions apply in the third and fourth quadrants.

**2.24** The solution curve must satisfy the initial condition $y(0) = 1$ for the particular I.V.P. being considered and must consequently pass through the point (0,1). The sketch of the solution given in Figure 2.5 is indicative of the way in which the information is used to formulate an estimate of the solution. The problem being solved has the true solution $y(t) = e^{t^2}$, and the correlation between this and the sketched solution is easy to see.

## Numerical Methods

2.25 Numerical methods use the direction fields of a differential equation in order to form their approximation to the solution. Numerical techniques exist since, for the majority of differential equations, there is no known solution, i.e. it is impossible to express a solution in terms of elementary functions. They seek to find an approximation to y(t) for one or more values of the independent variable t, rather than looking for a function y(t) that solves the problem at every value of t. Numerical methods are also used, when techniques developed to find the analytical solution to differential equations, require significantly more effort to form an answer. An example of this is the computational labour involved in solving exactly a system of many simultaneous first order differential equations, which may be formidable [30].

2.26 The number of numerical methods available to solve differential equations is legion, and a comprehensive study is not made here. However, the most common methods that are used for solving differential equations fall into categories, and these will be briefly discussed. The two classes are:

1) Multi-step methods
2) Single-step methods

## Multi-step Methods

2.27 **The general linear multi-step method.** Considering the I.V.P. for a single first order differential equation

$$y' = f(t,y) \quad y(a) = \delta \tag{2.33}$$

A solution is required in the range $a \leqslant t \leqslant b$, where a and b are finite. It is assumed that f satisfies the conditions stated in theorem 1 on uniqueness and existence, and so the problem has a unique continuously differentiable solution described by y(t). Considering the sequence of points $t_n = a + nh$, $n = 0,1,2,...$ where the parameter h, which for the present will be regarded as constant, is called the steplength, then it is possible to discretize the interval [a,b]. As has been already mentioned, numerical methods seek to approximate the solution on this point set, and not on the continuous interval $a \leqslant t \leqslant b$. Letting $y_n$ be an approximation to the true solution $y(t_n)$, at $t_n$ and $f_n = f(t_n, y_n)$, then if a method for determining the sequence $y_0, y_1, y_2, \cdots$ is formed by a linear relationship between $y_{n+j}$, $f_{n+j}$, $j = 0,1,2,...k$ it is termed a linear k-step method [31].

2.28 The general linear multi-step method is written as:

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f_{n+j} \tag{2.34}$$

where $\alpha_j$, $\beta_j$ are constants, $\alpha_k \neq 0$ and at least one of $\alpha_0$ and $\beta_0$ are non-zero. Furthermore, $\alpha_k = 1$, without loss of generality, and this ensures the uniqueness of the method.

2.29 Hence the problem is to find the sequence $y_0, y_1, y_2, \cdots$ that satisfies the difference equation (2.34). Since $f_n$ is, in general, a non-linear function of $y_n$, equation (2.34) is a non-linear difference equation. The sequence $y_0, y_1, y_2, \cdots$ is computed numerically, and in order to do this, a set of starting values $y_0, y_1, y_2, ..., y_{k-1}$ must be supplied. Methods for obtaining starting values are explained by Hall & Watt [32]. The method given by equation (2.34) is explicit if $\beta_k = 0$, and implicit if $\beta_k \neq 0$. For an explicit method, equation (2.34) gives the current value $y_{n+k}$ directly in terms of $y_{n+j}$, $f_{n+j}$, $j = 0,1,...,k-1$ which are already known. However, for an implicit method, to find $y_{n+k}$ will require at each step the solution of the equation

$$y_{n+k} = h\beta_k f_{n+k} + K \qquad (2.35)$$

where

$$K = \sum_{j=0}^{k-1} (h\beta_j f_{n+j} - \alpha_j y_{n+j})$$

is a known function of previously calculated values.

**2.30** When the original differential equation is linear, then equation (2.35) is also linear in $y_{n+k}$, and is easy to solve. If f is non-linear, then equation (2.35) must be solved by an iterative technique, and a typical fixed-point iteration scheme is of the form:

$$y_{n+k}^{(s+1)} = h\beta_k f(t_{n+k}, y_{n+k}^{(s)}) + K \quad s=0,1,2,... \qquad (2.36)$$

This process will only converge to the unique true solution for $y_{n+k}$ if the right-hand side of equation (2.36) satisfies a Lipschitz condition, i.e.

$$| h\beta_k f(t,y) - h\beta_k f(t,y^*) | \leqslant M_y | y - y^* | \quad 0 \leqslant M_y < 1 \qquad (2.37)$$

where $M_y$ is the Lipschitz constant w.r.to $y_{n+k}$.

**2.31** If the Lipschitz constant of f with respect to y is L, then $M_y$ will have the value $Lh|\beta_k|$, and so a unique solution for $y_{n+k}$ exists, and the above iteration converges to it, if

$$h < \frac{1}{L|\beta_k|}$$

Obviously if L is very large, then this requirement can impose a severe restricition on the size of the steplength. Stiff systems can lead to very large values of L, so the reason for examining iteration schemes that are suitable for stiff problems is seen. The ideas here follow through to a system of simultaneous non-linear equations. These, in the case of an implicit method, can also be solved by iteration, by forming a series of vector iterates. In order tc ensure the convergence of the iteration scheme, the absolute values of scalars are replaced by the norms of the corresponding vectors [33].

2.32 The coefficients $\alpha_j, \beta_j$ in equation (2.34) can be derived in various ways depending on the requirements placed on the method that will formed. Two such approaches are via Taylor's Expansions and numerical integration.

**2.33 Taylor's series expansions.** For small h, $y(t_n+h)$ can be expanded by a Taylor's series about the point $t_n$ as follows:

$$y(t_n+h) = y(t_n) + hy''(t_n) + \frac{h^2}{2!}y''(t_n) + \frac{h^3}{3!}y'''(t_n) + \cdots$$

where

$$y'(t_n) = \frac{dy}{dt}\,|_{t=t_n}, \; y''(t_n) = \frac{d^2y}{dt^2}\,|_{t=t_n}, \; \cdots$$

Truncating the Taylor's series after two terms and substituting for $y'(t_n)$ from the differential equation (2.33) gives:

$$y(t_n+h) \approx y(t_n) + hf(t_n, y(t_n)) \tag{2.38}$$

The error involved in estimating the solution to $y(t_n + h)$ is given by:

$$\frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + \cdots \tag{2.39}$$

Equation (2.38) gives a relation between the exact values of the solution of equation (2.33). Replacing $y(t_n)$ by $y_n$ and $y(t_n+h)$ by $y_{n+1}$ leaves an exact relation between approximate values of the solution of equation (2.33), and this is:

$$y_{n+1} = y_n + hf_n \tag{2.40}$$

This is known as Euler's method, the error in evaluating one step is given by the expression (2.39) and is called the local error. This error is of order $h^2$, and will be zero if the solution of equation (2.33) is a polynomial of degree 0 or 1, since y'', y''', $\cdots$ will all be zero. Geometrically, the meaning of Euler's method is shown in Figure 2.6, where the smooth curve is taken as the unknown exact solution of equation (2.33), which is being approximated by the broken line. In order to ensure that Euler's method accurately follows the true solution curve, the value of the steplength h must be restricted. This restriction is problem dependent. As a further note, Euler's method is a useful method with which to illustrate stability, convergence and accuracy analysis.

**2.34 Numerical integration.** Considering the identity

$$y(t_{n+2}) - y(t_n) = \int_{t_n}^{t_{n+2}} y'(t)dt \qquad (2.41)$$

y'(t) may be replaced by f(t,y) using the differential equation (2.33). If a linear two-step method is to be derived, then the data available will be $f_n, f_{n+1}, f_{n+2}$. Letting P(t) be the unique polynomial of degree two that passes through the three points $(t_n,f_n)$, $(t_{n+1},f_{n+1})$, $(t_{n+2},f_{n+2})$, then using the Newton-Gregory interpolation formula [34], given by:

$$P(t) = P(t_n+rh) = f_n + r\Delta f_n + \frac{r(r-1)}{2!}\Delta^2 f_n$$

where:

$$\Delta f_n = f_{n+1} - f_n, \ \Delta^2 f_n = f_{n+2} - 2f_{n+1} + f_n$$

the integrand in equation (2.41) becomes:

$$\int_{t_n}^{t_{n+2}} y'(t)dt \approx \int_0^2 [f_n + r\Delta f_n + \frac{r(r-1)}{2}\Delta^2 f_n]h\, dr$$

$$= h(2f_n + 2\Delta f_n + \frac{1}{3}\Delta^2 f_n) \qquad (2.42)$$

Expanding $\Delta f_n$ and $\Delta^2 f_n$ and substituting in equation (2.41) gives:

$$y_{n+2} - y_n = \frac{h}{3}(f_{n+2} + 4f_{n+1} + f_n)$$

which is know as Simpson's rule. This method is used in chapter 7 in an attempt to improve the accuracy of the new integration method that is developed in chapter 3.

**2.35 Convergence.** Many integration methods can be formulated by these approaches, some of which will be discussed later. However, before a formula can be used it must satisfy certain criteria. One of these is the property of convergence, which requires that the solution $y_1, y_2, y_3, \ldots$, generated by the method, converges to the theoretical solution $y(t)$ as the steplength h tends to zero. As a precise definition, then:

A method is said to convergent if, for all initial value problems of the form of equation (2.26), subject to the hypothesis of theorem 1, then the condition

$$y_n \to y(t_n) \text{ as } h \to 0, \quad nh = t-a$$

holds for all t in the interval [a,b]

**2.36 Order and error constant for linear multi-step methods.** For the linear multi-step method given by equation (2.34), then an operator L can be defined by: [35]

$$L[y(t);h] = \sum_{j=0}^{k}[\alpha_j y(t + jh) - h\beta_j y'(t + jh)] \tag{2.43}$$

where $y(t)$ is now an arbitrary function that is continuously differentiable on the interval [a,b]. The order of accuracy of the operator and of the associated linear multi-step method can now be defined. Expanding the test function $y(t+jh)$ and its derivative $y'(t+jh)$ using Taylor's series about the point t, and collecting terms together yields:

$$L[y(t);h] = C_0 y(t) + C_1 h y'(t) + C_2 h^2 y''(t) + \ldots + C_q h^q y^{(q)}(t) + \ldots \qquad (2.44)$$

where the $C_q$, with $q = 1,2,3,\ldots$ are constants.

**2.37 Definition of order for a linear multi-step method.** The difference operator given in equation (2.43) and the associated linear-multi step method in equation (2.34), are said to be of order p, if in equation (2.44), $C_0 = C_1 = 0 = \ldots = C_p$ and $C_{p+1} \neq 0$. Since the coefficient values $C_q$'s can be written in terms of the $\alpha_j$'s and $\beta_j$'s, it is possible to construct a linear multi-step method of a given order by solving a set of simultaneous equations. Also, the local error at $t_{n+k}$ of the method is defined by the expression L [y(t) ; h], given by equation (2.43), where y(t) is the solution of the I.V.P. (2.33).

2.38 Characteristic polynomials [36] can be formed from linear multi-step methods in order to assess their stability properties. For example, from equation (2.33), the first and second characteristic polynomials, defined as $\rho(\xi)$ and $\sigma(\xi)$ respectively are given by:

$$\rho(\xi) = \sum_{j=0}^{k} \alpha_j \xi^j$$

$$\sigma(\xi) = \sum_{j=0}^{k} \beta_j \xi^j \quad .$$

A linear multi-step method, defined by equation (2.34) is said to consistent if it has order $p \geqslant 1$. As a consequence, for consistency, the method must satisfy the conditions

$$\sum_{j=0}^{k} \alpha_j = 0 \quad \text{and} \quad \sum_{j=0}^{k} j\alpha_j = \sum_{j=0}^{k} \beta_j$$

**2.39 Zero-stability.** Zero-stability ensures that the solutions of the difference equation for $y_n$, which arise because the first order differential equation is being replaced by a higher order difference equation, are damped out in the limit as $h \to 0$. These solutions are frequently called parasitic solutions and the linear multi-step method given by equation (2.34) is said to be zero-stable if no root of the first characteristic polynomial $\rho(\xi)$ has modulus greater than one, and if every root with modulus one is simple.

This leads to Dalquist's fundamental theorem [37], which is:

**Theorem 2.** A linear multi-step method is convergent if and only if it is both consistent and zero-stable.

The proof of this theorem can again be found in Henrici [26] and it is an important result. It is saying that both consistency, which controls the magnitude of the local error arising at each stage of the solution, and zero-stability, which controls the manner in which this error is propogated as the calculation proceeds, are essential if convergence is required. For a one step method, an important result is immediately forthcoming. Since the polynomial $\rho(\xi)$ is of degree one, and a consistent method will lead to a solitary root $\xi = 1$, then consistency implies zero-stability, and hence only consistency is required to ensure convergence.

## Single-step Methods

**2.40** Single-step methods are better able to cope with discontinuities and easier to implement than multi-step methods which can invoke great difficulty in choosing the starting values $y_1, y_2, \cdots y_{k-1}$, particularly when discontinuities are met. It is at these points that multi-step methods require modifications in order to prevent them from failing. This is normally done by using a smoothing function, or an integration restart, both which will be discussed, but the smoothing function which can be used does not represent the true solution, and so divergence from the theoretical solution is possible. As a consequence, single-step methods are useful in these circumstances, since they are self-starting, requiring only information at the last time step. It is possible for a single-step method to finish at the point of discontinuity and resume the integration process at the new level. Multi-step methods, however, cannot easily do this, and neither do they readily permit a change in steplength during the computation, since they rely on information from preceeding integration steps. Finally, single-step methods are generally far easier to implement and code than multi-step methods, since they do not require the storage of back values, either in data or polynomial form. Multi-step methods frequently lead to long and cumbersome computer programs which are very difficult to modify to suit individual problems.

**2.41** Linear multi-step methods achieve high order accuracy by sacrificing the desirable one-step nature of numerical algorithms, but do retain linearity with respect to $y_{n+j}, f_{n+j}$ $j = 0,1,...k$. Single-step methods can attain higher order accuracy by sacrificing linearity, and this is the idea behind the methods first proposed by Runge [38] and subsequently developed by Kutta [39] and Heun [40]. This leads to the class of Runge-Kutta methods that are easy to implement, but require more effort in error analysis than their multi-step counterparts. Explicit and implicit Runge-Kutta methods exist, although explicit methods are the most common form. However, recent development work with implicit Runge-Kutta methods has lead to several breakthroughs, and this work will be discussed in a subsequent chapter.

**2.42** A general single-step method, in explicit form, can be written as:

$$y_{n+1} - y_n = h\Phi(t_n, y_n, h) \qquad (2.45)$$

This method is said to have order p, if p is the largest integer for which

$$y(t+h) - y(t) = h\Phi(t, y(t), h) = O(h^{p+1}) \qquad (2.46)$$

holds, where y(t) is the theoretical solution of the I.V.P. given in equation (2.33).

The method is consistent with the I.V.P. if

$$\Phi(t,y,0) = f(t,y) \qquad (2.47)$$

Euler's method is the only linear multi-step method which falls within the class of equation (2.47), and is obtained by setting

$$\Phi(t,y,h) = f(t,y)$$

It is consistent and has order one.

A theorem exists that gives conditions to ensure the convergence of single-step methods, the proof can again be found in Henrici [26].

**Theorem 3.** i) Let the function $\Phi(t,y,h)$ be continuous jointly as a function of its three arguments, in the region $D_f$ defined by t in $[a,b]$, y in $(-\infty,\infty)$ and h in $[0, h_0]$, $h_0>0$

ii) Let $\Phi(t,y,h)$ satisfy a Lipschitz condition of the form

$$|\Phi(t,y^{*}\cdot h) - \Phi(t,y,h)| \leqslant M_y |y^{*}-y|$$

for all points $(t,y^{*},h)$, $(t,y,h)$ in $D_f$

Then the method given by equation (2.45) is convergent if and only if it is consistent.

**2.43** The definitions and theorems follow through for systems of differential equations, with the necessary amendments from absolute values to vector norms. They also apply to methods of the form:

$$y_{n+1} - y_n = h\Phi(t_{n+1}, y_{n+1}, h) \tag{2.48}$$

which will in general require the employment of an iteration scheme in their solution, since they will form implicit relationships between variables. The requirements for the convergence of an iteration scheme when applied to an implicit linear multi-step method will extend simply to this method since it will be of the form:

$$y_{n+1}^{(s+1)} = h\Phi(t_{n+1}, y_{n+1}^{(s)}, h) + y_n \quad s=0,1,2,... \tag{2.49}$$

**Stability Of Numerical Methods**

2.44 Besides the zero-stability, consistency and convergence of a numerical method, there is another important property that determines its usefulness as an integrator. Zero-stability ensures that the local inaccuracies caused by the method are not propogated in an unwanted manner. However it deals with the case as $h \to 0$ and $n \to \infty$, with nh constant. Consequently, it is necessary to know the way in which errors propogate if h is fixed, or at least does not tend towards zero, and n still tends towards infinity. The global error is defined as the overall error that a numerical method makes in forming a solution sequence from $t_0$ to $t_n$; then, if $e_n$ denotes global error

$$e_n = y_n - y(t_n)$$

where $y_n$ is the solution formed by the scheme and $y(t_n)$ is the theoretical solution to the I.V.P. given in equation (2.33) at the point $t = t_n$. Since the local error determines the error made by the scheme in taking one step, it must be limited in some way to ensure that the global error is controlled also. The new requirement is a stability definition in which the steplength is fixed and the demand is that the error is propogated in a stable manner as $n \to \infty$. this error will include round-off error, which is formed by the computer in the calculated values at each step and can also create difficulties in the generation of accurate solutions if it is not controlled.

**2.45 Absolute stability.** "A numerical method of the type given in equation (2.45) or equation (2.48) is said to be absolutely stable for a given fixed steplength and for a given I.V.P. if the global error $e_n := y_n - y(t_n)$ remains bounded as $n \to \infty$." [41] This definition relies on the choice of the I.V.P. and the problem that is generally used is called the test equation and is given by:

$$y' = \lambda y \quad y(0) = 1 \tag{2.50}$$

where $\lambda$ is a complex constant.

The way in which the absolute stability of a method is monitored is to consider the effects of a single error e.g. in the initial condition, and then to apply the numerical method to the test ordinary differential equation. Then the effect of the initial perturbation can be seen and appropriate restrictions can be made, if possible, to ensure that the global error is bounded.

**2.46 Systems of o.d.e.'s.** The same stability analysis applies to a system of o.d.e.'s in order to ensure control over the growth of the global error. Considering the test system of m equations

$$\underline{y}' = A\underline{y} \quad \underline{y}(0) = \underline{\beta} \tag{2.51}$$

then, assuming that the constant matrix A has m linearly independent eigenvectors [42], it is possible to pre and post multiply A so as to form:

$$P_m^{-1}AP_m = \Lambda$$

where:

$P_m$ is the modal mxm matrix consisting of the eigenvectors of A [43].
$\Lambda$ is the diagonal mxm matrix with the eigenvalues of A on the diagonal

Now defining $\underline{y} = P_m\underline{Z}$ then equation (2.51) becomes:

$$P_m\underline{Z}' = AP_m\underline{Z} \tag{2.52}$$

premultiplying by $P_m^{-1}$ gives:

$$\underline{Z}' = P_m^{-1}AP_m\underline{Z} \tag{2.53}$$

and since $P_m^{-1}AP_m = \Lambda$ this leaves:

$$\underline{Z}' = \Lambda\underline{Z} \tag{2.54}$$

which uncouples into the equations

$$z_i' = \lambda_i z_i \quad z_i(0) = \beta_i \quad i=1,...m$$

The analysis for each of these equations may be carried out in the same way as for a normal scalar equation by applying the method to each equation in turn, after perturbing each initial condition. The reason for having $\lambda$ as complex in the original scalar test equation is now apparent since the eigenvalues of A may be complex. If $\underline{f}(t,\underline{y})$ is differentiable with respect to $\underline{y}$, then the local behaviour of the general I.V.P. is determined, approximately, by the solution of the linearised equation

$$\underline{y}' = \frac{\partial f}{\partial \underline{y}} \underline{y}$$

where $\frac{\partial f}{\partial \underline{y}}$ is the Jacobian matrix of the system.This can be modelled by the linearised equation $\underline{y}' = A\underline{y}$ which is of the same form as the test system. In general the Jacobian for a problem is time-dependent, and in order to carry out stability analysis, the values of the matrix must be 'frozen' at a value of time, so that the constant A matrix can be formed.

**2.47 Absolute stability of Euler's method.** Euler's method applied to a scalar equation is now considered to demonstrate how the stability region of the method is determined. Applying Euler's method then the test o.d.e. given in equation (2.50) will lead to the computed sequence $y_0, y_1, y_2, \ldots$ from the difference equation given by:

$$y_{n+1} = y_n + h\lambda y_n \qquad y_0 = 1 \qquad n = 0,1,2,3,\ldots \tag{2.55}$$

If $y_0$ is perturbed to $y_0 + \epsilon = z_0$, then the sequence will become $z_0, z_1, z_2, \ldots$ from the modified difference equation

$$z_{n+1} = z_n + h\lambda z_n \qquad z_0 = 1+\epsilon \qquad n = 0,1,2,3,\ldots \tag{2.56}$$

Subtracting equation (2.55) from equation (2.56) will give:

$$z_{n+1} - y_{n+1} = (1 + h\lambda)(z_n - y_n) \qquad n = 0,1,2,3,\ldots \tag{2.57}$$

and, recursively applied, this leads to:

$$z_n - y_n = (1+h\lambda)^n(z_0 - y_0) \qquad n = 1,2,3,\ldots \tag{2.58}$$

The requirement is that the effect of the initial perturbation will die away with increasing n, i.e. that $|z_n - y_n| \rightarrow 0$ as $n \rightarrow \infty$. this is only ensured if $|1+h\lambda| < 1$, which leads to the conditions:

i) $h < \dfrac{2}{|\lambda|} \qquad \lambda$ real, $\lambda < 0$

ii) $h < \dfrac{2|\mu|}{\mu^2 + \omega^2} \qquad \lambda$ complex $= \mu + i\omega,\ \mu < 0$

**2.48 Systems of equations.** The analysis for the system of equations given in equation (2.51) is as shown in section 2.46. The system is transformed to an uncoupled set of equations, and Euler's method is applied to each of these in turn. For absolute stability the requirement will become:

$$|1 + h\lambda_i| < 1 \qquad i = 1,2,3,...m$$

which will describe a region in the complex hλ-plane since the eigenvalues of A could be complex. The stability region for Euler's method is shown in Figure 2.7, and it can be seen that for eigenvalues with large real or imaginary parts, the values of h required for stability will be greatly restricted.

**2.49 Inherently stable systems.** In general, the equations arising in Fluid Power simulation correspond to system eigenvalues with negative real parts. Systems which comprise only eigenvalues with negative real parts are termed inherently stable. For these cases the solution is either decreasing or non-increasing, and so it possible to analyse the behaviour of numerical method as time increases, by the way in which perturbations in the initial conditions are reflected throughout the solution computed by the numerical scheme. This solution, for a stable method, should not diverge from the true solution as time increases.

**2.50** However, when the system eigenvalues are positive, this is termed instability of the original system. In this case the solution is either non-decreasing or increasing and consequently any initial error will be likely to grow as time increases. The stability property of a numerical method when it is applied to a non-inherently stable system must ensure that the inaccuracies caused by the numerical method are small in comparison to the errors amplified by the system itself. This concept has been dealt with by Gear [32] in his definition of a stiffly stable method.

**2.51 Implications of stability.** Since the stability of a method is necessary to ensure that the global error is bounded, it is essential to ensure that integration methods employed by HASP have good stability properties. However, stability is a property that is only ensured with most methods if the steplength is limited [44], as has been demonstrated by Euler's method. Mathematical stiffness creates a problem if stability is required since when $\lambda$ is large, the steplength used during the integration process must be very small. Consequently, when solving a highly stiff problem, it is generally more economical to use a method that needs a relatively large steplength in order to be stable, rather than one that places a stringent demand on the choice of time step, even though the first method may require more computational effort to find the solution at each step. In general, such methods are implicit, which in general do require more effort at each step to find the solution than explicit methods. Figure 2.8 shows the stability region for the Backward Euler method which is given by:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \tag{2.59}$$

and although there is no limitation on steplength in order to ensure stability for an inherently stable system, to solve a non-linear system the method requires the use of an iteration scheme. This can involve a great deal of computational effort, particularly when the system being solved is mathematically stiff [45].

**2.52** Consequently, for the solution of inherently stable systems, the Backward Euler method is ideal from a stability point of view. If, however, the consistency and convergence of the method are also considered, it is found that Backward Euler has a local error of the form $C_1h^2$, and although this ensures that the method is both consistent and hence convergent, the method is of low order accuracy and hence will require a small time step in order to ensure an accurate estimation of the true solution. Hence the difficulties that arise when choosing suitable numerical methods for different classes of problems become apparent. If accuracy is required, a method with a high-order error expansion must be used if the steplength is not to be unrealistically small, but at the same time, if the problem to be solved is very stiff, or has eigenvalues with large magnitude, then a method with a low order error expansion must be used, else a prohibitively small time step will be needed to ensure the stability of the method. In general, high order methods have small stability regions, whether explicit or implicit, and so low-order methods must be used in the solution of stiff sets of equations, and a small steplength must be employed [46].

**2.53** An example of methods commonly used are Adam's-type methods which are multi-step, and the way in which these methods are derived can be found in [47]; the first characteristic polynomial of these methods is of the form $\rho(\xi) = \xi^k - \xi^{k-1}$ for a k-step method, which ensures that the methods are zero-stable. Adam's methods that are explicit are called Adams-Bashforth methods, whilst those that are implicit are called Adams-Moulton methods. Adam's methods are implemented in many o.d.e.. solving packages and are particularly suitable for solving large, non-stiff systems [48]. Examples of the stability regions for Adam's methods can be found in Figure 2.9. The diagram illustrates the decreasing stability regions for the methods with increasing accuracy.

**2.54** At present the method used by HASP is Gear's method [49], and this method is incorporated into a package which is both multi-step and multi-order. The stability regions for Gear's method are shown in Figure 2.10 and it can be seen that these regions are larger than the stability regions for Adam's methods for corresponding values of k. Consequently, these methods are advantageous in the solution of stiff systems of o.d.e.'s

2.55 Having discussed integration methods and the properties of stability, convergence and accuracy, a further consideration of mathematical stiffness is now made, this time from a stability viewpoint. Then, further problems that often arise in the simulation of Fluid Power systems are presented since the mathematical theory that must accompany their investigation has been covered.

**Further Consideration of Stiffness**

2.56 Considering the following example

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -2000 & 999.75 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 1000.25 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \end{bmatrix}$$

then the eigenvalues of the system are:

$$\lambda_1 = -2000.5$$
$$\lambda_2 = -0.5$$

and the exact solution is given by:

$$y_1(t) = -1.499875e^{-0.5t} + 0.4999875e^{-2000.5t} + 1$$

$$y_2(t) = -2.99975e^{-0.5t} - 0.00025e^{-2000.5t} + 1$$

The general form of the solution curve is shown in Figure 2.11. The fast transient is negligible after $t \approx 0.002$ and the slow transient at around $t = 10$. Integration in the interval $0 \leqslant t \leqslant 0.002$ requires a very small steplength. However, if an inappropriate numerical method is used to integrate in the range $t > 0.002$ then a small steplength must be used in order to avoid instability. For example, if a fourth-order Runge-Kutta method is used anywhere in the range $t \geqslant 0$, a steplength of less than 0.0014 must be used to avoid instability, resulting in over 7600 steps to reach the steady-state solution. A larger time step can be used if an implicit method is employed to solve this problem, but this will require much more computational effort at each time step to form a solution, and so the classic problems with mathematical stiffness are apparent.

**2.57 Geometric interpretation of stiffness.** Practically, it appears that stiffness occurs in a system when stability rather than accuracy dictates the choice of steplength. Geometrically, looking at the set of solution curves for stiff and non-stiff systems explains why stiffness causes a problem to integration methods. Figure 2.12 shows both a component of the solution curve, and the components of the neighbouring integral curves, in heavy and lighter lines respectively, for typical non-stiff and stiff systems.

**2.58** The broken line shows that in the stiff case there can exist a solution curve which has a component with no fast transient, although the neighbouring components may all have fast transients: stiffness is not related to the geometry of the solution curve, but to the geometry of the family of solution curves. If an unsuitable method is used to solve a stiff problem, then the wrong components are often approximated by the method. Looking at Figure 2.13 shows how an unsuitable method such as Euler's method, can become unstable when used to integrate a stiff system, even when the fast transient is dead. A further interpretation of mathematical stiffness is given by Lambert [50].

**Physical Discontinuities**

**2.59** A discontinuous function is defined by a mathematically instantaneous change in the values of that function as the value of the independent variable increases, and must be taken into consideration, along with mathematical stiffness, when selecting an integration method to solve the differential equations representing the dynamic behaviour of a Fluid Power system. When the equations that represent the behaviour of the system are of a discontinuous nature, many classical integration methods for the solution of the equations are prone to failure at the points of discontinuity.

In Fluid Power systems, discontinuities fall into two main categories:

i) Discontinuities that occur at known times, taking as an example the case of a duty cycle where a variable changes its value to a pre-determined level

ii) Discontinuities that occur because of a variable reaching a particular value, taking as an example the velocity of a linear actuator when it hits an end-stop, causing the velocity to change abruptly.

Since discontinuities present difficulties to numerical methods, it is necessary to design an adequate way with which to deal with this problem, and this will be discussed in chapter 5.

**2.60 Examples showing the occurrence of discontinuities.** The hydraulic actuator circuit in Figure 2.2 can be used to demonstrate how discontinuities may arise. When the actuator hits the end-stop then the velocity trace is discontinuous. Typical responses for velocity and displacement are shown in Figures 2.14 and 2.15. The velocity is an example of a discontinuous time derivative, and another example of this is given by the operation of a relief valve. As the differential pressure across the relief valve rises to the cracking pressure, the flow rate increases from zero to some finite value as shown in Figure 2.16. The discontinuity in the time derivative is shown in Figure 2.17.

**Oscillatory Problems**

**2.61** Another class of I.V.P.'s which arises in Fluid Power simulation consists of problems whose solutions are periodic, or oscillate with a known or unknown frequency. If the frequency were known in advance then a class of methods based on trigonometrical polynomials, developed by Gautshi [51], is particularly appropriate. However, generally in HASP, the frequency of the problems that have oscillatory solutions are not known, and consequently standard numerical methods must be used. Oscillatory problems do not necessarily cause problems to integration methods, and, in general, only do so when the frequency of the oscillations is high, particularly if the amplitude of the solution curve is large as well. In this instance, then numerical methods can require a very small time step in order to pick up the trace of the solution; this is likely to be a problem for all numerical techniques.

**2.62 Example demonstrating oscillatory behaviour.** Referring back to the hydrostatic transmission system in section 2.8, demonstrating how stiffness can arise, then the equationsdescribing the behaviour of the system can lead to an oscillatory solution. If the net load torque is sufficiently small, then the system pressure will not exceed the relief valve cracking pressure and the flow rate through the relief valve will be zero. Consequently, the differential equations for the rates of change of pipe pressure and angular velocity of the rotary load can be written as:

$$\frac{dP}{dt} = \frac{B}{v}(D_p\omega_p - D_m\omega_m) \tag{2.60}$$

$$\frac{d\omega_m}{dt} = \frac{1}{J}(D_mP - f\omega_m - T_l) \tag{2.61}$$

which can be written as the matrix equation:

$$\begin{bmatrix} \frac{dP}{dt} \\ \frac{d\omega_m}{dt} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-BD_m}{v} \\ \frac{D_m}{J} & \frac{-f}{J} \end{bmatrix} \begin{bmatrix} P \\ \omega_m \end{bmatrix} + \begin{bmatrix} \frac{B}{v}(D_p\omega_p) \\ \frac{-T_l}{J} \end{bmatrix} \tag{2.62}$$

The eigenvalues of the system matrix will be the roots of the quadratic equation:

$$\lambda^2 + \frac{f}{J}\lambda + \frac{BD_m^2}{vJ} = 0$$

The solutions will be:

$$\lambda = \frac{-f}{2J} + \frac{1}{2}\left(\frac{f^2}{J^2} - \frac{4BD_m{}^2}{vJ}\right)^{\frac{1}{2}}$$

i.e.

$$\lambda = \frac{-f}{2J} + \left(\frac{f^2}{4J^2} - \frac{BD_m{}^2}{vJ}\right)^{\frac{1}{2}} \tag{2.63}$$

The term:

$$\frac{f^2}{J^2} - \frac{4BD_m{}^2}{Jv}$$

will determine the nature of the eigenvalues and consequently the transient behaviour of the system. If it is positive, the eigenvalues are real and not normally widely separated for practical hydraulic circuits. Oscillations in the pressure and angular velocity will not occur as the system will be overdamped forcing these variables to attain steady state. If, however, the term is negative, which is the case when there is low friction in the system, then the eigenvalues will be complex conjugates and the system will behave in an oscillatory manner. The real part of the eigenvalues will determine the damping coefficient and the complex part will determine the frequency of the oscillations of the solution. When steady state has been reached, the solutions will be:

$$\omega_m = \frac{D_p\omega_p}{D_m}$$

and

$$P = \frac{1}{D_m}(f\omega_m + T_l)$$

**Predictor-Corrector Pairs**

**2.63** When solving stiff systems of differential equations, the time step restriction placed on an explicit method to ensure the stability of the solution is normally so excessive that it prohibits the use of the method. In this case, an implicit method must be used since the time step restrictions are often much more lenient. However in order to use an implicit method, it is necessary to form predicted values, and this is done by using an explicit method. Predictor-Corrector pairs are now discussed and the necessary iteration schemes that must be used with stiff systems are explained.

**2.64** If an implicit k-step method is to be used to solve an I.V.P. in preference to an explicit method, then, referring to section 2.30, at each step the solution of the equation (2.36) must be found, which is:

$$y_{n+k} + \sum_{j=0}^{k-1} \alpha_j y_{n+j} = h\beta_k f(t_{n+k}, y_{n+k}) + h\sum_{j=0}^{k-1} \beta_j f_{n+j}$$

For a non-linear problem, this equation will require an iterative procedure to determine an accurate value for $y_{n+k}$. Although convergence is guaranteed if $h < \frac{1}{L|\beta_k|}$, it is desirable to take as few iterations as possible before a suitably accurate value for $y_{n+k}$ is found, particularly since the evaluation of f at given values of its arguments can be time-consuming. Consequently, the initial estimation should be as close to the true solution as possible. To do this, a separate explicit method is used to estimate $y_{n+k}$, and the value determined by this method is termed $y_{n+k}^{(0)}$. The explicit method is called the predictor, and the implicit method the corrector.

**2.65 Correcting to convergence.** Once a predictor-corrector pair has been established there are two possible routes which can be taken to determine an estimation to $y_{n+k}$. The first of these is to continue the iterative procedure given by equation (2.36) until the iterates have converged, which in practice means until some criteria such as $|y_{n+k}^{(s+1)} - y_{n+k}^{(s)}| < \epsilon$, where $\epsilon$ is a pre-assigned tolerance, is satisfied. The way in which to determine $\epsilon$ is difficult to generalise, but the solution obtained, $y_{n+k}^{(s+1)}$, is an acceptable approximation to the exact solution $y(t_{n+k})$ of the original scheme. This mode of operation, where each iteration corresponds to one application of the corrector, is termed correcting to convergence. In this mode, the number of iterations needed cannot be calculated in advance; or, alternatively, the number of function evaluations that will be required at each step can not be foretold. However, since the accepted value $y_{n+k}^{(s+1)}$ will be independent of the value $y_{n+k}^{(0)}$ estimated by the predictor, then the local error and stability characteristics of the combined method are those of the corrector alone; the properties of the predictor not being important [52]. Hence, h must be chosen so that $\lambda h$ lies within the stability interval of the corrector, it is of no concern if the value of $\lambda h$ does not lie within the stability interval of the predictor.

**2.66 Limited application of the corrector.** The alternative approach is to stipulate in advance the number of times, m, that the corrector is applied at each time step. This approach is motivated by the desire to restrict the number of function evaluations per step. However, the local error and stability characteristics ae no longer those of the corrector alone, but are dependent on both of the methods used. Hull and Cremer [53] have introduced a standard notation which describes the mode in which a predictor-corrector pair is applied, and also tells immediately how many function evaluations per step are required.

2.67 Letting P denote an application of the predictor, C a single application of the corrector, and E an evaluation of f in terms of known values of its arguments; then if $y_{n+k}^{(0)}$ is computed by the predictor, the evaluation of $f_{n+k}^{(0)} \equiv f(t_{n+k}, y_{n+k}^{(0)})$ is made, and the corrector is applied once to obtain $y_{n+k}^{(1)}$, then the calculation made so far can be represented by the expression PEC. A further evaluation of $f_{n+k}^{(1)} \equiv f(t_{n+k}, y_{n+k}^{(1)})$ followed by a second application of the corrector yields $y_{n+k}^{(2)}$, and the overall calculation is now denoted by PECEC, or $P(EC)^2$. Applying the corrector m times is similarly denoted by $P(EC)^m$. Because m is fixed beforehand, the value $y_{n+k}^{(m)}$ is accepted as the numerical solution at $t_{n+k}$.

**2.68** It is advantageous if the predictor and corrector are separately of the same order, and the following result is indicative of the reasons for this [54]. If the predictor-corrector method for which the predictor has order $p^*$ and the corrector has order p, is applied in $P(EC)^m$ mode, where p , $p^*$ ,m are integers and $p^* \geqslant 0$ , $p \geqslant 1$ , $m \geqslant 1$, then, if $p^* \geqslant p$, the principal local error of the algorithm is that of the corrector alone. If $p^* = p-q$, $0 < q \leqslant p$, then the principal local error term of the algorithm is:

i) That of the corrector alone, when $m \geqslant q + 1$
ii) Of the same order as that of the corrector, but not identical with it, when $m = q$.

In the mode of correcting to convergence, the principal local error term of the predictor-corrector pair is that of the corrector alone, no matter what the order of the predictor.

**2.69 Example to illustrate predictor-corrector pair in PECE mode.** Using two Adam's second order methods will illustrate the technique of the PECE mode and demonstrate that the local error of the combined pair is equivalent to the local error of the corrector alone. The methods are:

$$y_{n+1} = y_n + \frac{h}{2}[3f_n - f_{n-1}] \quad :\text{—Local T.E. is } \frac{5}{12}h^3 y_n''' + O(h^4)$$

$$y_{n+1} = y_n + \frac{h}{2}[f_{n+1} + f_n] \quad :\text{—Local Error is } -\frac{h^3}{12}y_n''' + O(h^4)$$

If the explicit method is applied to the I.V.P. first then this will yield:

$$y_{n+1}^{(0)} = y_n + \frac{h}{2}[3f_n - f_{n-1}] \tag{2.64}$$

and the true solution would satisfy

$$y(t_{n+1}) = y(t_n) + \frac{h}{2}[3f(t_n) - f(t_{n-1})] + \frac{5}{12}h^3y'''(t_n) + O(h^4) \tag{2.65}$$

Subtracting equation (2.64) from (2.65) and applying the assumption of exact back values will give:

$$y(t_{n+1}) - y_{n+1}^{(0)} = \frac{5}{12}h^3y'''(t_n) + O(h^4) \tag{2.66}$$

Applying the implicit method will give:

$$y_{n+1} = y_n + \frac{h}{2}[f_{n+1}^{(0)} + f_n] \tag{2.67}$$

and the true solution will satisfy:

$$y(t_{n+1}) = y(t_n) + \frac{h}{2}[f(t_{n+1}) + f(t_n)] - \frac{h^3}{12}y'''(t_n) + O(h^4) \tag{2.68}$$

Subtracting equation (2.67) from (2.68) and applying the Mean Value Theorem, and the assumption of exact back values, will lead to:

$$y(t_{n+1}) - y_{n+1} = \frac{h}{2}\frac{\partial f}{\partial y}\Big|_{(y(t_{n+1}),\eta_{n+1})}(y(t_{n+1}) - y_{n+1}^{(0)}) - \frac{h^3}{12}y'''(t_n) + O(h^4) \tag{2.69}$$

where

$\eta_{n+1}$ is in $(y(t_{n+1}), y_{n+1}^{(0)})$

With the substitution of equation (2.66), equation (2.69) reduces to:

$$y(t_{n+1}) - y_{n+1} = -\frac{h^3}{12}y'''(t_n) + O(h^4) \tag{2.70}$$

Hence the local error of the combined pair is that of the corrector alone.

**Suitable Iteration Schemes For Stiff Systems**

**2.70** If the problem

$\underline{y}' = \underline{f}(t,\underline{y}) \quad y_i(0) = \alpha_i \quad i = 1,2,...m$

is considered, then applying the predictor-corrector pair of forward Euler and backward Euler will give:

$$\underline{y}_{n+1}^{(0)} = \underline{y} + h\underline{f}(t_n,\underline{y}_n)$$
$$\underline{y}_{n+1}^{(1)} = \underline{y} + h\underline{f}(t_{n+1},\underline{y}_{n+1}^{(0)})$$

If the predictor-corrector pair is applied in a correcting to convergence mode, then the natural iterative technique would appear to be a fixed point method of the form:

$$\underline{y}_{n+1}^{(s+1)} = \underline{y} + h\underline{f}(t_{n+1},\underline{y}_{n+1}^{(s)}) \quad s = 1,2,3,... \qquad (2.71)$$

By formulating the Jacobian of the original system, the Lipschitz constant can be found, this being the spectral radius [55] of the Jacobian matrix, that is, the eigenvalue of largest modulus, where the elements of the Jacobian are given by:

$$J_{ac_{ij}} = \frac{\partial f_i}{\partial y_j} \quad i = 1,2,...m \quad j = 1,2,...m$$

A contraction mapping on the reals is present if hL < 1, which will ensure the convergence of the iteration scheme. However, for a stiff system, L will be large and hence the "convergence of iterations " restrictions is of the kind that must be avoided; since using an implicit method allows a larger time step for stability, but the iteration scheme demands a small time step h in order to guarantee convergence.

2.71 However, Newton-Raphson's method [56] for the iterative solution of equation (2.71) does not rely on the value of h. The method is :

$$\underline{y}_{n+1}^{(s+1)} = \underline{y}_{n+1}^{(s)} - [I - hJ_{ac}(\underline{y}_{n+1}^{(s)})]^{-1}[\underline{y}_{n+1}^{(s)} - \underline{y}_n - h\underline{f}(t_{n+1},\underline{y}_{n+1}^{(s)})] \qquad (2.72)$$

where

$$J_{ac}(\underline{y}_{n+1}^{(s)}) = \frac{\partial \underline{f}}{\partial \underline{y}} \Big|_{\underline{y}_{n+1}^{(s)}}$$

2.72 A graphical representation of the way in which Newton-Raphson's method finds the roots of the function $F(x) = 0$ is shown in Figure 2.18. The method works by advancing down the tangent of $F(x)$, for a particular value of x, until it crosses the x-axis. It then advances down the tangent of $F(x)$ corresponding to this crossover value of x, and proceeds recursively until a root is located. Newton-Raphson's method does not demand a restriction on h to guarantee that the iterates converge. However, it does require that the original estimation made by the predictor be accurate, else the method can 'wander' and fix onto a false root, or alternatively not find a root at all. Precise requirements to ensure convergence are given by Churchhouse [57].

**2.73** Besides the stipulation of an accurate starting value, Newton-Raphson's method also requires the evaluation of a Jacobian at each time step that it is applied, and the consequent solution of a set of simultaneous equations. If a full Jacobian is evaluated, then equation (2.72) can be rewritten in the following form so as to allow LU decompositon rather than matrix inversion to be carried out when solving the equations.

$$[I - hJ_{ac}(\underline{y}_{n+1}^{(s)})][\underline{y}_{n+1}^{(s+1)} - \underline{y}_{n+1}^{(s)}] = -[\underline{y}_{n+1}^{(s)} - \underline{y}_{n} - h\underline{f}(t_{n+1}, \underline{y}_{n+1}^{(s)})] \qquad (2.73)$$

**2.74** There are alternative forms of Newton's method, and the structure of these is different from Newton-Raphson because a full Jacobian is not necessarily evaluated. An example is the Newton-Jacobi method [58], in which only the diagonal elements of the Jacobian matrix are considered in the equation

$$[I - hJ_{ac}(\underline{y}_{n+1}^{(s)})] \qquad (2.74)$$

The result of this is that the effort required to solve the sets of simultaneous equations is substantially reduced. Once the diagonal elements of the Jacobian have been evaluated, then the inverse of the matrix given in equation (2.74) is simple to find. The inverse is in turn a diagonal matrix, whose elements are the reciprocals of the elements of the diagonal matrix, which equation (2.74) will lead to. For example the inverse of the (5x5) diagonal matrix given by:

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 & 0 \\ 0 & 0 & a_{33} & 0 & 0 \\ 0 & 0 & 0 & a_{44} & 0 \\ 0 & 0 & 0 & 0 & a_{55} \end{bmatrix}$$

is easily evaluated.

**2.75** The rate of convergence of the iteration scheme for this different method is dependent on the nature of the original full Jacobian. If the diagonal elements are large in modulus compared with the sum of the off-diagonal elements in the same row, then convergence is fast and the method is efficient. It is when the off-diagonal elements 'swamp' the diagonal elements that care must be taken, since the rate of conergence may be considerably reduced. However, since there is now no need for LU-decomposition or matrix inversion, the computational work required to find a solution at each time step has been considerably reduced.

**2.76** Some numerical methods do not require the re-evaluation of the Jacobian matrix at every iteration, or even at every time step. Methods such as Gear's keep the same Jacobian matrix for as long as the iteration scheme is converging rapidly. Not evaluating the Jacobian often makes little difference to the accuracy of the results obtained, but makes a large difference to the computational time spent by the method whilst solving a problem.

## Conclusions

2.77 The issues raised in this chapter are applied when developing and testing the new integration method that is introduced in chapter 3, and further investigated in the subsequent chapters. In particular, the considerations of stability, covergence and accuracy are particularly relevant when solving the stiff systems of ordinary differential equations that are studied to test the new method. The test problems have been chosen to ensure that the new method to be developed can cope with the typical mathematical difficulties that are met inside the HASP package, such as discontinuous and oscillatory problems, as well as highly stiff systems.

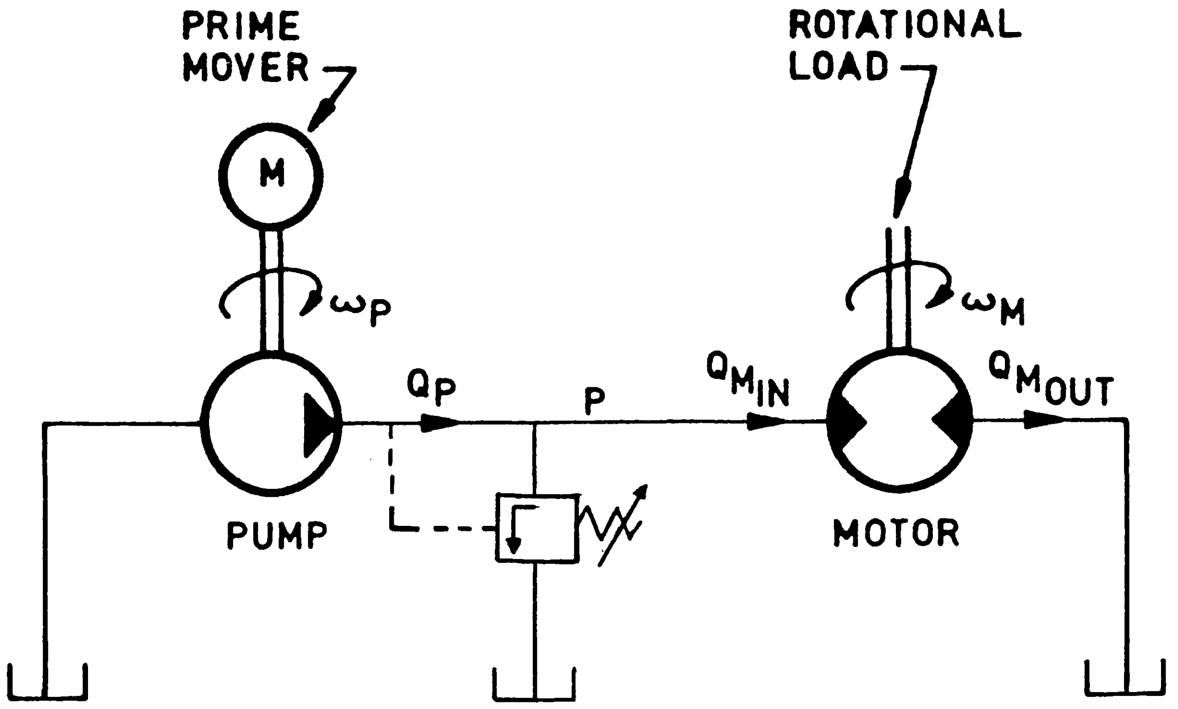


FIGURE 2.1 SIMPLE HYDROSTATIC TRANSMISSION SYSTEM

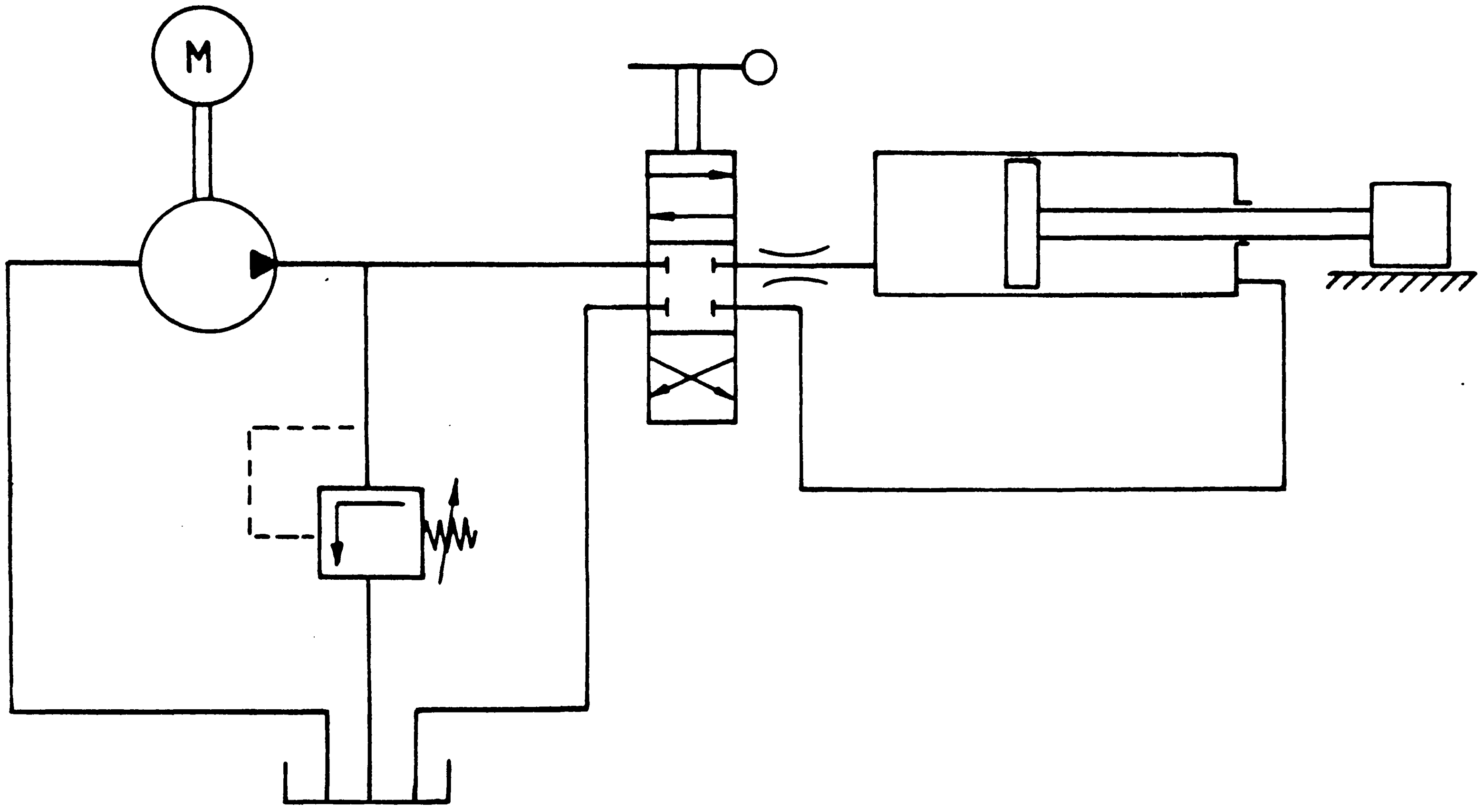


FIGURE 2.2 LINEAR ACTUATOR CIRCUIT WITH METER-IN ORIFICE

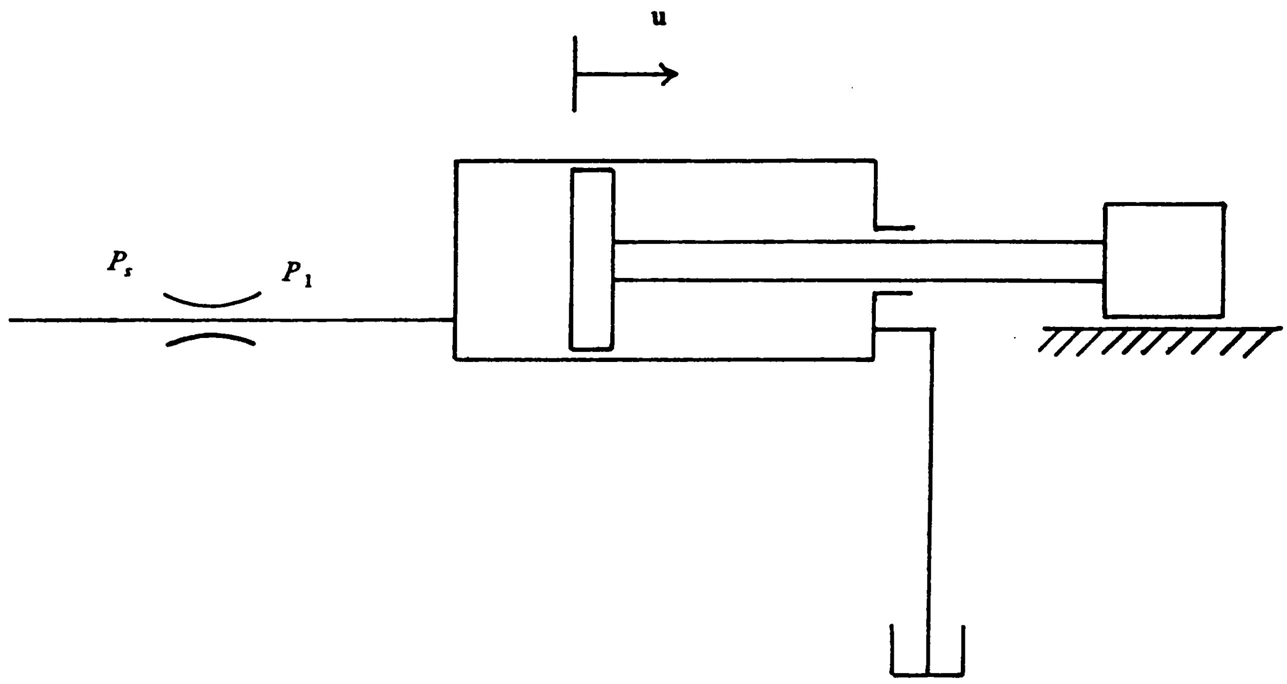


FIGURE 2.3 SIMPLIFIED LINEAR ACTUATOR CIRCUIT

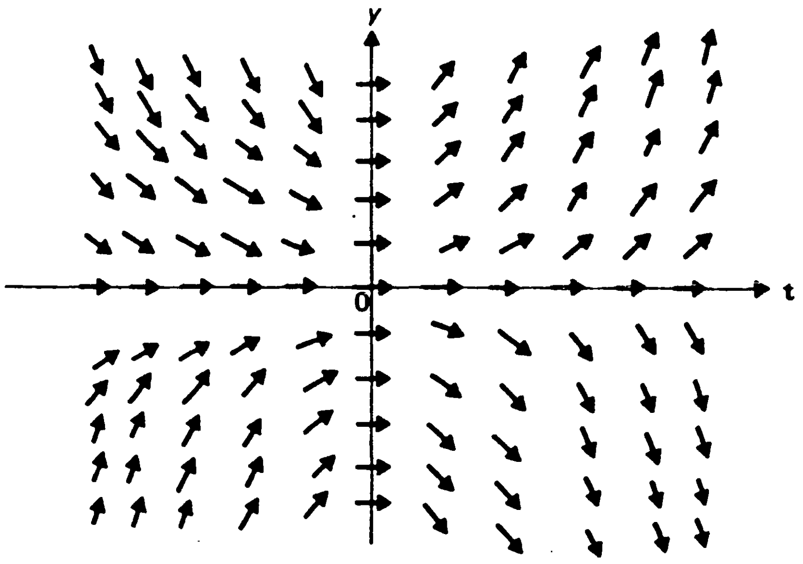


FIGURE 2.4 DIRECTION FIELD FOR y' = 2ty WITH I.C. y(0) = 1

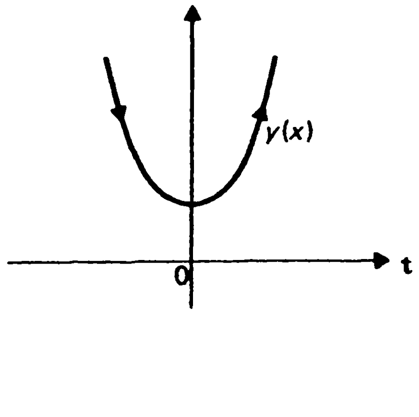


FIGURE 2.5 SOLUTION OF y' = 2ty WITH I.C. y(0) = 1

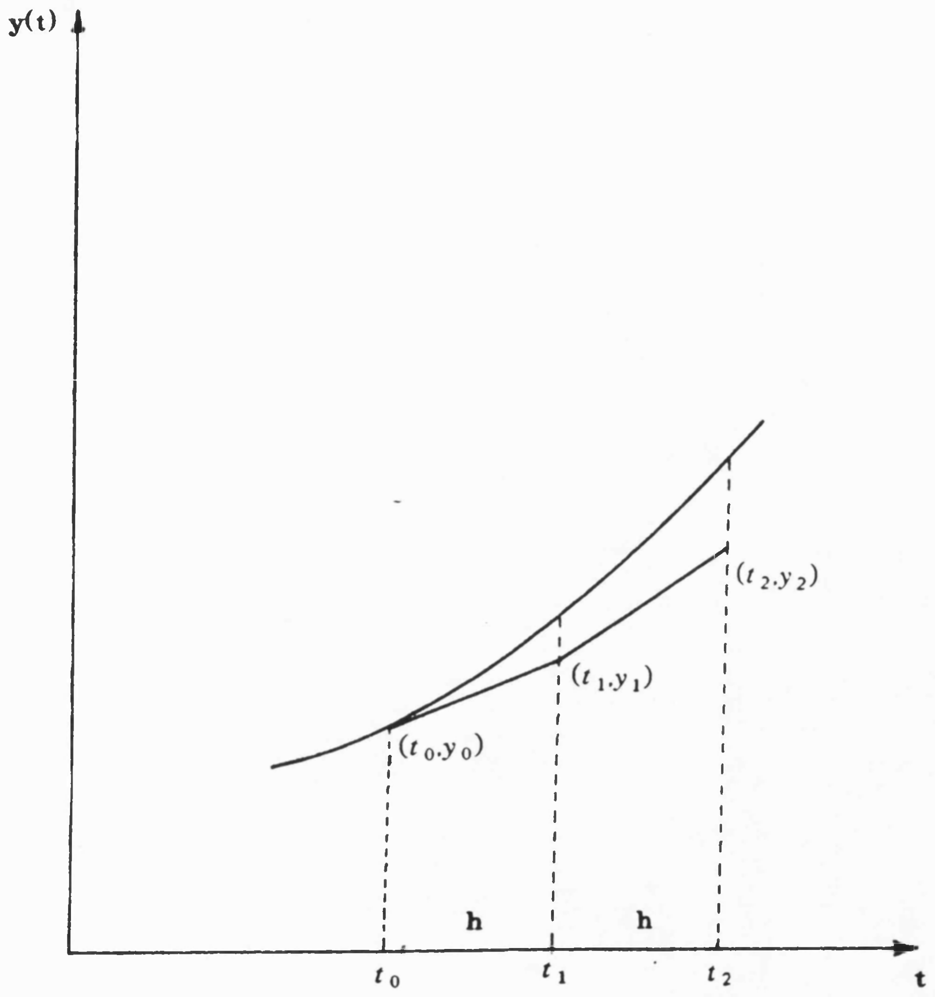


FIGURE 2.6 GEOMETRIC INTERPRETATION OF EULER'S METHOD

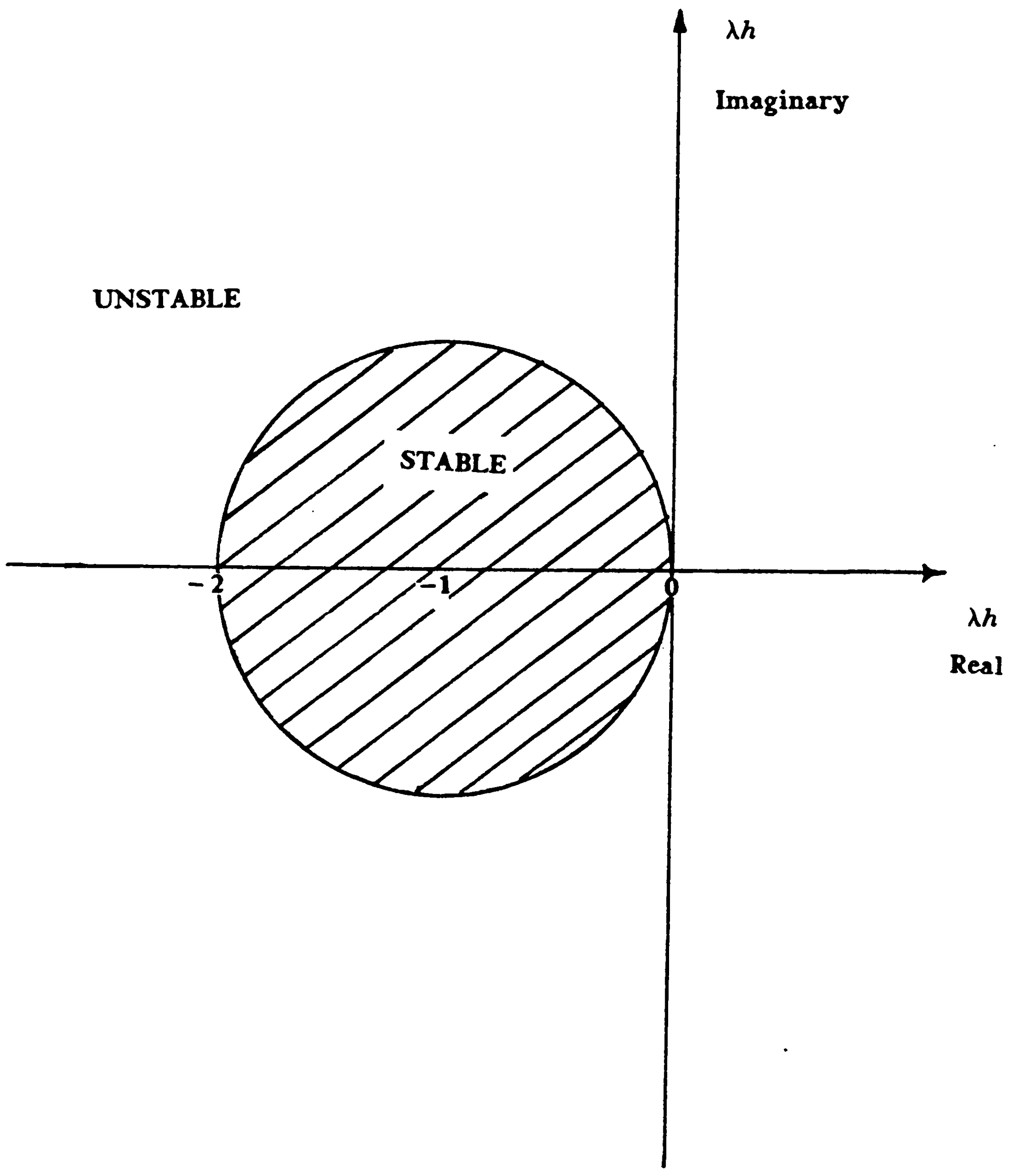


FIGURE 2.7 STABILITY REGION FOR EULER'S METHOD

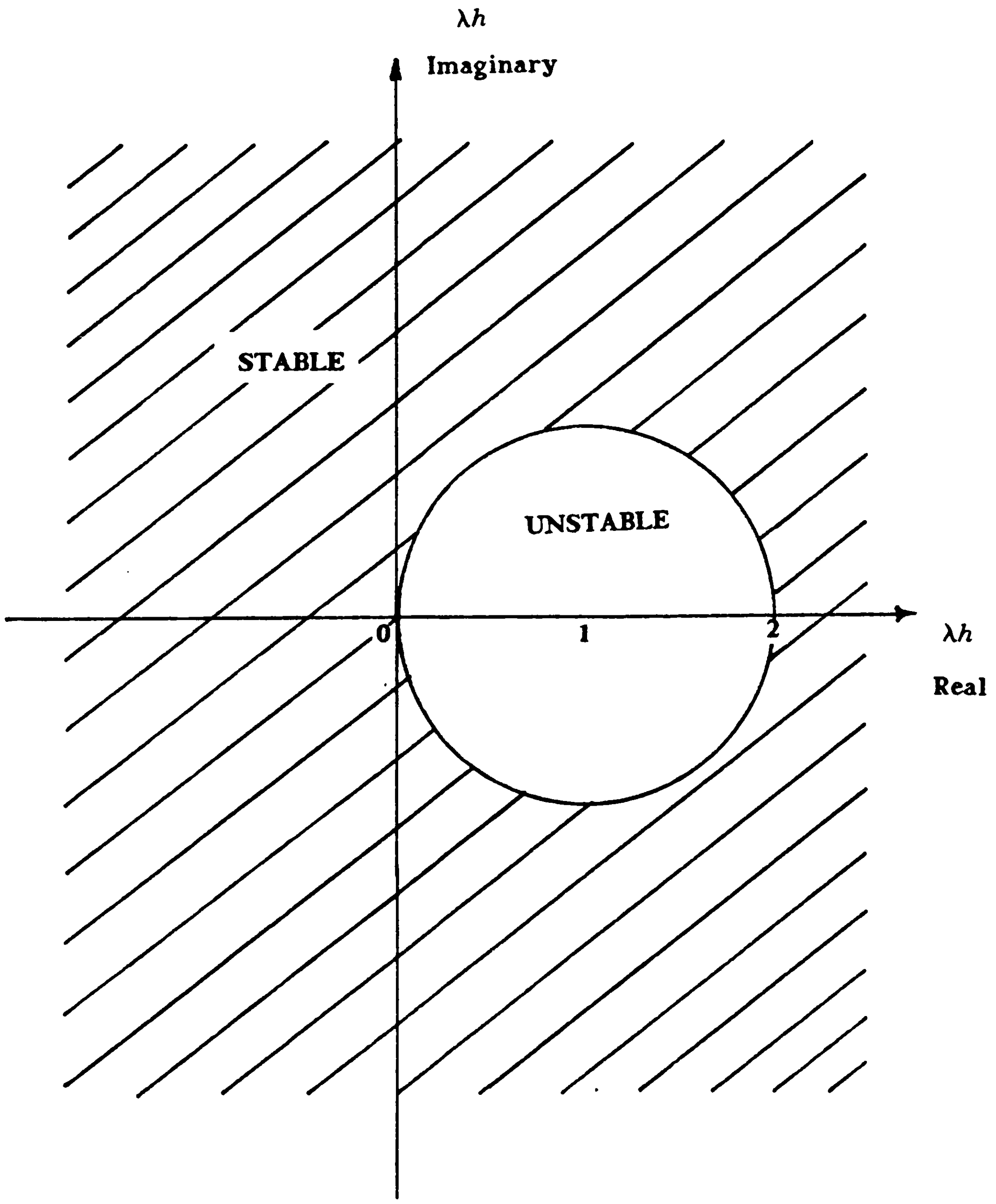
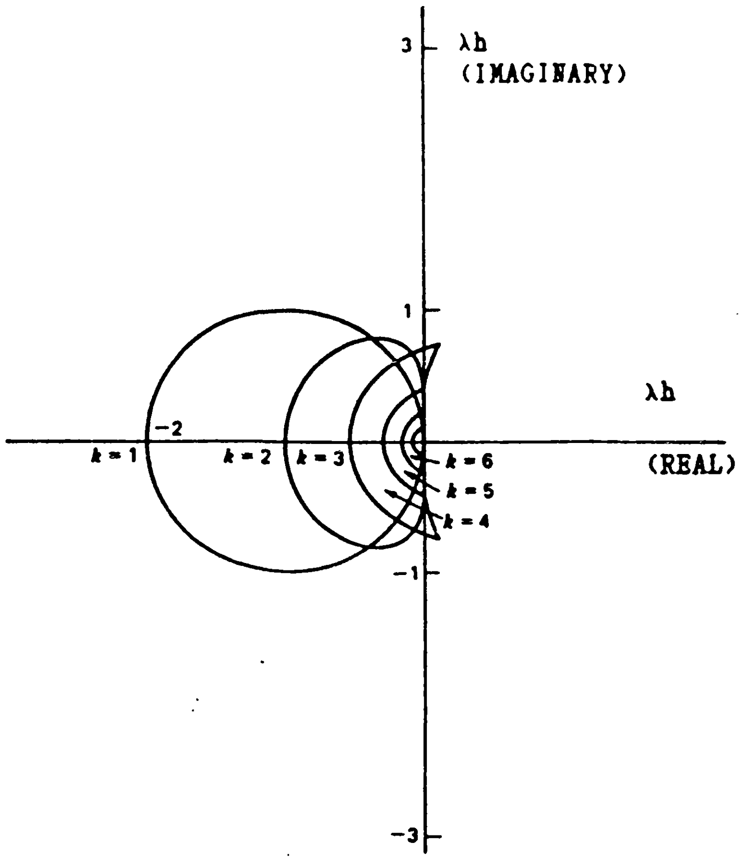
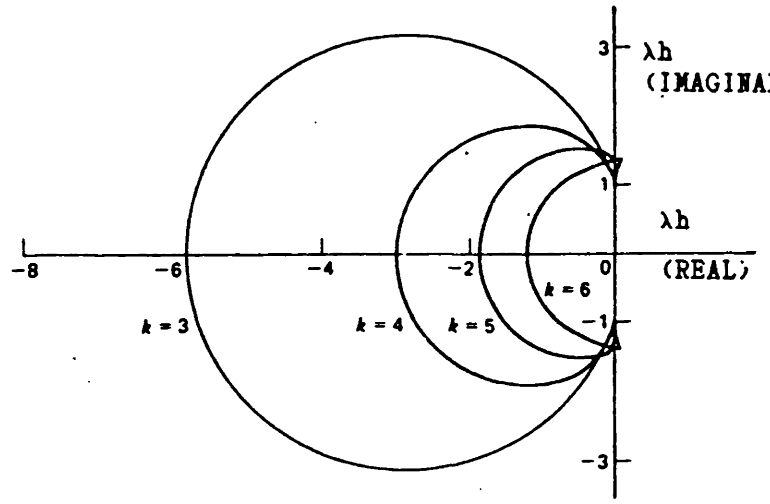


FIGURE 2.8 STABILITY REGION FOR THE BACKWARD EULER METHOD



Stability regions for Adams-Bashforth methods. The method of order $k$ is stable inside the region indicated.



Stability regions for Adams-Moulton methods. The method of order $k$ is stable inside region indicated.

FIGURE 2.9 STABILITY REGIONS FOR ADAM'S METHODS

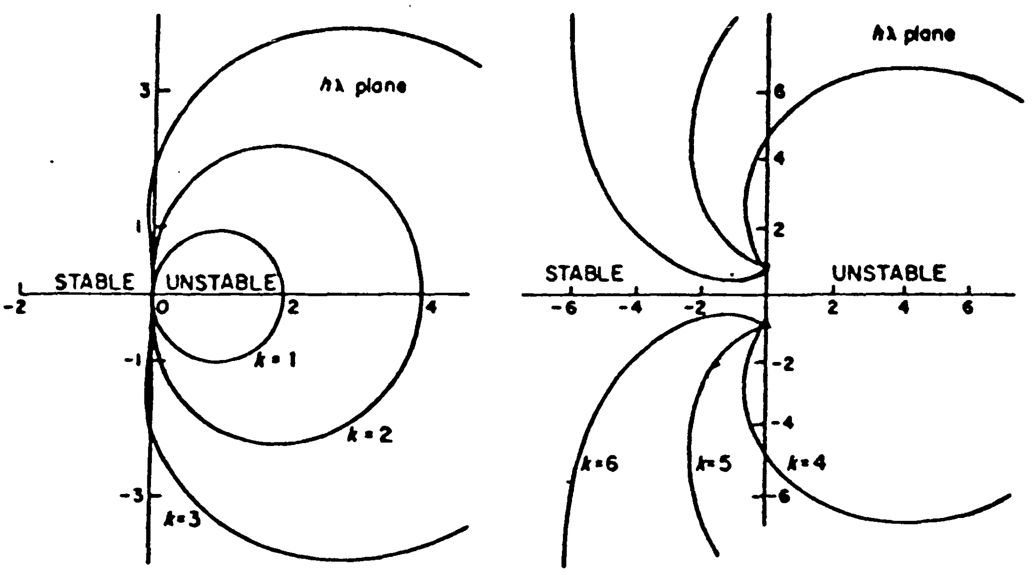


FIGURE 2.10 STABILITY REGIONS FOR THE GEAR'S METHODS

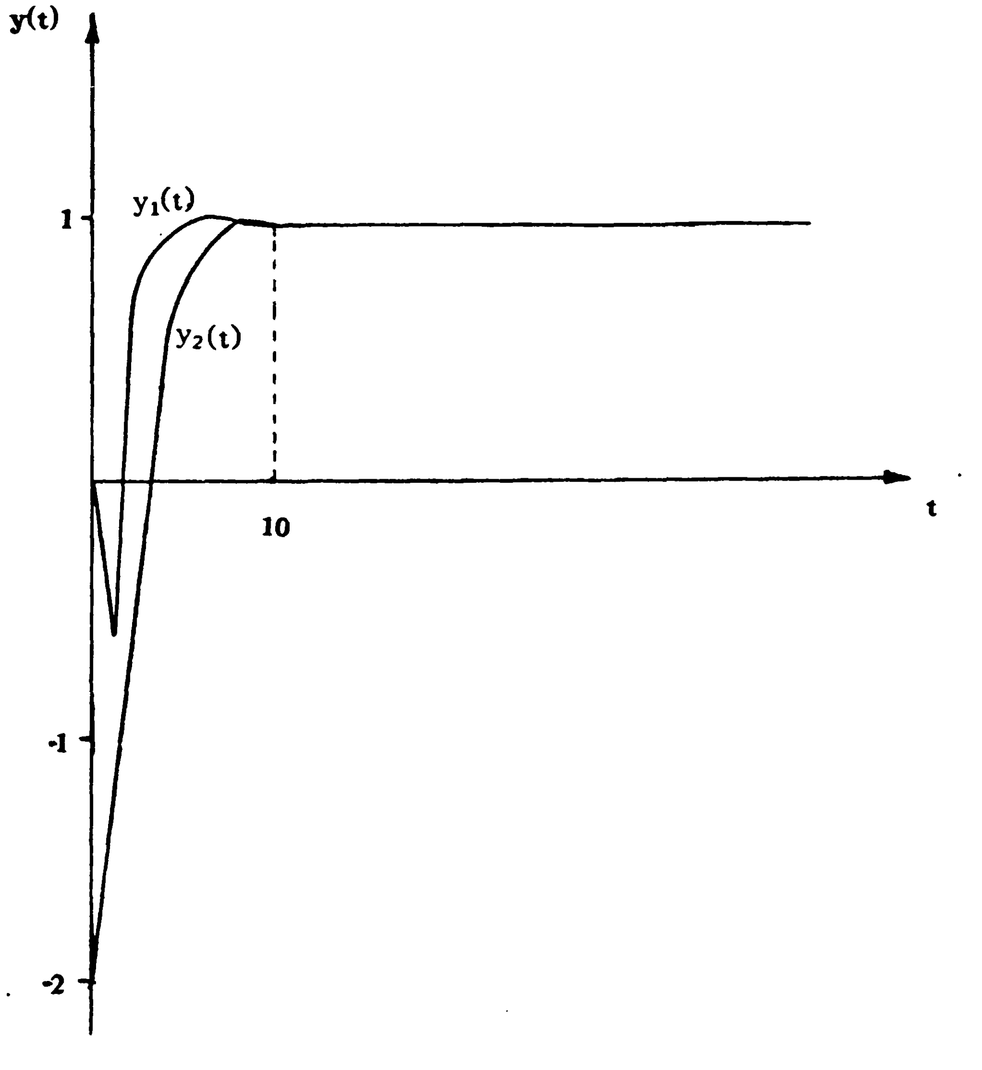
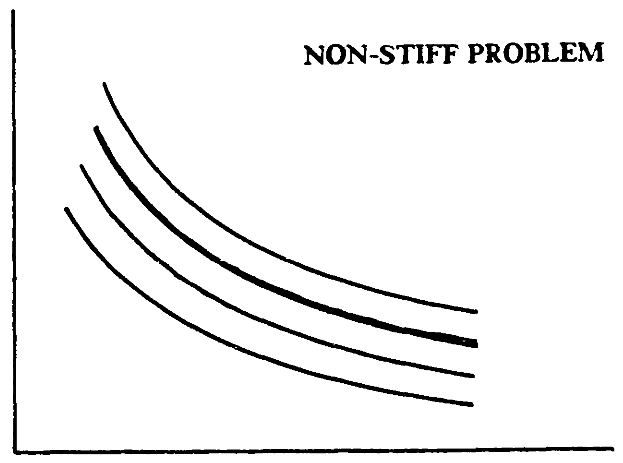


FIGURE 2.11 GENERAL FORM OF SOLUTION CURVES FOR STIFF PROBLEM



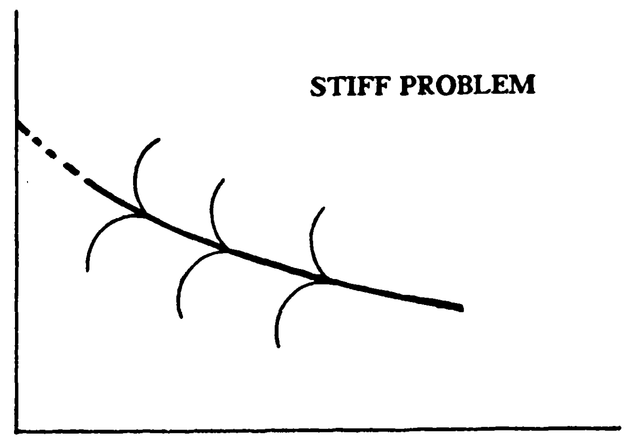


FIGURE 2.12 COMPONENTS OF SOLUTION CURVES FOR TYPICAL NON-STIFF AND STIFF PROBLEMS

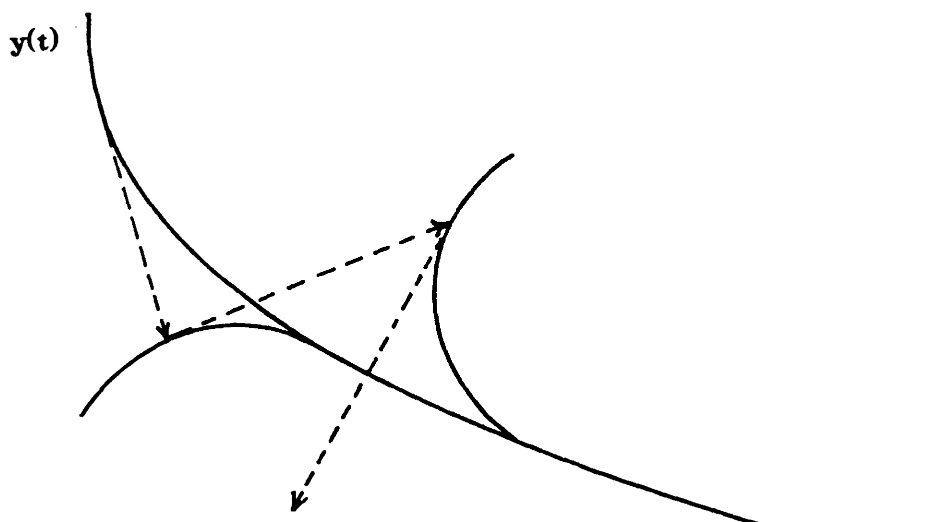


FIGURE 2.13 DEMONSTRATING INSTABILITY WHEN USING AN UNSUITABLE METHOD TO SOLVE A STIFF PROBLEM

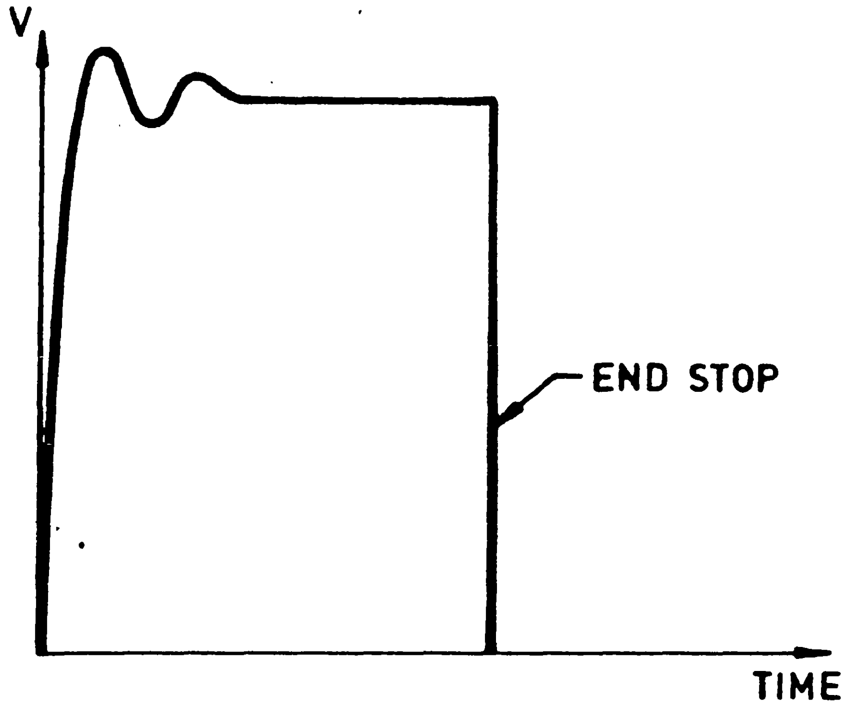


FIGURE 2.14 ACTUATOR VELOCITY WHEN AN END STOP DISCONTINUITY IS ENCOUNTERED

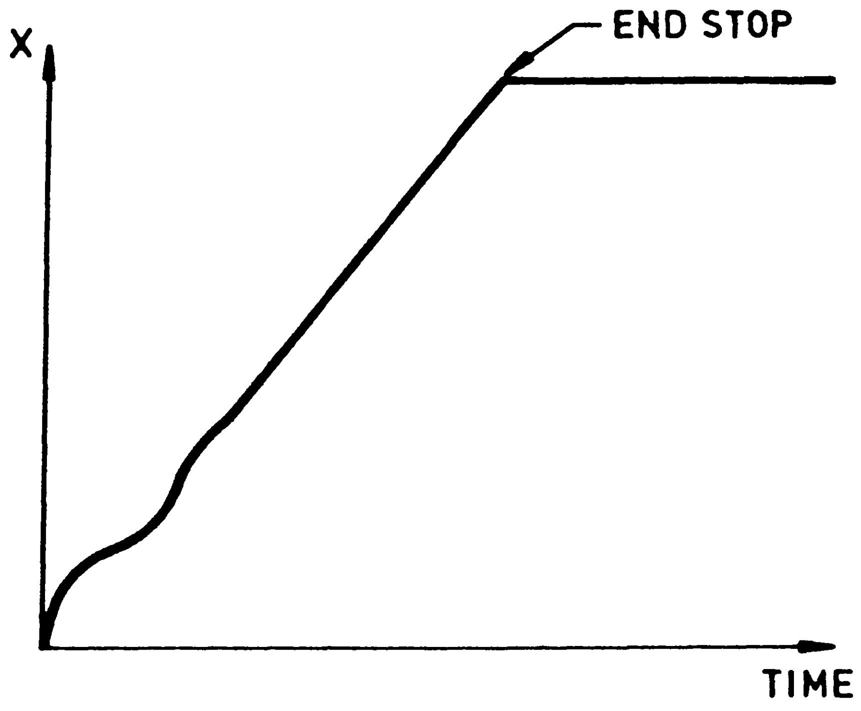


FIGURE 2.15 ACTUATOR DISPLACEMENT WHEN AN END STOP DISCONTINUITY IS ENCOUNTERED

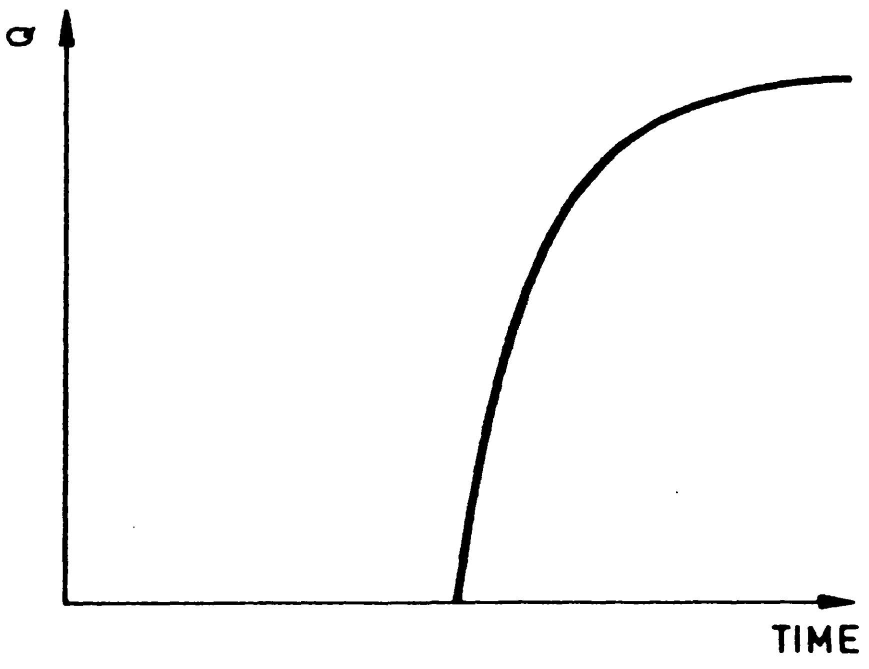


FIGURE 2.16 DISCONTINUITY IN FLOW RATE AS THE DIFFERENTIAL PRESSURE ACROSS A RELIEF VALVE EXCEEDS THE CRACKING PRESSURE

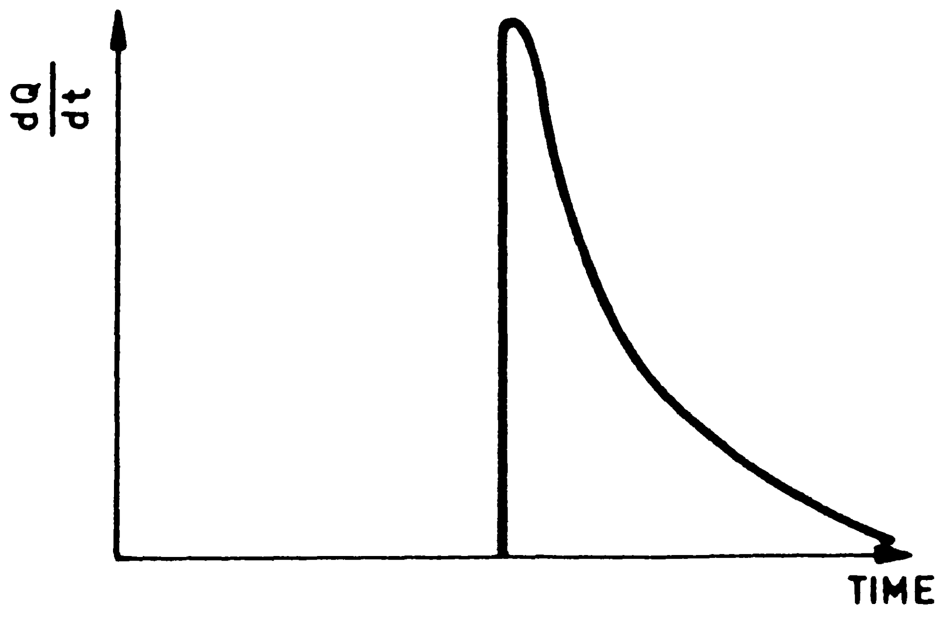


FIGURE 2.17 TIME DERIVATIVE OF FLOW RATE THROUGH A RELIEF VALVE AS THE DIFFERENTIAL PRESSURE EXCEEDS THE CRACKING PRESSURE

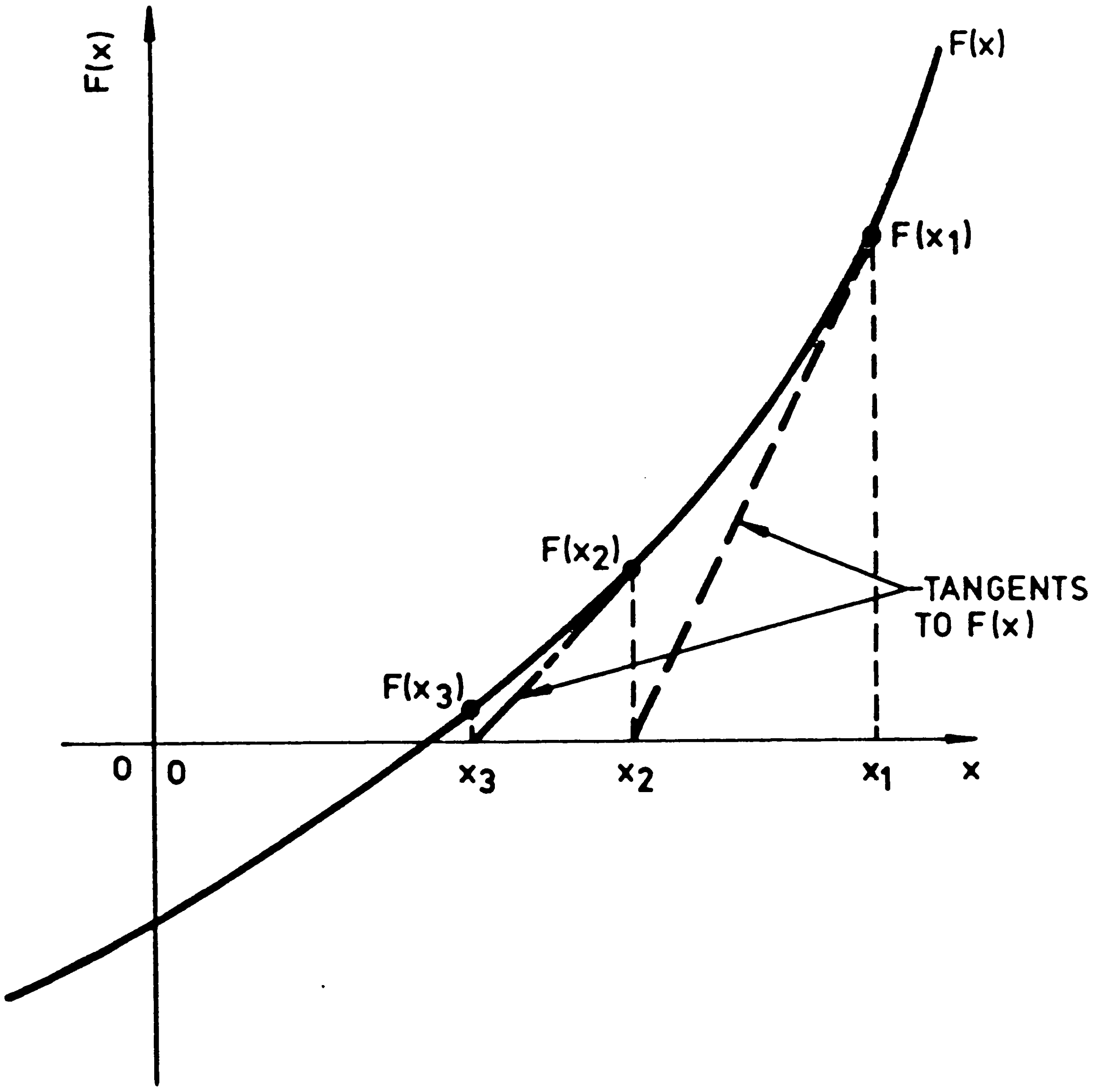


FIGURE 2.18 NEWTON'S METHOD TO SOLVE F(X) = 0

# CHAPTER 3

**DETAILED CONTENTS** **Page**

# CHAPTER 3

# INITIAL DEVELOPMENT AND TESTING OF A NEW INTEGRATION METHOD

## Introduction

3.1 This chapter introduces a new method that will be investigated. The method is developed and subsequently tested on several systems of differential equations, each of which arises from a practical engineering problem.

First, three methods are formulated, one of which is explicit and the other two implicit, and considerations that must be dealt with when programming these methods are discussed. This includes the software that is written, C.P.U. timing and the convergence of iterations when using the implicit method. Then the method is applied to a range of test problems which include: a linear actuator circuit; a second order oscillatory system; a non-linear hydraulic actuator circuit and a discontinuous and mathematically stiff hydraulic actuator system. These problems have been chosen since they lead to systems of equations which demonstrate the mathematical difficulties discussed in chapter 2. The performance of the method is compared with other numerical methods such as Gear's method and the Backward Euler method, which have been used to solve the same problems, and conclusions are drawn concerning the merits of the new integration method.

## A New Integration Method

3.2 The new method being investigated is a numerical scheme based on the formula for the true solution of a first order linear ordinary differential equation. The method was suggested by Professor D.E. Bowns of the Engineering Department at the

University of Bath and was originally studied by P.S. Leung [2], although the methods studied by Leung were formulated using a less general approach. A similar approach to that taken by Leung has also been adopted by Keener and Meyer [59], although their work was not extensive. The mathematical analysis of the method yields desirable results from a stability viewpoint for certain types of problems, which are discussed in chapter 4, and as a consequence the method has been studied in more detail, since the benefits it enjoys may be extended to ensure its worthwhile application to certain problem areas.

## Explicit Formulation Of The New Integration Method

**3.3** For the linear first order differential equation

$$y' = a(t)y + b(t) \qquad y(t_0) = \alpha \tag{3.1}$$

then the true solution [60] is given by the following equation:

$$y(t) = e^{\int_{t_0}^{t} a(z)dz}\left[y(t_0) + \int_{t_0}^{t} b(s)\, e^{-\int_{t_0}^{s} a(x)dx}\, ds\right] \tag{3.2}$$

An integration method can be formulated by considering, at first, a(t) and b(t) as constant functions. Provided that neither function is discontinuous and that the interval $[t_0, t]$ is small, this is not an unreasonable assumption. Consequently, the integrals in equation (3.2) can be now be evaluated and y(t) can be written as:

$$y(t) = e^{(t-t_0)a}(y(t_0) + b\int_{t_0}^{t} e^{(t_0-s)a}ds) \tag{3.3}$$

which becomes:

$$y(t) = e^{(t-t_0)a}(y(t_0) - b\left[\frac{e^{(t_0-s)a}}{a}\right]_{t_0}^{t}) \tag{3.4}$$

Simplifying equation (3.4) and replacing $(t - t_0)$ by h will leave:

$$y(t) = e^{ah}y(t_0) - \frac{b}{a}(1 - e^{ah})$$

which simplifies to:

$$y(t) = \frac{b}{a}(e^{ah} - 1) + y(t_0)e^{ah} \tag{3.5}$$

Consequently, this can be generalised into a numerical scheme that will give an estimation to $y(t_{n+1})$, having been given an initial condition, $y(t_n)$, and the values of the functions a(t) and b(t) at time $t = t_n$. This scheme, which is explicit, will be given by:

$$y(t_{n+1}) = \frac{b(t_n)}{a(t_n)}(e^{a(t_n)h} - 1) + y(t_n)e^{a(t_n)h} \tag{3.6}$$

and this is the explicit formulation of the new integration method. From this approach, several methods can be developed, and it is these methods that are studied in the work presented in this thesis.

## Implicit Formulation Of the New Integration Method

**3.4** The scheme in equation (3.6) will give the exact solution for y(t) if a(t) and b(t) are both constant functions. Alternatively, since a(t) and b(t) will not generally be constant, a more accurate representation of the solution may be made if these functions are considered as variable and the values of a(t) and b(t) are taken at $t = t_{n+1}$, rather than at $t = t_n$, and replaced in the scheme to give:

$$y(t_{n+1}) = \frac{b(t_{n+1})}{a(t_{n+1})}(e^{a(t_{n+1})h} - 1) + y(t_n)e^{a(t_{n+1})h} \tag{3.7}$$

This is not an implicit scheme for a linear ordinary differential equation. However, if the equation to be solved is non-linear, then an iteration scheme will be needed for the solution of the method at each time step. As an example, for the O.D.E.

$$y' = y^2 + 3t \qquad y(0) = 1$$

then a(t) = y(t) and b(t) = 3t, and, since y(t) is not known at $t = t_{n+1}$, an iteration scheme will be required if the scheme given in equation (3.7) is to be applied.

## An Alternative Implicit Formulation Of The New Integration Method

**3.5** Since the method has been formed by approximating the integrals in the formula for the true solution of a first order differential equation, other methods are apparent if these integrals are approximated differently. An example is when the forcing function b(t) is assumed to vary linearly with time, with a(t) still being considered as a constant function. Equation (3.3) will then become:

$$y(t) = e^{(t-t_0)a}[y(t_0) + \int_{t_0}^{t} b(s)e^{(t_0-s)a}ds] \tag{3.8}$$

and the integral

$$\int_{t_0}^{t} b(s)e^{(t_0-s)a}ds \tag{3.9}$$

can be evaluated using integration by parts, since

$$\int_{t_0}^{t} b(s)ds = (b(t) - b(t_0))\frac{(t-t_0)}{2}$$

and also

$$b'(s) = \frac{b(t) - b(t_0)}{h}$$

Equation (3.9) then becomes

$$e^{t_0 a}\int_{t_0}^{t} b(s)e^{-sa}ds \equiv -e^{-ah}\frac{b(t)}{a} + \frac{b(t_0)}{a} - \frac{\Delta b}{ha^2}e^{-ah} + \frac{\Delta b}{ha^2} \tag{3.10}$$

where $h = t - t_0$ and $\Delta b = b(t) - b(t_0)$

and so the numerical scheme will be, on substituting equation (3.10) into equation (3.8), and simplifying the resultant expression:

$$y(t_{n+1}) = \frac{b(t_n)}{a(t_n)}(e^{a(t_n)h} - 1) + \frac{\Delta b_n}{a(t_n)}(\frac{e^{a(t_n)h}-1}{a(t_n)h} - 1) + y(t_n)e^{a_n h} \tag{3.11}$$

where now, $\Delta b_n = b(t_{n+1}) - b(t_n)$

Alternatively, $a(t_n)$ could be replaced with $a(t_{n+1})$ in this scheme, since $a(t)$ is being considered as a constant function.

Many numerical schemes can be constructed using this approach, but those already formulated provide an adequate starting set to apply to the following test problems. The ideas met here are discussed again when seeking to increase the accuracy of the new method, when the new method is used for solving partial differential equations, in chapter 7.

## Introduction To The Test Problems

3.6 The following test problems have been chosen to ensure that the new method is exposed to the mathematical difficulties which were raised in chapter 2.

3.7 The first problem, a hydraulic actuator circuit, which is a linear and moderately stiff example, is solved by the explicit method given in equation (3.6), and then by the second of the implicit methods, given by equation (3.11). Since the problem being solved is linear, the implicit scheme will reduce to an explicit relationship, and will hence not require an iterative procedure to form the solution at each time step.

3.8 The second problem, a second order differential equation typical of the type that describes the behaviour of a hydraulic actuator, has two sets of data values; one set which leads to a stiff and non-oscillatory problem and another set which leads to a non-stiff and oscillatory problem. The ordinary differential equation is solved by the explicit method alone, and to employ the method a particular strategy must be used to prevent a division by zero. This is fully discussed later.

3.9 The third and fourth problems are hydraulic actuator circuits; one where the actuator is supplied with an input flow through an orifice, and the other where a flow is taken off of the main input flow to the actuator, through an orifice, and back to tank. For both circuits, the hydraulic oil is discharged through an orifice to tank. With the third problem, the orifice flow characteristics are assumed to be linear, and for the fourth problem this assumption is dropped, which leads to a non-linear problem. For both of these problems, the explicit method, and the first implicit method, given by equations (3.6) and (3.7) respectively, are implemented as a predictor-corrector pair. The first implicit method has been chosen because of the ease of implementation in comparison with the second implicit method, particularly when an iterative scheme is employed which will require the evaluation of a Jacobian matrix. Also, because of the similarity of this method to the explicit method, it appears sensible to choose these two methods to work together. To use this pair, a suitable iteration scheme must be found, since these problems, with the data used, lead to matlematically stiff and oscillatory problems. This iteration scheme has been discussed in section 2.70.

**3.10** The final problem that is solved is a fifth order system extracted from a working simulation inside the HASP package. As well as being stiff, oscillatory and non-linear, the problem also encounters discontinuities and so provides an opportunity to examine the method when it used to solve a problem which has all of these difficulties together. This problem is again solved by the predictor-corrector pair discussed above, and the evaluation of the Jacobian to be used with the iteration scheme is explained in some detail.

**3.11** Before solving these problems, some aspects of the computer programming involved are discussed so as to highlight important areas which must be considered.

## Software Consideration And C.P.U. Timing

**3.12** All of the programs presented and discussed in this thesis were written in FORTRAN 77 and run on the Vax 750 computer within the School of Engineering. The new method has been programmed in the form of a general purpose integration method, with separate routines handling the explicit and the implicit methods. By using function routines for the coefficient evaluations, it is possible to solve different problems without rewriting the the main structure of the program.Theflow chart given in Figure 3.1 illustrates the action of the program. The user has the choice of:

i) which fixed-step method is to be used
ii) the final time, until which the integration proceeds
iii) the value of the time step used throughout the integration
iv) the time step used throughout the integration
v) graphical or screen output
vi) whether to re-run the program with a different time step.

A fully commented version of a program implementing the new method can be found in Appendix A.

**3.13 Measurement of C.P.U. time.** Since the amount of C.P.U. time used by the integration method to solve stiff problems is paramount in this work, the method used for measuring the C.P.U. time on the computer system that the work was conducted upon is discussed.

**3.14** A system timing function has been used, and a commented version of the coding for this function can be found in Appendix A. The function, called TIM, when it is called, records the C.P.U. clock measurement at a designated instant and is accurate to within microseconds [61]. Consequently, if the function is invoked either side of a call statement to a routine, it is possible, by the subtraction of the value returned the first time that the function is used from the value returned the second time that it is used, to record the C.P.U. time spent in the routine. For example, if the subroutine FSTORM(X,Y,T) is called, and the amount of C.P.U. time used by this routine is required, then the following coding would provide the necessary information.

```
ATIME = TIM(TIME)

CALL FSTORM(X,Y,T)

BTIME = TIM(TIME)

TTIME = BTIME - ATIME
```

TTIME will hold the total C.P.U. time spent in the routine FSTORM, measured in microseconds. This measurement of time will also include a measurement of the duration of the calling process, which is assumed to be insignificant in relation to the time spent in the routine. TTIME is converted, using another function called CONVER, from microseconds into days, hours, minutes, seconds, tenths of seconds and hundredths of seconds. This is the form that is presented to the user.

## Application Of The Method To Test Problem 1 - Hydraulic Actuator Circuit

**3.15** The diagram describing this circuit is shown in Figure 3.2. The actuator is assumed to be moving initially at a constant velocity, and then encounters a step increase in force opposing the load. This causes the actuator to decelerate and the system pressure to subsequently rise. Further assuming, both that the increase in pressure causes the relief valve to operate, and that there is a viscous friction associated with the load; then to describe the behaviour of the system mathematically, the following equations are needed:

$$\frac{dP}{dt} = \frac{B}{v}(Q_p - Q_a - Q_r) \qquad (3.12)$$

$$\frac{du}{dt} = \frac{PA_r}{M} - \frac{uf}{M} - \frac{F}{M} \qquad (3.13)$$

where:

$A_r$ is the area of the actuator piston
B is the bulk modulus of the hydraulic oil
f is the viscous friction coefficient
F is the force opposing the load
M is the mass of the load
P is the pressure of the oil on the piston side of the actuator, the pressure on the rod side is taken to be negligible
$Q_a$ is the actuator flow rate
$Q_p$ is the pump flow rate
$Q_r$ is the relief valve flow rate
u is the velocity of the actuator piston
v is the combined pipe and actuator volume, on the piston side

Taking

$$Q_a = A_r u \qquad \textbf{(3.14a)}$$

and the flow discharged through the relief valve to be

$$Q_r = k_r(P - P_c) \qquad \textbf{(3.14b)}$$

where:

$k_r$ is the flow coefficient of the relief valve

$P_c$ is the relief valve cracking pressure

then, substituting equations (3.14a) and (3.14b) into equation (3.12), and rearranging the resultant equation gives:

$$\frac{dP}{dt} = \frac{B}{v}(Q_p + k_r(P_c - P) - A_r u) \tag{3.15}$$

For the data:

$A_r = 0.0015 \text{ m}^2$
$B = 1.4 \times 10^9 \text{ N/m}^2$
$f = 800 \text{ N/m/sec}$
F stepped from $14.6 \times 10^3$ N
to $15.6 \times 10^3$ N
$k_r = 1.667 \times 10^{-9}\text{m}^3 \text{/sec/N/m}^2$
$M = 1000$ Kg
$Q_p = 7.5 \times 10^{-4} \text{ m}^3\text{/sec}$
$v = 1 \times 10^{-3} \text{ m}^3$
P - initial value $= 1 \times 10^7 \text{ N/m}^2$
u - initial value $= 0.5$ m/sec

then the problem is stiff and non-oscillatory, with the eigenvalues corresponding to the problem being real and negative, and differing widely in magnitude.

**3.16 The explicit method applied to problem 1.** Rewriting equations (3.13) and (3.15) in a form suitable for the new method, namely:

$$\frac{dP}{dt} = \frac{-k_r B}{v} P + \frac{B}{v}(Q_p + k_r P_c - A_r u) \tag{3.16}$$

$$\frac{du}{dt} = \frac{-f}{M} u + \frac{1}{M}(PA_r - F) \tag{3.17}$$

The coefficients can now be formulated in order to use the explicit method given in equation (3.6). These are, at time $t = t_n$:

$$a_{1n} = \frac{-k_r B}{v} \qquad b_{1n} = \frac{B}{v}(Q_p + k_r P_c - A_r u_n) \tag{3.18}$$

$$a_{2n} = \frac{-f}{M} \qquad b_{2n} = \frac{1}{M}(P_n A_r - F) \tag{3.19}$$

where the subscript n denotes the state variable value at time $t = t_n$.

**3.17** Hence the scheme which will give the numerical solution at advancing time steps is:

$$P_{n+1} = \frac{b_{1n}}{a_{1n}}(e^{a_{1n}h} - 1) + P_n e^{a_{1n}h} \tag{3.20}$$

$$u_{n+1} = \frac{b_{2n}}{a_{2n}}(e^{a_{2n}h} - 1) + u_n e^{a_{2n}h} \tag{3.21}$$

where the subscript n+1 denotes the state variable value at time $t = t_{n+1}$.

**3.18 The second implicit method applied to problem 1.** After solving the problem using the explicit method, it was then solved using the implicit method given in equation (3.11). This method was chosen since the problem, when written in the form $y' = ay + b$ as in equations (3.1) and (3.2), will have a variable b term. The coefficients for the implicit method will remain the same as in equations (3.18) and (3.19) and the scheme will reduce to give a linear relationship, since the problem being solved is linear. The solution scheme for pressure is given by:

$$P_{n+1} = [1 + \frac{A_r^2 B}{vMa_{1n}a_{2n}}(\frac{(e^{a_{2n}h} - 1)}{a_{2n}h} - 1)(\frac{(e^{a_{2n}h} - 1)}{a_{1n}h} - 1)]^{-1}[Z_1 + Z_2 + Z_3 + Z_4]$$

where:

$$Z_1 = \frac{b_{1n}}{a_{1n}}(e^{a_{1n}h} - \frac{(e^{a_{1n}h} - 1)}{a_{1n}h}) + P_n e^{a_{1n}h}$$

$$Z_2 = \frac{A_r B}{va_{1n}}[\frac{(e^{a_{1n}h} - 1)}{a_{1n}h} - 1][\frac{b_{2n}}{a_{2n}}(e^{a_{2n}h} - \frac{(e^{a_{2n}h} - 1)}{a_{2n}h}) + u_n e^{a_{2n}h}]$$

$$Z_3 = \frac{B}{va_{1n}}[Q_p + kP_c][\frac{(e^{a_{1n}h} - 1)}{a_{1n}h} - 1]$$

$$Z_4 = \frac{A_r BF}{v a_{1n} a_{2n} M}\left[\frac{(e^{a_{2n}h} - 1)}{a_{2n}h} - 1\right]\left[\frac{(e^{a_{1n}h} - 1)}{a_{1n}h} - 1\right]$$

The solution scheme for velocity is represented by a similar expression.

**3.19 Results.** The two methods have been applied to the problem in a single-step form using a variety of different time steps. The corresponding results for each of these time steps has been monitored. Figure 3.3 shows the results obtained by applying the explicit method with two different fixed integration step sizes. The figure shows the velocity and pressure response curves, and in each case the exact solution determined by analytical means [2]. There is a decrease in accuracy as the step size increases, which is characteristic of all numerical methods. However, solutions accurate to two decimal places have been gained using a steplength of 0.05 seconds. Gear's method, when applied to the same problem, works with a low order integrator and uses step sizes in the region of $10^{-3}$ and $10^{-4}$ seconds in order to obtain results of a similar accuracy [2]. The Backward Euler method required 4000 steps to solve the problem on the interval $t = [0 , 4]$ seconds, whereas the new explicit method took only 80 steps to solve the problem on the same interval.

**3.20** Figure 3.4 shows the results for the same problem obtained by applying the implicit method with a fixed integration step size of 0.5 seconds. As is shown, a larger step size is used with the implicit method than with the explicit method in order to obtain a higher degree of accuracy. This indicates that the implicit form of the method is more accurate than the explicit form, and the local error for each of the methods is compared in chapter 4 in an attempt to understand these findings.

**3.21** Given the data values in section 3.15, then the initial stiffness ratio of the system was calculated to be $\approx 1.4 \times 10^3$, which indicates a moderately stiff problem, and hence a good initial test for the method. With the final values taken at $t = 4$ seconds, then the explicit method took 0.12 seconds of C.P.U. time to run, operating with a step size of 0.05 seconds. The implicit method took 0.13 seconds of C.P.U. time to run, operating with a step size of 0.05 seconds, but only took 0.02 seconds of C.P.U. time to run with a step size of 0.5 seconds, which gave a more accurate solution than the explicit method gave using a step size of 0.05 seconds. As a comparison, the Backward Euler method took over 14 seconds to solve the same problem.

**Application Of The Method To Test Problem 2 - Second Order System**

3.22 The behaviour of a linear actuator can be described by a second order differential equation of the form:

$$\frac{d^2x}{dt^2} + K_1\frac{dx}{dt} + K_2x = K_3 \qquad (3.22)$$

where:

$x$ is displacement
$\frac{dx}{dt}$ is velocity
$\frac{d^2x}{dt^2}$ is acceleration

For this problem $K_1$ ,$K_2$ and $K_3$ are considered as being constant. By changing their values, the solution to the problem can be made oscillatory or non-oscillatory, and it is possible to obtain a stiff system also, thus providing a useful test for the new method. The data values are:

**Problem A**

$K_1 = 1$, $K_2 = 50$, $K_3 = 10$

With Initial Conditions

$x = 100$ m, $\frac{dx}{dt} = 0$ m/sec

**Problem B**

$K_1 = 200, K_2 = 10, K_3 = 0$

With Initial Conditions

$x = 100$ m, $\frac{dx}{dt} = 0$ m/sec

Equation (3.22) must be reduced to two first order differential equations in order to be in form suitable for solution by the new method. This technique was shown in section 2.21.

The transformation

$$\frac{dx}{dt} = u$$

will allow equation (3.22) to be rewritten as the system

$$\frac{dx}{dt} = u \tag{3.23}$$

$$\frac{du}{dt} = -K_1u - K_2x + K_3 \tag{3.24}$$

The new method can be applied straightforwardly to equation (3.24), but the "a" coefficient in equation (3.23) is zero, and since the method uses the term $\frac{b}{a}$, manipulation is required before the new method can be used, to avoid a division by zero. One easily applicable solution is to add and subtract x to the right-hand side of equation (3.23), viz.

$$\frac{dx}{dt} = -x + (u + x) \tag{3.25}$$

Consequently, the coefficients can be formulated in order to use the explicit method, given by equation (3.6) .These coefficients are, at time $t = t_n$:

$$a_{1n} = -1 \qquad b_{1n} = u_n + x_n \tag{3.26}$$

$$a_{2n} = -K_1 \qquad b_{2n} = -K_2 x_n + K_3 \tag{3.27}$$

The scheme giving the numerical solution at advancing time steps will then be:

$$x_{n+1} = \frac{b_{1n}}{a_{1n}}(e^{a_{1n}h} - 1) + x_n e^{a_{1n}h} \tag{3.28}$$

$$u_{n+1} = \frac{b_{2n}}{a_{2n}}(e^{a_{2n}h} - 1) + u_n e^{a_{2n}h} \tag{3.29}$$

**3.23 Results.** The data values given in section 3.11 lead to an oscillatory problem for case A, and a stiff and non-oscillatory problem for case B. The results obtained for displacement and velocity in problem A are shown in Figure 3.5. The results obtained are of the same form as the true solution for displacement and velocity shown in Figure 3.6. The new method gives results within 0.05 % of the true solution using a steplength of 0.1 seconds, and the C.P.U. time taken to solve the problem on the interval $t = [0\ ,\ 2]$ seconds was 0.10 seconds.

**3.24** The results found for problem B are shown in Figure 3.7. The displacement, $x$, is shown, and this gives a good indication of the results found by the new method. The stiffness ratio of the system for problem B is $\approx 4 \times 10^3$. Using the explicit method once again, results agreeing with the true solution to within 0.25 % were found using a step size of 0.01 seconds. The C.P.U. time taken to solve the problem over the interval $t = [0\ ,\ 2]$ seconds was 0.7 seconds.

**3.25** A similar problem was solved by Leung [2]. He however, did not use the idea given in equation (3.25). He solved the problem using the new method to solve equation (3.24), coupled with the Backward Euler method to solve equation (3.23).

The results he found for problem A are shown in Figure 3.8 for two different time steps, and are not satisfactory, since the trace of the solution is represented by a series of straight lines rather than a smooth curve. Also the results he found are very inaccurate.

## Convergence Criterion For The Iteration Scheme

**3.26** Before solving the problems using a predictor-corrector pair, it is important to discuss the convergence of the iteration scheme employed by the corrector. This section demonstrates the convergence criterion that have been used in conjunction with the Newton iteration scheme, which has been used in problems 3, 4 and 5. The predictor-corrector pair has been applied in a iteration to convergence mode. When trying to find the roots of $\underline{F}(\underline{x}) = \underline{0}$, then the Newton-Raphson iteration scheme given by equation (2.72) is used. Equation (2.72) is:

$$\underline{x}_{n+1}^{(s+1)} = \underline{x}_{n+1}^{(s)} - [\underline{F}'(\underline{x}_{n+1}^{(s)})]^{-1}[\underline{F}(\underline{x}_{n+1}^{(s)})] \qquad s = 0,1,2,...$$

For problem 3, for example, $\underline{F}'(\underline{x}_{n+1}^{(s)})$ is a (3x3) matrix, $\underline{x}_{n+1}^{(s)}$ is the previous estimate to the root of $\underline{F}(\underline{x}) = \underline{0}$, and $\underline{x}_{n+1}^{(0)}$ is the initial estimation provided by the explicit method. $\underline{F}(\underline{x}_{n+1}^{(s)})$ is evaluated, which gives a (3x1) column vector. In the corrector subroutine, whilst the iterations are being performed, there is a set of FORTRAN IF statements that determines whether the iterations have converged, and if not, how they must proceed. Figure 3.9 illustrates the action of iterations inside the corrector subroutine.

**3.27** When $\underline{F}(\underline{x}_{n+1}^{(s)})$ has been evaluated, then if all the elements of the vector are less than a pre-given tolerance, the iterations will cease, since these elements are considered not to contribute towards a significant change in the estimated solution. If this condition is not satisfied, then the elements of $\underline{F}(\underline{x}_{n+1}^{(s)})$ are multipied by the inverse of the Jacobian matrix, and the resultant column vector subtracted from $\underline{x}_{n+1}^{(s)}$ to give a new estimate to the solution, $\underline{x}_{n+1}^{(s+1)}$. The inverse of the Jacobian is determined by a routine from the NAG library, namely F01AAF [62], and is found to within a set tolerance defined by the use, although alternatively, the set of equations formed could have been solved.

3.28 A mixed error test is now performed on the iterates to see if convergence has occurred. This error test takes the form of evaluating $\dfrac{||\underline{x}_{n+1}^{(s+1)} - \underline{x}_{n+1}^{(s)}||}{1 + ||\underline{x}_{n+1}^{(s)}||}$, where $||.||$ is the max norm, and checking to see if this is less than a pre-set tolerance. If this error test is satisfied, then the last iterate $\underline{x}_{n+1}^{(s+1)}$, is taken as the estimate to the solution. If the test is not satisfied, then the iterations continue, and a new $\underline{F}'(\underline{x}_{n+1}^{(s)})$ is evaluated with s being updated by one. A new Jacobian is also evaluated for a non-linear problem if the iterations do not converge with the old Jacobian values. The Jacobian is not updated at every iteration since this requires more computational effort, and the solution obtained by doing so is not necessarily any more accurate [63].

**Application Of The Method To Test Problem 3 - Hydraulic Actuator Circuit**

3.29 The circuit for this problem is given in Figure 3.10. The actuator is supplied with a constant input flow, and the hydraulic oil is discharged through the orifices to tanks. The assumption is made that the orifice flow characteristics are linear since this will lead to a linear problem. Considering the behaviour of the individual components, it is possible to represent the system with the following set of equations:

$$\frac{dP_1}{dt} = \frac{B}{v_1}(Q_i - k_1P_1 - A_ru) \tag{3.30}$$

$$\frac{dP_2}{dt} = \frac{B}{v_2}(A_r u - k_2 P_2) \tag{3.31}$$

$$\frac{du}{dt} = -\frac{f}{M}u + \frac{A_r}{M}(P_1 - P_2) \tag{3.32}$$

where:

$A_r$ is the cross-sectional area of the actuator piston
$B$ is the bulk modulus of the hydraulic oil
$f$ is the viscous friction coefficient
$k_1$ is the pressure-flow coefficient of orifice 1
$k_2$ is the pressure-flow coefficient of orifice 2
$M$ is the mass to be moved by the actuator
$P_1$ is the pressure in pipe 1
$P_2$ is the pressure in pipe 2
$Q_i$ is the input flow rate
$u$ is the velocity of the actuator piston
$v_1$ is the combined volume of the actuator and pipe 1
$v_2$ is the combined volume of the actuator and pipe 2

For the data shown in table 3.1, then the problem is oscillatory. However, for the data shown in table 3.2 then the problem is initially mathematically stiff, with a stiffness ratio of $\approx 1 \times 10^3$

**3.30 Forming the coefficients for the method.** Rewriting equations (3.30), (3.31) and (3.32) into a form suitable for solution by the new method gives:

$$\frac{dP_1}{dt} = -\frac{k_1 B}{v_1}P_1 + \frac{B}{v_1}(Q_i - A_r u) \tag{3.33}$$

$$\frac{dP_2}{dt} = -\frac{k_2 B}{v_2}P_2 + \frac{B}{v_2}(A_r u) \tag{3.34}$$

$$\frac{du}{dt} = -\frac{f}{M}u + \frac{A_r}{M}(P_1 - P_2) \tag{3.35}$$

**3.31** This problem has been solved by the first of the implicit methods, given by equation (3.7). Although the method could be applied in an explicit way, since the "a" coefficients, given below, are constant, the method was applied directly, and so both a predictor and an iteration scheme are needed. The explicit method given by equation

(3.6) was chosen as the predictor, and the coefficients needed by both the explicit and the implicit method are given by, at time $t = t_n$:

$$a_{1n} = -\frac{k_1 B}{v_1} \qquad b_{1n} = \frac{B}{v_1}(Q_i - A_r u_n) \tag{3.36}$$

$$a_{2n} = -\frac{k_2 B}{v_2} \qquad b_{2n} = \frac{B}{v_2}(A_r u_n) \tag{3.37}$$

$$a_{3n} = -\frac{f}{M} \qquad b_{3n} = \frac{A}{M}(P_{1_n} - P_{2_n}) \tag{3.38}$$

3.32 The schemes that provide the predicted values are:

$$P_{1_{n+1}} = \frac{b_{1n}}{a_{1n}}(e^{a_{1n}h} - 1) + P_{1_n} e^{a_{1n}h} \tag{3.39}$$

$$P_{2_{n+1}} = \frac{b_{2n}}{a_{2n}}(e^{a_{2n}h} - 1) + P_{2_n} e^{a_{2n}h} \tag{3.40}$$

$$u_{n+1} = \frac{b_{3n}}{a_{3n}}(e^{a_{3n}h} - 1) + u_n e^{a_{3n}h} \tag{3.41}$$

with the schemes that provide the corrected values being:

$$P_{1_{n+1}} = \frac{b_{1n+1}}{a_{1n+1}}(e^{a_{1n+1}h} - 1) + P_{1_n} e^{a_{1n+1}h} \tag{3.42}$$

$$P_{2_{n+1}} = \frac{b_{2n+1}}{a_{2n+1}}(e^{a_{2n+1}h} - 1) + P_{2_n} e^{a_{2n+1}h} \tag{3.43}$$

$$u_{n+1} = \frac{b_{3n+1}}{a_{3n+1}}(e^{a_{3n+1}h} - 1) + u_n e^{a_{3n+1}h} \tag{3.44}$$

Considering equations (3.42), (3.43) and (3.44), and putting them into matrix form in order to establish a Newton scheme for this system, then the resultant equation will be

$$\begin{bmatrix} P_{1_{n+1}} \\ P_{2_{n+1}} \\ u_{n+1} \end{bmatrix} = \begin{bmatrix} e^{a_{1n+1}h} & 0 & 0 \\ 0 & e^{a_{2n+1}h} & 0 \\ 0 & 0 & e^{a_{3n+1}h} \end{bmatrix} \begin{bmatrix} P_{1_n} \\ P_{2_n} \\ u_n \end{bmatrix} +$$

$$\begin{bmatrix} e^{a_{1n+1}h}-1 & 0 & 0 \\ 0 & e^{a_{2n+1}h}-1 & 0 \\ 0 & 0 & e^{a_{3n+1}h}-1 \end{bmatrix} \begin{bmatrix} \frac{b_{1n+1}}{a_{1n+1}} \\ \frac{b_{2n+1}}{a_{2n+1}} \\ \frac{b_{3n+1}}{a_{3n+1}} \end{bmatrix} \qquad (3.45)$$

For this problem, then $a_{in+1} = a_{in} \quad i = 1,2,3$

To use the Newton Raphson iteration scheme, $\underline{F}(\underline{x}) = \underline{0}$ must be defined, where $\underline{x} = [P_{1_{n+1}}, P_{2_{n+1}}, u_{n+1}]^T$. Hence, if $\underline{F}(\underline{x})$ is defined as:

$$\begin{bmatrix} P_{1_{n+1}} \\ P_{2_{n+1}} \\ u_{n+1} \end{bmatrix} - \begin{bmatrix} e^{a_{1n+1}h} & 0 & 0 \\ 0 & e^{a_{2n+1}h} & 0 \\ 0 & 0 & e^{a_{3n+1}h} \end{bmatrix} \begin{bmatrix} P_{1_n} \\ P_{2_n} \\ u_n \end{bmatrix} -$$

$$\begin{bmatrix} e^{a_{1n+1}h}-1 & 0 & 0 \\ 0 & e^{a_{2n+1}h}-1 & 0 \\ 0 & 0 & e^{a_{3n+1}h}-1 \end{bmatrix} \begin{bmatrix} \frac{b_{1n+1}}{a_{1n+1}} \\ \frac{b_{2n+1}}{a_{2n+1}} \\ \frac{b_{3n+1}}{a_{3n+1}} \end{bmatrix} \qquad (3.46)$$

then $\underline{F}(\underline{x}) = \underline{0}$

Referring to equation (2.72), Newton's iteration scheme is of the form

$$\underline{x}_{n+1}^{(s+1)} = \underline{x}_{n+1}^{(s)} - [\underline{F}'(\underline{x}_{n+1}^{(s)})]^{-1}[\underline{F}(\underline{x}_{n+1}^{(s)})] \qquad s = 0,1,2,...$$

or

$$\underline{F}'(\underline{x}_{n+1}^{(s)})[\underline{x}_{n+1}^{(s+1)} - \underline{x}_{n+1}^{(s)}] = -\underline{F}(\underline{x}_{n+1}^{(s)})$$

It is necessary to form the Jacobian

$$\underline{F}'(\underline{x}_{n+1}^{(s)}) = \frac{\partial \underline{F}}{\partial \underline{x}} \text{ at } \underline{x}_{n+1}^{(s)}$$

in order to apply the iteration scheme. This Jacobian will be given by the expression:

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} - \begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{vmatrix} -$$

$$\begin{vmatrix} e^{a_{1n+1}h} - 1 & 0 & 0 \\ 0 & e^{a_{2n+1}h} - 1 & 0 \\ 0 & 0 & e^{a_{3n+1}h} - 1 \end{vmatrix} \begin{vmatrix} 0 & 0 & -\dfrac{A_r B}{v_1 a_{1n+1}} \\ 0 & 0 & \dfrac{A_r B}{v_2 a_{2n+1}} \\ \dfrac{A_r}{M a_{3n+1}} & -\dfrac{A_r}{M a_{3n+1}} & 0 \end{vmatrix} \tag{3.47}$$

and so

$$\underline{F}'(\underline{x}_{n+1}^{(s)}) = \begin{vmatrix} 1 & 1 & \dfrac{(e^{a_{1n+1}h} - 1)A_r B}{v_1 a_{1n+1}} \\ 0 & 1 & -\dfrac{(e^{a_{2n+1}h} - 1)A_r B}{v_2 a_{2n+1}} \\ -\dfrac{(e^{a_{3n+1}h} - 1)A_r}{M a_{3n+1}} & \dfrac{(e^{a_{3n+1}h} - 1)A_r}{M a_{3n+1}} & 1 \end{vmatrix} \tag{3.48}$$

Consequently, the predictor-corrector pair can now be used as a complete method to formulate the solution at ascending time-levels for problem 3.

**3.33 Results.** Figures 3.11 and 3.12 show the results obtained using the predictor-corrector pair applied to problem 3. Figure 3.11 shows the results for pressure in pipe 1 for the non-oscillatory problem, whereas Figure 3.12 shows the full results for the oscillatory problem. For both problems, a step size of 0.001 seconds was used. Leung

[2] encountered trouble at this point in his work because of the iteration technique he was using, which is not suitable for stiff problems.

3.34 However, Newton's method gives no trouble for this type of equations, and the time taken to solve the oscillatory problem on the interval t = [0 , 5] seconds using the Newton iteration technique with the corrector was 8.5 seconds. The C.P.U. time taken for the non-oscillatory problem on the same interval was 8.2 seconds. As a comparison, on the same two problems, Gear's method took 21 seconds and 24 seconds respectively [2], to complete the simulation. Gear's method does encorporate a time step control which monitors the local error at each step, unlike a fixed time step method where the local error may be excessively large during the computation. With a fixed step method, although it is possible to check on the accuracy of the results by successively reducing the time step until consecutive sets of solutions show no significant change, stability criteria may not be satisfied throughout the integration process for all the values of the time step that are used.

## Application Of The Method To Test Problem 4 - Hydraulic Actuator

3.35 The circuit for this problem is shown in Figure 3.13. A pump is assumed to operate at a constant pressure and supply hydraulic oil to the actuator via an orifice. The oil is then discharged to tank through another orifice. The major difference between this problem and the last one is that the pressure-flow characteristics of the orifices are not assumed to linear. Considering the behaviour of each individual component, it is possible to represent the system with the following set of equations:

$$\frac{dP_1}{dt} = \frac{B}{v_1}(Q_{01} - A_1u) \tag{3.49}$$

$$\frac{dP_2}{dt} = \frac{B}{v_2}(A_2u - Q_{02}) \tag{3.50}$$

$$\frac{du}{dt} = \frac{1}{M}(P_1A_1 - P_2A_2) - \frac{f}{M} \tag{3.51}$$

where:

$A_1$ is the cross-sectional area of the piston side of the actuator
$A_2$ is the cross-sectional area of the rod side of the actuator
B is the bulk modulus of the hydraulic oil
f is the viscous friction coefficient
M is the mass to be moved by the actuator
$P_1$ is the pressure in pipe 1
$P_2$ is the pressure in pipe 2
$Q_{01}$ is the input flow rate
$Q_{02}$ is the output flow rate
u is the velocity of the actuator piston
$v_1$ is the combined volume of the actuator and pipe 1
$v_2$ is the combined volume of the actuator and pipe 2

The flow rates are given by:

$$Q_{01} = \frac{k_{01}}{\sqrt{(P_s - P_1)}}(P_s - P_1) \tag{3.52}$$

and

$$Q_{02} = \frac{k_{01}}{\sqrt{(P_2)}}P_2 \tag{3.53}$$

where:

$P_s$ is the supply pressure
$k_{01}$ and $k_{02}$ are constants

Equations (3.52) and (3.53) can be rewritten as:

$$Q_{01} = k_a(P_s - P_1) \tag{3.54}$$

$$Q_{02} = k_bP_2 \tag{3.55}$$

where:

$k_a$ is the pressure-flow coefficient of the piston-side orifice
$k_b$ is the pressure-flow coefficient of the annulus-side orifice

and $k_a$ and $k_b$ are given by:

$$k_a = \frac{k_{01}}{\sqrt{(P_s - P_1)}} \tag{3.56}$$

$$k_b = \frac{k_{02}}{\sqrt{P_2}} \tag{3.57}$$

Substituting equation (3.52) into equation (3.49) will transform the differential equation for pressure, $P_1$, into:

$$\frac{dP_1}{dt} = \frac{B}{v_1}\left(\frac{k_{01}}{\sqrt{(P_s - P_1)}}(P_s - P_1) - A_1 u\right) \tag{3.58}$$

Similarly, by substituting equation (3.53) into equation (3.50), the equation for pressure, $P_2$, will become:

$$\frac{dP_2}{dt} = \frac{B}{v_2}\left(A_2 - \frac{k_{02}}{\sqrt{P_2}}P_2\right) \tag{3.59}$$

with the third differential equation, describing the actuator velocity, being:

$$\frac{du}{dt} = \frac{1}{M}(P_1A_1 - P_2A_2) - \frac{f}{M}u \tag{3.60}$$

This problem is the first non-linear problem on which the new method has been tested. For the parametric values given in table 3.3 then the initial eigenvalues for the problem are $-0.576 \times 10^6$, $-0.649 \times 10^2 + 0.567 \times 10^2 j$ ; hence giving a very stiff, oscillatory problem, with an initial stiffness ratio of $\approx 1 \times 10^4$.

**3.36 Formulating the coefficients for the method.** Rewriting equations (3.58), (3.59) and (3.60) into a form suitable for solution by the new method will give:

$$\frac{dP_1}{dt} = -\frac{B}{v_1}\frac{k_{01}}{\sqrt{(P_s - P_1)}}P_1 + \frac{B}{v_1}\left(\frac{k_{01}P_s}{\sqrt{(P_s - P_1)}} - A_1 u\right) \tag{3.61}$$

$$\frac{dP_2}{dt} = -\frac{B}{v_2}\frac{k_{02}}{\sqrt{P_2}}P_2 + \frac{B}{v_2}(A_2 u) \tag{3.62}$$

$$\frac{du}{dt} = -\frac{f}{M}u + \frac{1}{M}(P_1A_1 - P_2A_2) \tag{3.63}$$

This problem has been solved using a predictor-corrector pair, with the explicit method given by equation (3.6) acting as the predictor, and the implicit method given by equation (3.7) acting as the corrector. Again, a Newton-Raphson iteration scheme is used to solve the corrector. .

**3.37** The coefficients needed by both the explicit and the implicit method are given by, at time $t = t_n$:

$$a_{1n} = -\frac{B}{v_{1_n}}\frac{k_{01}}{\sqrt{(P_s - P_{1_n})}} \qquad b_{1n} = \frac{B}{v_{1_n}}\left(\frac{k_{01}P_s}{\sqrt{(P_s - P_{1_n})}} - A_1u_n\right) \tag{3.64}$$

$$a_{2n} = -\frac{B}{v_{2_n}}\frac{k_{02}}{\sqrt{P_{2_n}}} \qquad b_{2n} = \frac{B}{v_{2_n}}(A_2u_n) \tag{3.65}$$

$$a_{3n} = -\frac{f}{M} \qquad b_{3n} = \frac{1}{M}(P_{1_n}A_1 - P_{2_n}A_2) \tag{3.66}$$

In this example, $a_1$ and $a_2$ are non-constant coefficients, but $a_{3n+1} = a_{3n}$ for all values of n. The schemes that provide the predicted values are identical to those in equations (3.39), (3.40) and (3.41), and the schemes that provide the corrected values are identical to those given in equations (3.42), (3.43) and (3.44). Again the matrix form found in equation (3.45) can be established, and in order to use Newton's method, it is necessary to define $\underline{F}(\underline{x}) = \underline{0}$ where $\underline{x} = [P_{1_{n+1}}, P_{2_{n+1}}, u_{n+1}]^T$

**3.38 Forming the Jacobian matrix.** $\underline{F}(\underline{x})$ is defined identically to $\underline{F}(\underline{x})$ in equation (3.46), and the Newton's iteration method used is of the same form as that shown in equation (2.72). The difference between this problem and the last is that the Jacobian matrix is non-constant for this problem. When forming the Jacobian at any time step, the variable coefficients are "frozen" and assumed to be constant, which in effect leads to a 'local' derivative, or 'local' form of the Jacobian, which is applicable during that time step. This idea is dealt with in some depth by Richmeyer and Morton [64]. The

Jacobian for problem 4 is:

$$\begin{vmatrix} 1+\dfrac{(e^{a_{1_{n+1}}h}-1)A_1u_{n+1}}{2k_{01}\sqrt{(P_s-P_{1_{n+1}})}} & 0 & \dfrac{-(e^{a_{1_{n+1}}h}-1)A_1\sqrt{(P_s-P_{1_{n+1}})}}{k_{01}} \\ 0 & 1+\dfrac{(e^{a_{2_{n+1}}h}-1)A_2u_{n+1}}{2k_{02}\sqrt{P_{2_{n+1}}}} & \dfrac{(e^{a_{2_{n+1}}h}-1)A_2\sqrt{P_{2_{n+1}}}}{k_{02}} \\ \dfrac{(e^{a_{3_{n+1}}h}-1)A_1}{f} & -\dfrac{(e^{a_{3_{n+1}}h}-1)A_2}{f} & 1 \end{vmatrix} \qquad (3.67)$$

**3.39 Results.** Figure 3.14 shows the results obtained by using the predictor-corrector pair applied to problem 4. This was the first non-linear problem to be solved by the new method, and a satisfactory set of results was found. The results found using the new method with a fixed step size of 0.001 seconds are those shown in the figure. The solution curves are smoother than those found by using Gear's method, which are shown in Figure 3.15. Gear's method sank to a step size of $1 \times 10^{-10}$ seconds and reverted to a first order method. The new method took 7.8 seconds of C.P.U. time to solve the problem over the interval $t = [0\ ,\ 0.6]$ seconds, whereas Gear's method took 14.8 seconds to solve the problem over the same interval, and the solution given was not totally satisfactory, in the sense that reducing the tolerance value used in the iteration scheme increased the accuracy of the results.

**Application Of The Method To Test Problem 5 - Linear Actuator Operated By A Directional Control Valve**

**3.40** The circuit for this problem, shown in Figure 3.16, consists of a fixed displacement pump supplying fluid to a linear actuator by means of a directional control valve. The directional control valve is manually operated, and is a three position, four way closed centre type. When the directional control valve is centred, flow returns to tank through a relief valve. The actuator is extended by manual application of the directional control valve to a positon approximately one half of its stroke. It is held stationary for a time and then partially retracted to a position approximately one quarter of its stroke, where it is again held stationary.

**3.41 System equations.** The set of equations modelling the above circuit can be written as three first order ordinary differential equations; each describing pressure, and one second order ordinary differential equation describing the behaviour of the linear actuator. These equations will be:

$$\frac{dP_1}{dt} = \frac{B_1}{v_1}(Q_{pu} - Q_r - Q_{dcv_1}) \tag{3.68}$$

$$\frac{dP_2}{dt} = \frac{B_2}{v_2}(Sgn(z)Q_{dcv_2} - Sgn(z)A_1u) \tag{3.69}$$

$$\frac{dP_3}{dt} = \frac{B_3}{v_3}(Sgn(z)A_2u - Sgn(z)Q_{dcv_3}) \tag{3.70}$$

$$\frac{du}{dt} = \frac{1}{M}(P_2A_1 - P_3A_2 - fu - kx - F) \tag{3.71}$$

where:

$A_1$ is the cross-sectional area of the piston side of the actuator
$A_2$ is the cross-sectional area of the rod side of the actuator
$B_1,B_2,B_3$ is the bulk modulus of the hydraulic oil
in each of the three pipes
f is the viscous friction coefficient
M is the mass to be moved by the actuator
$P_1$ is the pressure in pipe 1
$P_2$ is the pressure in pipe 2
$P_3$ is the pressure in pipe 3
$Q_{pu}$ is the pump flow
$Q_r$ is the relief valve flow
$Q_{dcv_{1,2,3}}$ is the directional control valve flow
x is the displacement of the actuator piston
u is the velocity of the actuator piston
k is the spring stiffness
F is the force opposing the load
$v_1$ is the volume of pipe 1
$v_2$ is the combined volume of the actuator and pipe 2
$v_3$ is the combined volume of the actuator and pipe 3

Sgn(z) is a function of z, where z describes the behaviour of the directional control valve (d.c.v.)

$z = 0$ d.c.v. closed
$z = 1$ d.c.v. fully up
$z = -1$ d.c.v. fully down

The pump flow is given by:

$$Q_{pu} = \omega D_p - \frac{C_s P_1 D_p}{\mu} \tag{3.72}$$

where:

$\omega$ is the angular speed of the pump
$D_p$ is the pump displacement
$\mu$ is the viscosity of the hydraulic oil
$C_s$ is the slip loss due to differential pressure

and the relief valve flow is given by:

$$Q_r = k_r(P_1 - P_c) \quad \text{if } P_1 > P_c \tag{3.73}$$

$$Q_r = 0 \quad \text{if } P_1 < P_c \tag{3.74}$$

where:

$k_r$ is the relief valve coefficient
$P_c$ is the cracking pressure of the relief valve

Also, when the directional control valve is open in the upward direction, then

$$Q_{dcv_1} = Q_{dcv_2} = kE_1\sqrt{(P_1 - P_2)} \tag{3.75}$$

and

$$Q_{dcv_3} = kE_2\sqrt{P_3} \tag{3.76}$$

and when it is fully down, then

$$Q_{dcv_1} = Q_{dcv_3} = kE_2\sqrt{(P_1 - P_3)} \tag{3.77}$$

and

$$Q_{dcv_2} = kE_1\sqrt{P_2} \tag{3.78}$$

where:

$kE_1$, $kE_2$ are the flow coefficients of the d.c.v.

Substituting equations (3.72), (3.73) and (3.75) into equation (3.68) will give, when the relief valve cracking pressure has been exceeded, and the d.c.v. is fully up:

$$\frac{dP_1}{dt} = \frac{B_1}{V_1}(\omega D_p - \frac{C_s P_1 D_p}{\mu} - k_r(P_1 - P_c) - kE_1\sqrt{P_1 - P_2}) \qquad (3.79)$$

Similarly, equations (3.69) and (3.70), for the same case, will become:

$$\frac{dP_2}{dt} = \frac{B_2}{V_2}(kE_1\sqrt{(P_1 - P_2)} - A_1 u) \qquad (3.80)$$

$$\frac{dP_3}{dt} = \frac{B_3}{V_3}(A_2 u - kE_2\sqrt{P_3}) \qquad (3.81)$$

**3.42 Stiction logic and cavitation.** The way in which stiction of the load has been modelled is via the use of FORTRAN IF statements. Using equation (3.71), then taking $F_{NET} = P_2A_1 - P_3A_2$, the following logic has been used. If the velocity, u, is zero, then if $|F_{NET}| > F_S$, the value used for stiction, the force opposing the motion has been taken as a coulomb friction value, $F_C$. If, however, $|F_{NET}| < F_S$, then the acceleration has been taken as zero. Once the velocity, u, is non-zero, then F is taken as $F_C$. Taking the band about zero allows stiction to be accurately modelled, i.e. if $|u| > \epsilon$ then $F = F_C$, but $\epsilon$ must be carefully chosen [65].

**3.43** The pipework in the system has been modelled almost entirely as flexible hose. The equations used to represent the pressure in the system take into account air release if the system pressure falls below atmospheric pressure. This is done by use of work established by Dugdale [1]. The bulk modulus of the pipe system is changed if cavitation occurs. The equation describing the change in the effective bulk modulus is:

$$B_e = \frac{1}{\left[\frac{1}{B} + \frac{d_0(P_{sat} - P_i)}{n(P_i + 1)^2}\right]} \qquad (3.82)$$

where:

$P_i$ is the pressure in pipe i, i = 1,2,3
n is the polytropic index of air
$d_0$ is the fraction of air dissolved in the hydraulic fluid at S.T.P.
$B_e$ is the effective bulk modulus of the pipe material

In the system being modelled, no spring force exists, and the gravitational force acting is zero since the actuator is horizontally mounted. The motion of the directional control valve is idealised as a series of discrete steps as shown in Figure 3.17. The data values used are given in tables 3.4 and 3.5. The spring stiffness has been taken as zero.

**3.44 Application of the new method to solve the fifth order system.** The new method has again been applied in a fixed time step predictor-corrector form to solve this problem. The predicted values will be given by the schemes:

$$P_{i_{n+1}} = \frac{b_{in}}{a_{in}}(e^{a_{in}h} - 1) + P_{i_n}e^{a_{in}h} \qquad i = 1,2,3 \tag{3.83}$$

$$u_{n+1} = \frac{b_{4n}}{a_{4n}}(e^{a_{4n}h} - 1) + u_n e^{a_{4n}h} \tag{3.84}$$

$$x_{n+1} = \frac{b_{5n}}{a_{5n}}(e^{a_{5n}h} - 1) + x_n e^{a_{5n}h} \tag{3.85}$$

The schemes giving the corrected values are similar, but the $a_{in}$ and the $b_{in}$, i = 1,2,3,4,5, are replaced by $a_{in+1}$ and $b_{in+1}$ respectively. The coefficient values, when the directional control valve is fully up, are given by:

$$a_{1n} = -\frac{B_1}{v_{1_n}}\left(\frac{C_s D_p}{\mu} + k_r + \frac{kE_1}{\sqrt{(P_{1_n} - P_{2_n})}}\right) \quad b_{1n} = \frac{B_1}{v_{1_n}}\left(\omega D_p + k_r P_c + \frac{kE_1 P_{2_n}}{\sqrt{(P_{1_n} - P_{2_n})}}\right) \tag{3.86}$$

$$a_{2n} = -\frac{B_2}{v_{2_n}}\frac{kE_1}{\sqrt{(P_{1_n} - P_{2_n})}} \qquad b_{2n} = \frac{B_2}{v_{2_n}}\left(\frac{kE_1}{\sqrt{(P_{1_n} - P_{2_n})}}P_{1_n} - A_1 u_n\right) \tag{3.87}$$

$$a_{3n} = -\frac{B_3}{v_{3_n}}\frac{kE_2}{\sqrt{P_{3_n}}} \qquad b_{3n} = \frac{B_3}{v_{3_n}}A_2 u_n \tag{3.88}$$

$$a_{4n} = -\frac{f}{M} \qquad b_{4n} = \frac{1}{M}(P_{2_n}A_1 - P_{3_n}A_2 - F) \tag{3.89}$$

$$a_{5n} = -1 \qquad b_{5n} = (u_n + x_n) \tag{3.90}$$

The coefficient values require appropriate modification when the directional control valve is fully down or shut, but an exhaustive set is not given here. At time $t = t_{n+1}$, the coefficient values are the same, with n replaced by n+1 in each of the formula in equations (3.86) - (3.90), apart from $v_{1_n}, v_{2_n}$ and $v_{3_n}$, which can only be updated at the end of each set of state variable evaluations.

**3.45 Evaluation of the Jacobian matrix.** When the corrector is applied to this problem, it is again used in conjunction with a Newton-Raphson iteration scheme, and so a Jacobian matrix must be evaluated. This, by analogy to equation (3.47), will be:

$$I - D\gamma \tag{3.91}$$

where I is the (5x5) identity matrix,

$$D = \begin{bmatrix} e^{a_{1n+1}h}-1 & 0 & 0 & 0 & 0 \\ 0 & e^{a_{2n+1}h}-1 & 0 & 0 & 0 \\ 0 & 0 & e^{a_{3n+1}h}-1 & 0 & 0 \\ 0 & 0 & 0 & e^{a_{4n+1}h}-1 & 0 \\ 0 & 0 & 0 & 0 & e^{a_{5n+1}h}-1 \end{bmatrix}.$$

and for each of the three positions of the directional control valve, since there are different coefficient values, there will be a different matrix, $\gamma$. For the directional control valve fully up, then

$$\gamma = \begin{vmatrix} \alpha & \beta & 0 & 0 & 0 \\ \dfrac{A_1 u_n}{2kE_1\sqrt{(P_{1_n} - P_{2_n})}} - 1 & -\dfrac{A_1 u_n}{2kE_1\sqrt{(P_{1_n} - P_{2_n})}} & 0 & \dfrac{A_1\sqrt{(P_{1_n} - P_{2_n})}}{kE_1} & 0 \\ 0 & 0 & -\dfrac{A_2 u_n}{2kE_2\sqrt{P_{3_n}}} & -\dfrac{A_2\sqrt{P_{3_n}}}{kE_2} & 0 \\ 0 & -\dfrac{A_1}{f} & \dfrac{A_2}{f} & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 \end{vmatrix}$$

where

$$\alpha = \frac{-kE_1}{2(P_{1_n} - P_{2_n})^{\frac{3}{2}}} \left[ \frac{C_s}{\mu} + k_r + \frac{kE_1 P_{2_n}}{\sqrt{(P_{1_n} - P_{2_n})}} \right]^{-1} \left[ 1 - (\omega D_p + k_r P_c + \frac{kE_1 P_{2_n}}{\sqrt{(P_{1_n} - P_{2_n})}})(\frac{C_s D_p}{\mu} + k_r + \frac{kE_1 P_{2_n}}{\sqrt{(P_{1_n} - P_{2_n})}})^{-1} \right]$$

and

$$\beta = -2(P_{1_n} - P_{2_n})[1 + \frac{P_{2_n}}{2(P_{1_n} - P_{2_n})}]\,\alpha$$

and consequently, the Jacobian matrix can be evaluated for this case using equation (3.91). The corrector has been applied in an iteration to convergence mode as in the last two problems.

**3.46 Results and discussion.** Several sets of results were obtained by varying the data values, the integration time step and the final limit of the real simulation time. Figure 3.18 shows the simulated response of piston pressure with respect to time and Figure 3.19 the actuator displacement with respect to time. Figure 3.20 is a reproduction of experimental results obtained in the laboratory [4]. This shows the actuator piston pressure and displacement responses with respect to time.

**3.47** The results found by the method compare accurately with the experimental results and the results obtained using Gear's method, which has also been used to solve this problem. The piston pressure initially rises and after a period of oscillation during which cavitation occurs, settles out to a constant level. When the flow to the actuator is shut off by the directional control valve, the actuator settles at a steady level after a severe period of oscillation. The simulated and experimental actuator displacements show favourable agreement. The only real differences occur with the peak following an initial fall in piston pressure.

**3.48 C.P.U. time used by the new method.** The method performed well in comparison with Gear's method. For the standard data given in tables 3.4 and 3.5 the highest stiffness ratio for the problem is $\approx 1 \times 10^3$. This was found by analysing the Jacobian matrix for the problem, and numerically extracting the eigenvalues throughout the simulation. However, the problem was also oscillatory and discontinuous, and the single step method coped adequately with these difficulties. To complete the simulation on the interval $t = [0 , 6]$ seconds, then the new method took 8 minutes and 42 seconds C.P.U. time, using a fixed time step of $5 \times 10^{-5}$. Gear's method took 3 minutes and 41.2 seconds to solve the problem on the same interval. When the data was changed, in order to both increase the highest stiffness ratio to $\approx 1 \times 10^5$ and the frequency of the oscillations, then the new method gave the results shown in Figures 3.21 and 3.22. This time the new method completed the simulation in 1 hour 1 minute and 36 seconds. Gear's method became unstable and Figures 3.23 and 3.24 show the results given. With a reduced tolerance Gear's method completed the simulation in 30 minutes and 50 seconds, and gave similar results to those shown in Figures 3.21 and 3.22.

**3.49** Tables 3.6 and 3.7 give a comprehensive set of C.P.U. times for the five test problems. For test problems 1 and 2, the new method was much faster than the

Backward Euler method. For problems 3 and 4, the new method, as a predictor-corrector pair, proved to be faster than Gear's method. The only problem for which the new method was slower than Gear's method was number 5, and the C.P.U. times for both methods proved to be long, particularly when the stiffness ratio and the oscillatory behaviour is increased. The new method would benefit from a time step control when it is used to solve this problem.

## Conclusions

**3.50** A fixed time step algorithm now exists for an explicit and implicit form of the new integration method. When applied to a set of test equations designed to analyse the method, encouraging and satisfactory results have been forthcoming. However, the results found when applying the method to a stiff system must be taken into perspective. Firstly, the stiffness ratios quoted for the problems solved are only the ratios at a given time, and it must be appreciated that they vary since they are functions of time. Secondly, comparisons of the method do not take into account that Gear's method is a variable time step algorithm, which ensures that local error criteria are not violated. When applying the method in fixed step form, the accuracy of the results is hard to analyse. Also, the theoretical stability criteria may not be satisfied. The fast C.P.U. times recorded by the method, in comparison with other methods, in the test problems prior to problem 5, may not fully illustrate the qualities of the method, since The method may be bordering on the verge of instability throughout the integration. Similarly, the C.P.U. times from problem 5, for the new method, although slower than the times recorded for Gear's method, may be due to the small time step that must be taken throughout the computation to ensure that the method does not become unstable, during very stiff or osillatory portions of the integration. This small time step may be much smaller than the time step needed throughout other parts of the simulation, but as it must be stipulated at the beginning, and cannot be changed throughout the computation, it presents a drawback to the method.

**3.51** Consequently, to test thoroughly the new method and study its potential as a competitive numerical integration algorithm inside of the HASP package, it is necessary to:

i) construct a reliable time step control that monitors the local error
ii) establish the stability properties of the method
iii) generalise the algorithm for the method completely, which will include the numerical estimation of the Jacobian matrix
iv) apply the method to a comprehensive set of differential equations

These issues form the basis of the work presented in the following chapters.

| Symbol | Description | Value |
|---|---|---|
| $A_r$ | Area of the actuator piston | $2 \times 10^{-3}$ m$^2$ |
| B | Bulk modulus of the hydraulic oil | $1.8 \times 10^{9}$ N/m$^2$ |
| f | Viscous friction coefficient | $8 \times 10^{2}$ N/m/sec |
| $k_1$ | Flow coefficient for orifice 1 | $3.2 \times 10^{-10}$ m$^3$/sec/N/m$^2$ |
| $k_2$ | Flow coefficient for orifice 2 | $3.2 \times 10^{-10}$ m$^3$/sec/N/m$^2$ |
| M | Mass to be moved by the actuator | $1 \times 10^{3}$ Kg |
| $Q_i$ | Input flow rate | $3 \times 10^{-3}$m$^3$/sec |
| $v_1$ | Initial combined volume of actuator and pipe 1 | $1 \times 10^{-3}$m$^3$ |
| $v_2$ | Initial combined volume of actuator and pipe 2 | $1 \times 10^{-3}$m$^3$ |
| $P_1$ | Oil pressure in pipe 1 - initial value | 0 N/m$^2$ |
| $P_2$ | Oil pressure in pipe 2 - initial value | 0 N/m$^2$ |
| u | Velocity of the piston - initial value | 0 m/sec |

**TABLE 3.1 DATA VALUES USED FOR PROBLEM 3 (OSCILLATORY PROBLEM)**

| Symbol | Description | Value |
|---|---|---|
| $A_r$ | Area of the actuator piston | $2 \times 10^{-3}$ m$^2$ |
| B | Bulk modulus of the hydraulic oil | $1.8 \times 10^9$ N/m$^2$ |
| f | Viscous friction coefficient | $8 \times 10^2$ N/m/sec |
| $k_1$ | Flow coefficient for orifice 1 | $3.2 \times 10^{-7}$ m$^3$/sec/N/m$^2$ |
| $k_2$ | Flow coefficient for orifice 2 | $1.6 \times 10^{-7}$ m$^3$/sec/N/m$^2$ |
| M | Mass to be moved by the actuator | $1 \times 10^3$Kg |
| $Q_i$ | Input flow rate | $3 \times 10^{-3}$m$^3$/sec |
| $v_1$ | Initial combined volume of actuator and pipe 1 | $1 \times 10^{-3}$m$^3$ |
| $v_2$ | Initial combined volume of actuator and pipe 2 | $1 \times 10^{-3}$m$^3$ |
| $P_1$ | Oil pressure in pipe 1 - initial value | 0 N/m$^2$ |
| $P_2$ | Oil pressure in pipe 2 - initial value | 0 N/m$^2$ |
| u | Velocity of the piston - initial value | 0 m/sec |

**TABLE 3.2 DATA VALUES USED FOR PROBLEM 3 (NON-OSCILLATORY PROBLEM)**

| Symbol | Description | Value |
|---|---|---|
| $A_1$ | Area of the actuator piston - piston side | $19.6 \times 10^{-4} m^2$ |
| $A_2$ | Area of the actuator piston - rod side | $14.7 \times 10^{-4} m^2$ |
| B | Bulk modulus of the hydraulic oil | $1.8 \times 10^9$ N/m$^2$ |
| f | Viscous friction coefficient | $4 \times 10^3$N/m/sec |
| $k_{01}$ | Flow coefficient for orifice 1 | $3.2 \times 10^{-12}$ m$^3$/sec/N/m$^2$ |
| $k_{02}$ | Flow coefficient for orifice 2 | $3.2 \times 10^{-12}$ m$^3$/sec/N/m$^2$ |
| M | Mass to be moved by the actuator | $1 \times 10^3$ Kg |
| $v_1$ | Initial combined volume of actuator and pipe 1 | $1 \times 10^{-3} m^3$ |
| $v_2$ | Initial combined volume of actuator and pipe 2 | $1 \times 10^{-3} m^3$ |
| $P_1$ | Oil pressure in pipe 1 - initial value | 0 N/m$^2$ |
| $P_2$ | Oil pressure in pipe 2 - initial value | 0 N/m$^2$ |
| u | Actuator velocity - initial value | 0 m/sec |

**TABLE 3.3 DATA VALUES USED FOR PROBLEM 4**

| Symbol | Description | Value |
|---|---|---|
| $A_1$ | Area of the actuator piston - piston side | $2.03 \times 10^{-3}m^2$ |
| $A_2$ | Area of the actuator piston - rod side | $1.54 \times 10^{-3}m^2$ |
| $B_1$ | Bulk modulus of the hydraulic oil in pipe 1 | $1.2 \times 10^9$ N/m$^2$ |
| $B_2$ | Bulk modulus of the hydraulic oil in pipe 1 | $7 \times 10^8$ N/m$^2$ |
| $B_3$ | Bulk modulus of the hydraulic oil in pipe 1 | $7 \times 10^8$ N/m$^2$ |
| f | Viscous friction coefficient | $5 \times 10^3$N/m/sec |
| $kE_1$ | discharge coefficient for d.c.v. | $4.24 \times 10^{-7}$ m$^3$/sec/N/m$^2$ |
| $kE_2$ | discharge coefficient for d.c.v. | $4.30 \times 10^{-7}$ m$^3$/sec/N/m$^2$ |
| M | Mass to be moved by the actuator | $8.7 \times 10^2$ Kg |
| $v_1$ | Volume of pipe 1 | $1.7 \times 10^{-3}m^3$ |
| $v_2$ | Initial combined volume of actuator and pipe 2 | $4.62 \times 10^{-4}m^3$ |
| $v_3$ | Initial combined volume of actuator and pipe 3 | $1.94 \times 10^{-3}m^3$ |

**TABLE 3.4 DATA VALUES USED FOR PROBLEM 5**

| Symbol | Description | Value |
|---|---|---|
| $P_1$ | Oil pressure in pipe 1 - initial value | $7 \times 10^6$ N/m$^2$ |
| $P_2$ | Oil pressure in pipe 2 - initial value | $1.7 \times 10^6$ N/m$^2$ |
| $P_3$ | Oil pressure in pipe 3 - initial value | $2.1 \times 10^6$ N/m$^2$ |
| $x$ | actuator displacement - initial value | 0 m |
| $u$ | actuator velocity - initial value | 0 m/sec |
| $\omega$ | Angular speed of the pump | $1.57 \times 10^2$ rads/sec |
| $D$ | Pump displacement | $3.5 \times 10^{-6}$ m$^3$/rad |
| $\mu$ | Viscosity of the hydraulic oil | $6.974 \times 10^{-2}$ Nsec/m$^2$ |
| $C_A$ | Slip loss due to differential pressure | $7 \times 10^{-9}$ |
| $n$ | Polytropic index of air | 1.4 |
| $d_0$ | Fraction of air dissolved in the hydraulic fluid at S.T.P | 0.1 |
| $k_r$ | Relief valve coefficient | $5 \times 10^{-11}$ m$^3$/sec/N/m$^2$ |
| $P_c$ | Relief valve cracking pressure | $7 \times 10^6$ N/m$^2$ |

**TABLE 3.5 ADDITIONAL DATA VALUES FOR PROBLEM 5**

|  | Explicit Method |  | Implicit Method |  | Backward Euler Method |  |  |
|---|---|---|---|---|---|---|---|
| Test Problem | Step size (secs) | C.P.U. (secs) | Step size (secs) | C.P.U. (secs) | Step size (secs) | C.P.U. (secs) | Maximum Stiffness Ratio |
| Problem 1 t=[0,4] secs | 0.05 | 0.12 | 0.05<br>0.5 | 0.13<br>0.02 | $1 \times 10^{-3}$ | 14.08 | $1 \times 10^{3}$ |
| Problem 2 Case A t=[0,2] secs | 0.1 | 0.10 | — | -- | $1 \times 10^{-3}$ | 8.4 | 1 |
| Problem 2 Case B t=[0,2] secs | 0.01 | 0.70 | — | -- | $1 \times 10^{-3}$ | 10.8 | $4 \times 10^{3}$ |

**TABLE 3.6 COMPARATIVE RUN TIMES FOR THE SOLUTION OF PROBLEMS 1 AND 2**

|  | Predictor-Corrector Pair |  | Gear's Method |  |  |
|---|---|---|---|---|---|
| Test Problem | Step size (secs) | C.P.U. (secs) | Min. Step size (secs) | C.P.U (secs) | Maximum Stiffness Ratio |
| Problem 3 Case A t=[0,5] secs | $1 \times 10^{-3}$ | 8.5 | $1 \times 10^{-6}$ | 21 | $1 \times 10^{3}$ |
| Problem 3 Case B t=[0,5] secs | $1 \times 10^{-3}$ | 8.2 | $1 \times 10^{-7}$ | 24 | 1.8 |
| Problem 4 t=[0,0.6]secs | $1 \times 10^{-3}$ | 7.8 | $1 \times 10^{-10}$ | 14.8 | $1 \times 10^{4}$ |
| Problem 5 Case A t=[0,6] secs | $5 \times 10^{-5}$ | $5.22 \times 10^{2}$ | $1 \times 10^{-7}$ | $2.212 \times 10^{2}$ | $1 \times 10^{3}$ |
| Problem 5 Case B t=[0,6] secs | $1 \times 10^{-6}$ | $3.696 \times 10^{3}$ | $1 \times 10^{-11}$ | $1.849 \times 10^{3}$ | $1 \times 10^{5}$ |

**TABLE 3.7 COMPARATIVE RUN TIMES FOR THE SOLUTION OF PROBLEMS 3, 4 AND 5**

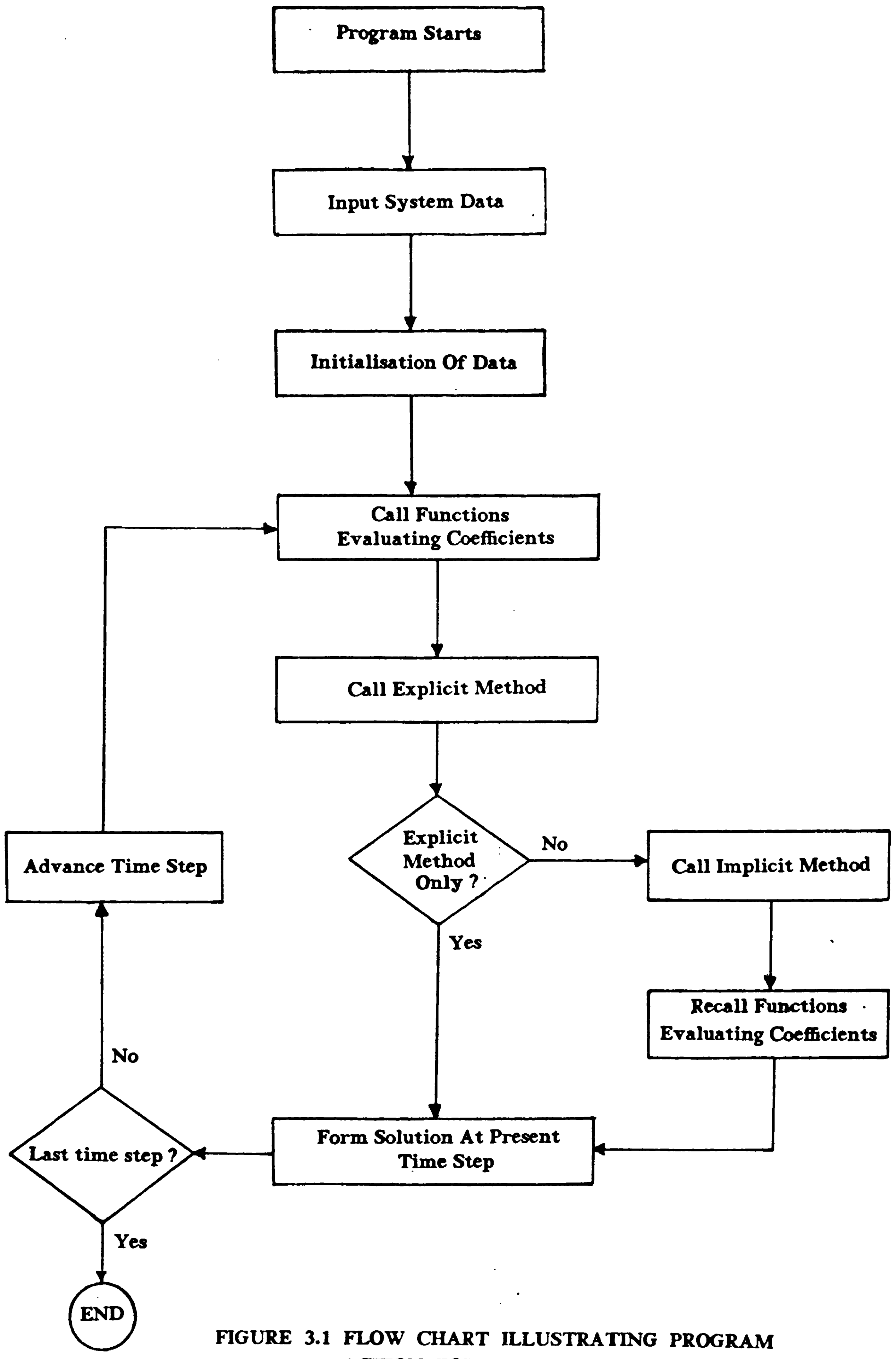


FIGURE 3.1 FLOW CHART ILLUSTRATING PROGRAM ACTION FOR THE NEW METHOD

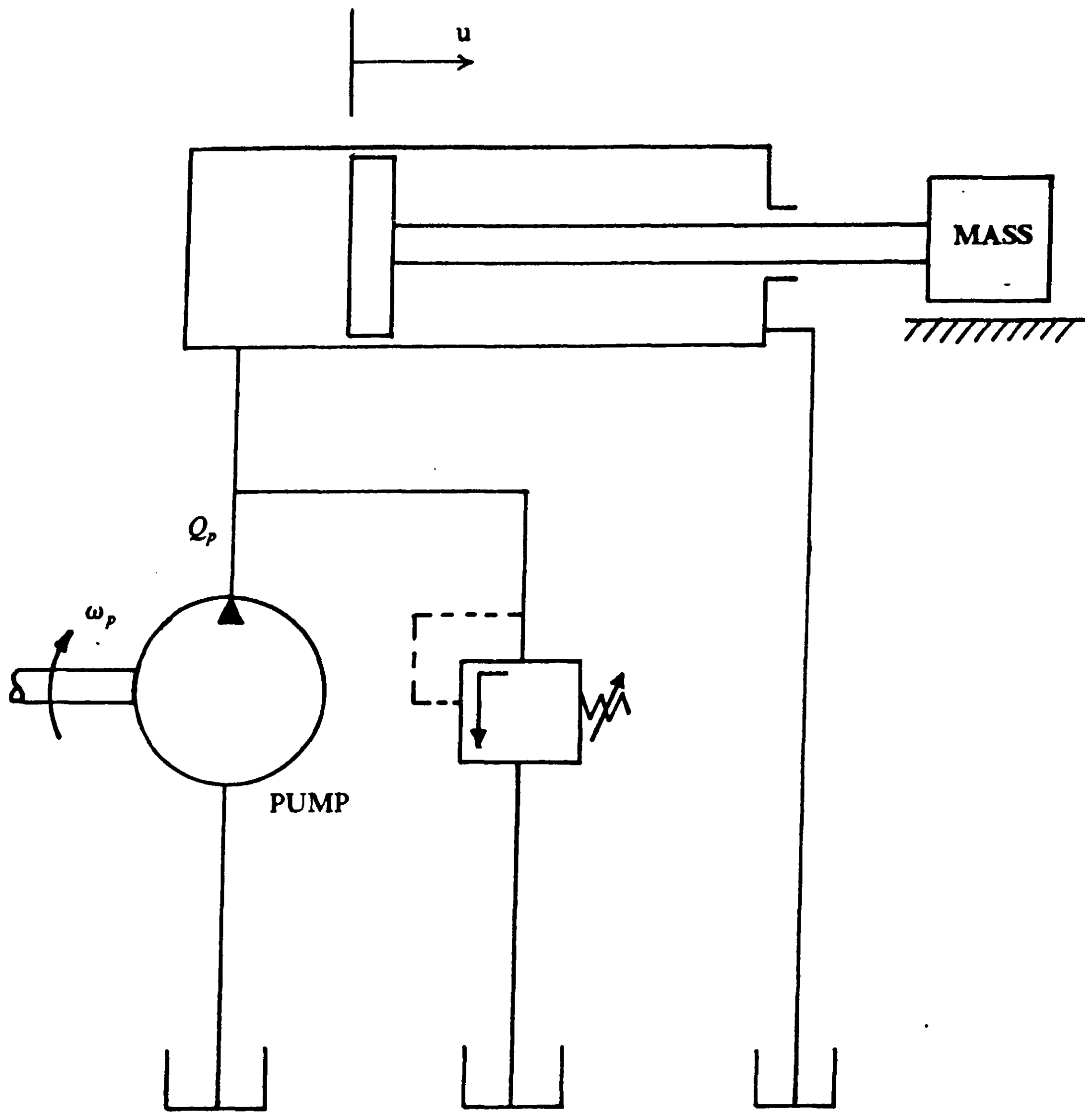
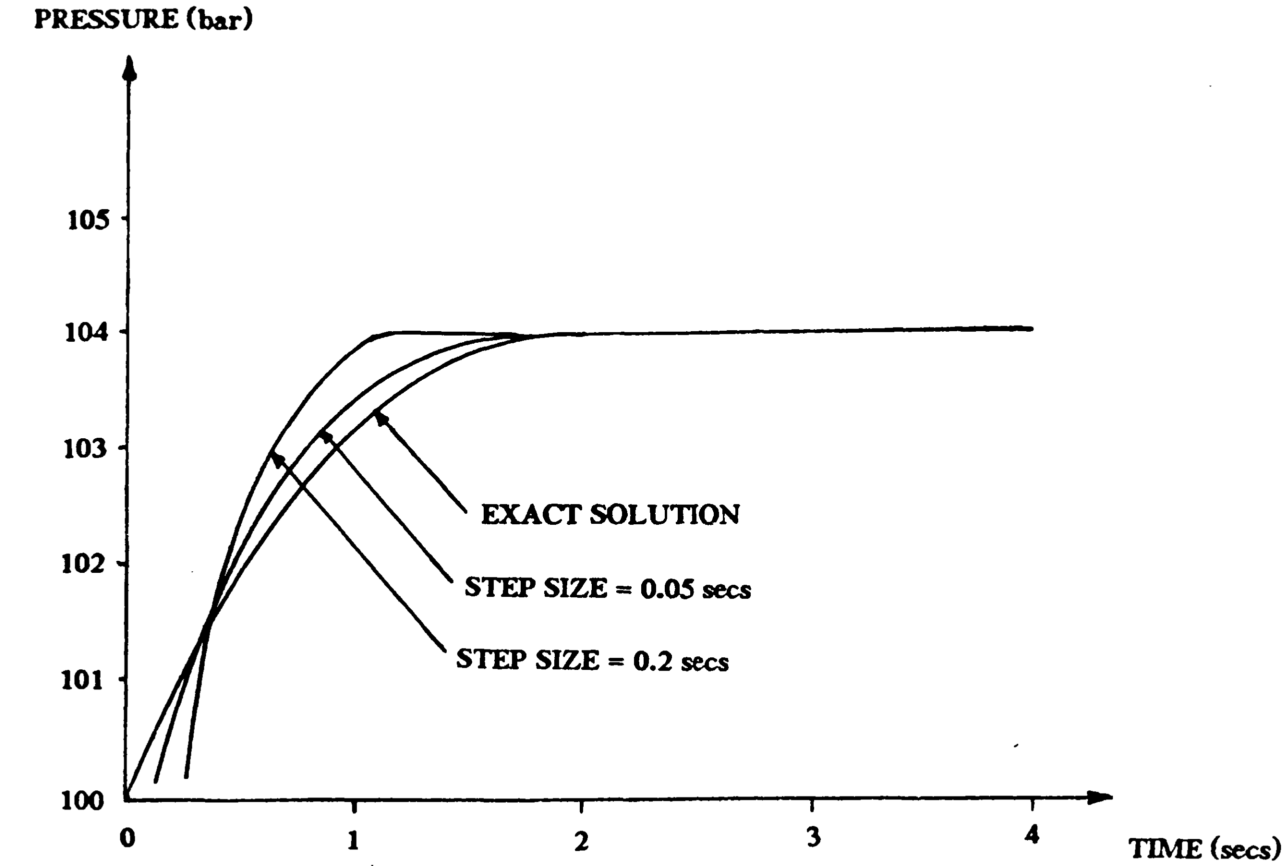


FIGURE 3.2 HYDRAULIC ACTUATOR CIRCUIT



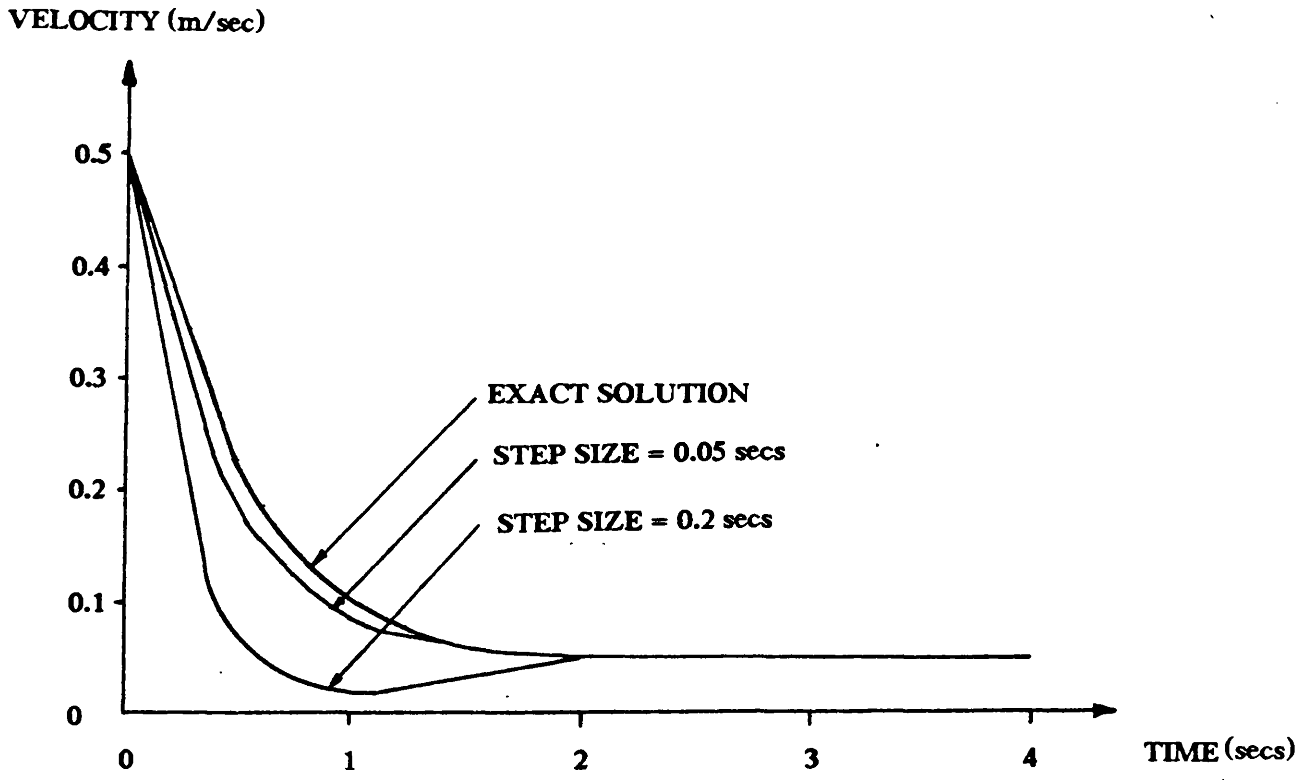
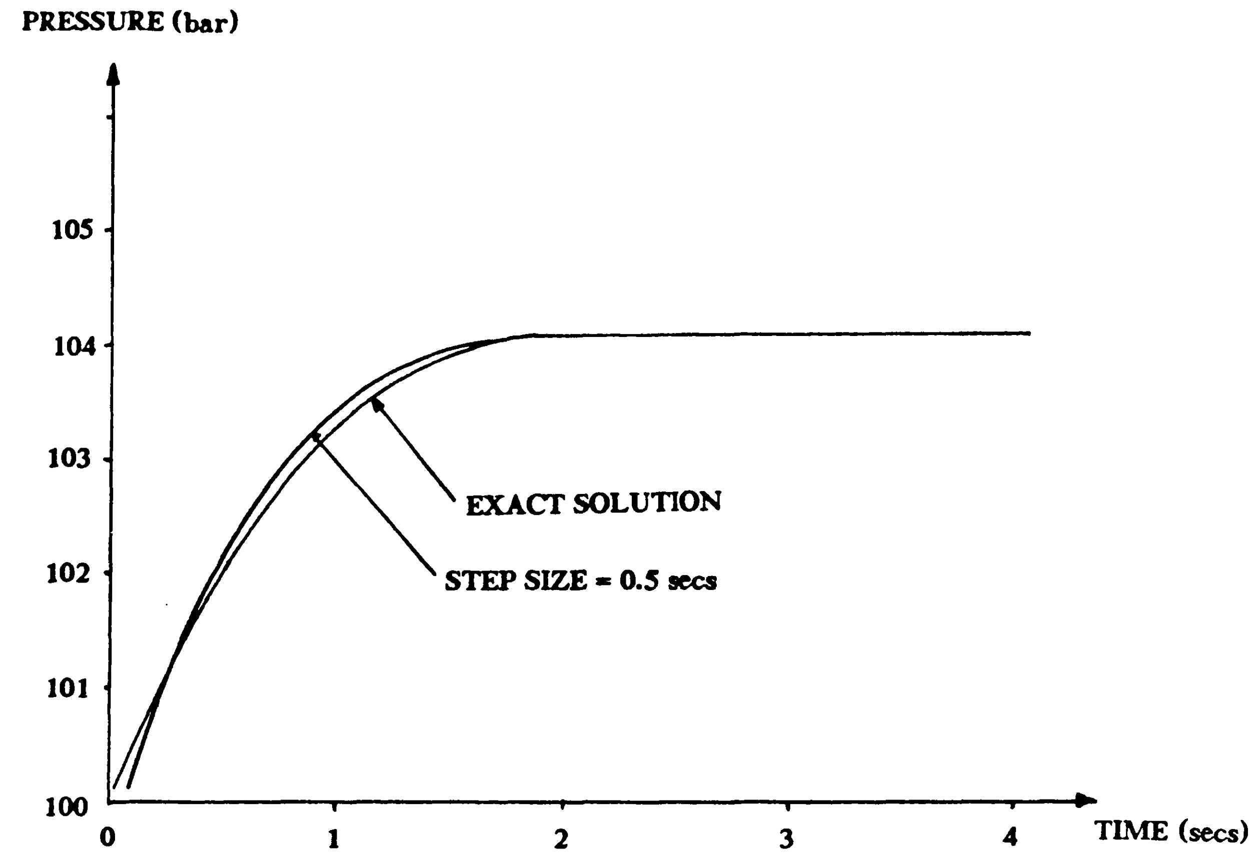


FIGURE 3.3 COMPARISON OF EXACT SOLUTION WITH RESULTS OBTAINED USING THE EXPLICIT METHOD FOR TEST PROBLEM 1



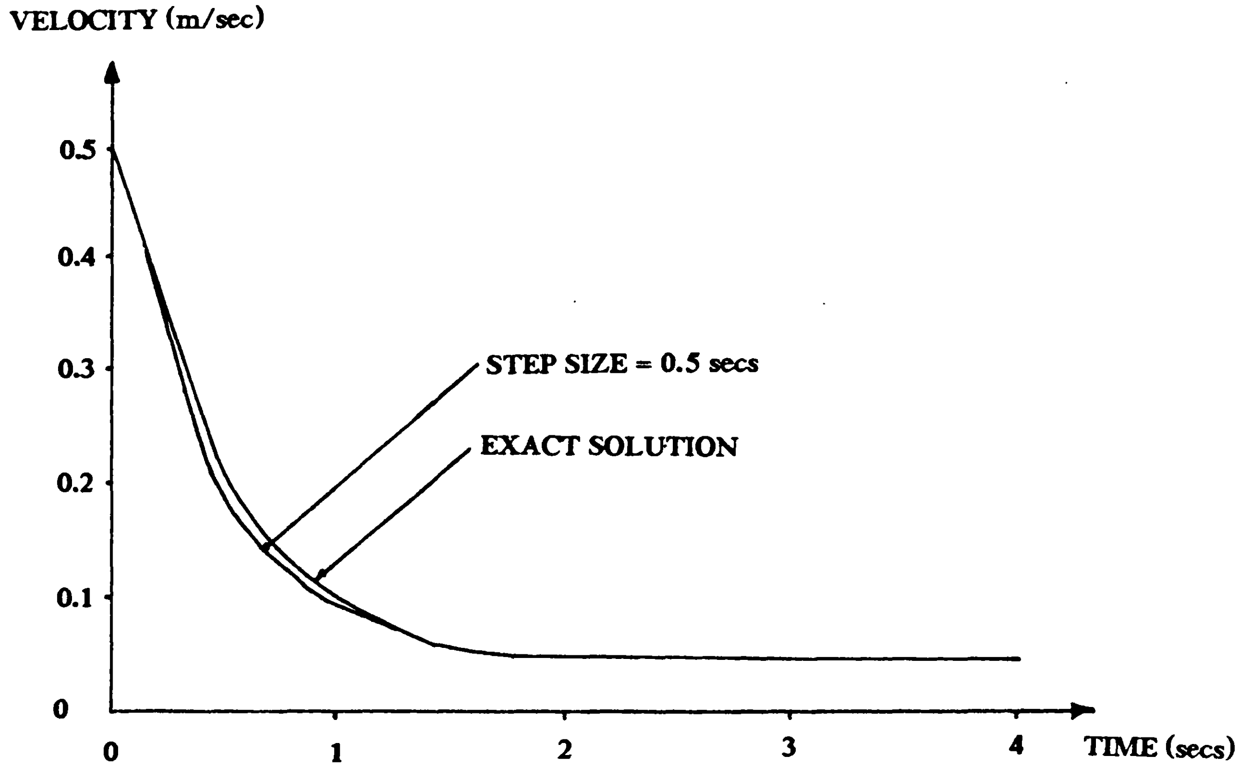


FIGURE 3.4 COMPARISON OF EXACT SOLUTION WITH RESULTS OBTAINED USING THE SECOND IMPLICIT METHOD FOR TEST PROBLEM 1

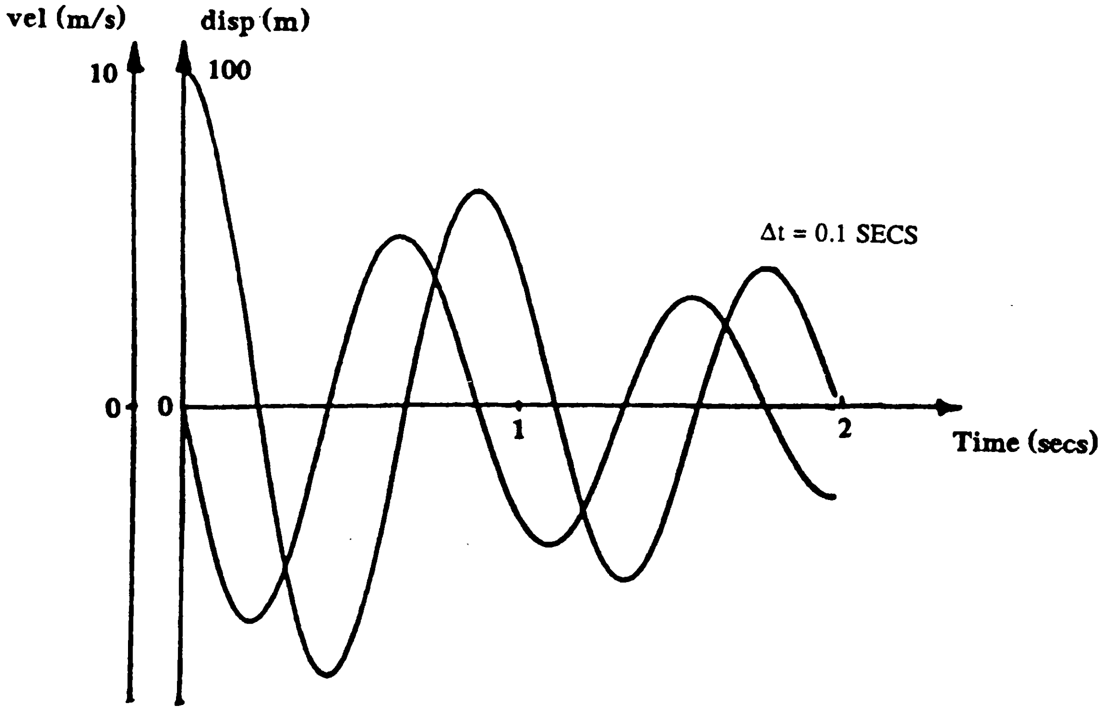


FIGURE 3.5 SYSTEM RESPONSE OF A SECOND ORDER EQUATION USING THE NEW METHOD

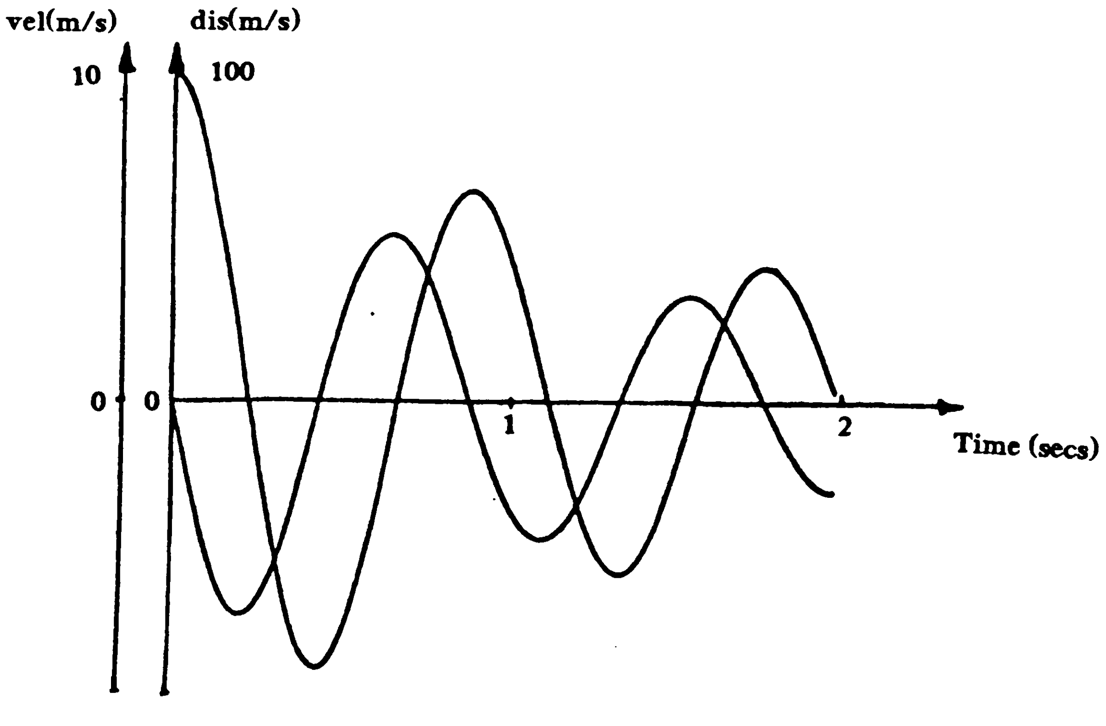


FIGURE 3.6 TRUE SOLUTION FOR THE SECOND ORDER EQUATION

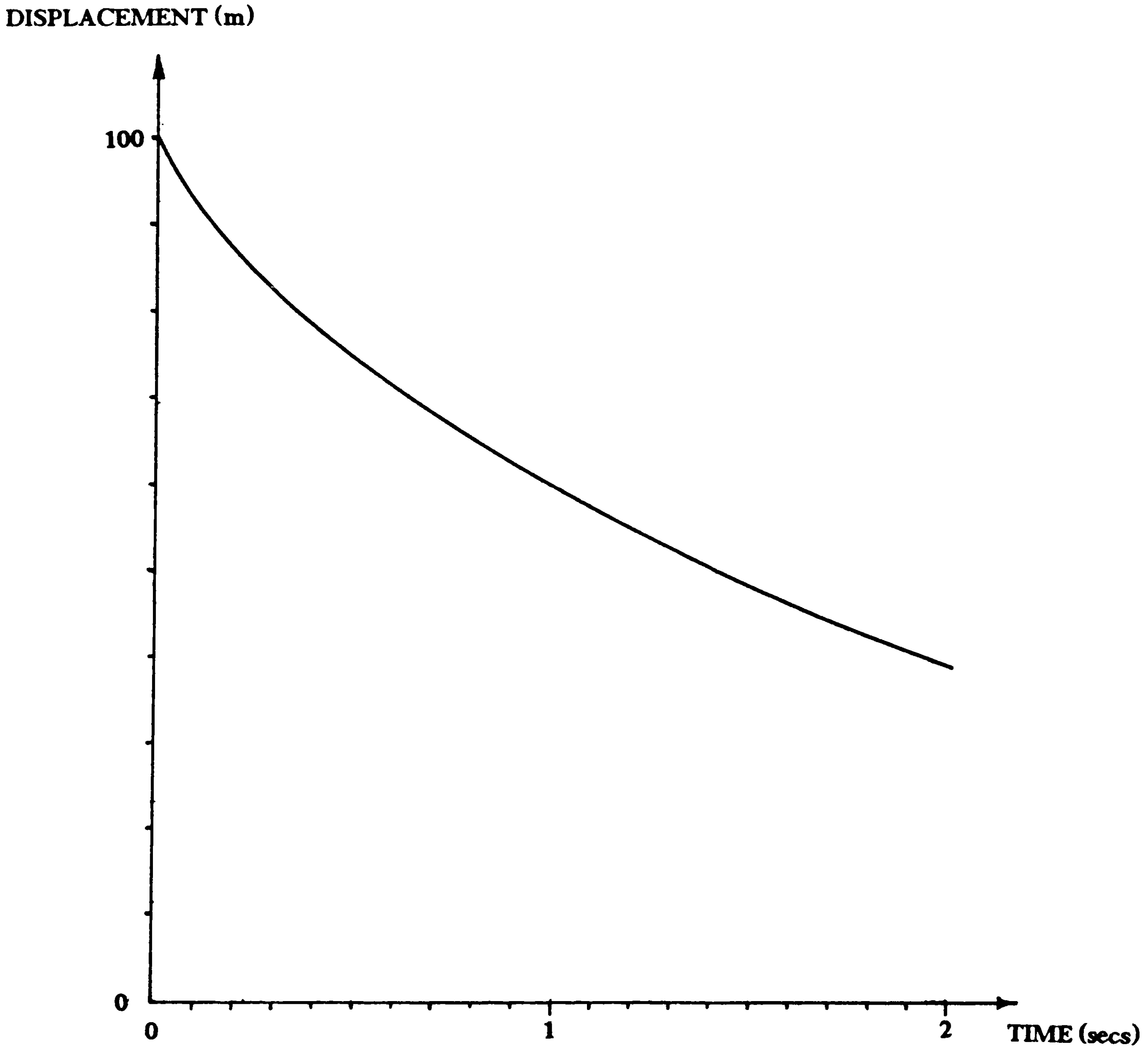
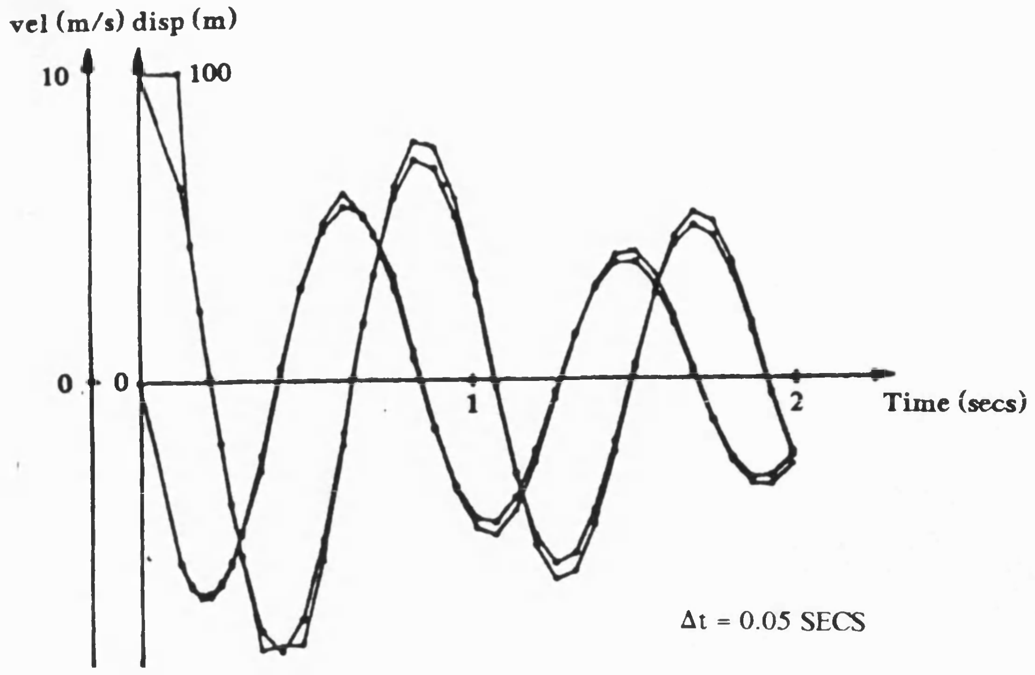


FIGURE 3.7 SOLUTION FOUND BY NEW METHOD FOR PROBLEM 2 CASE B



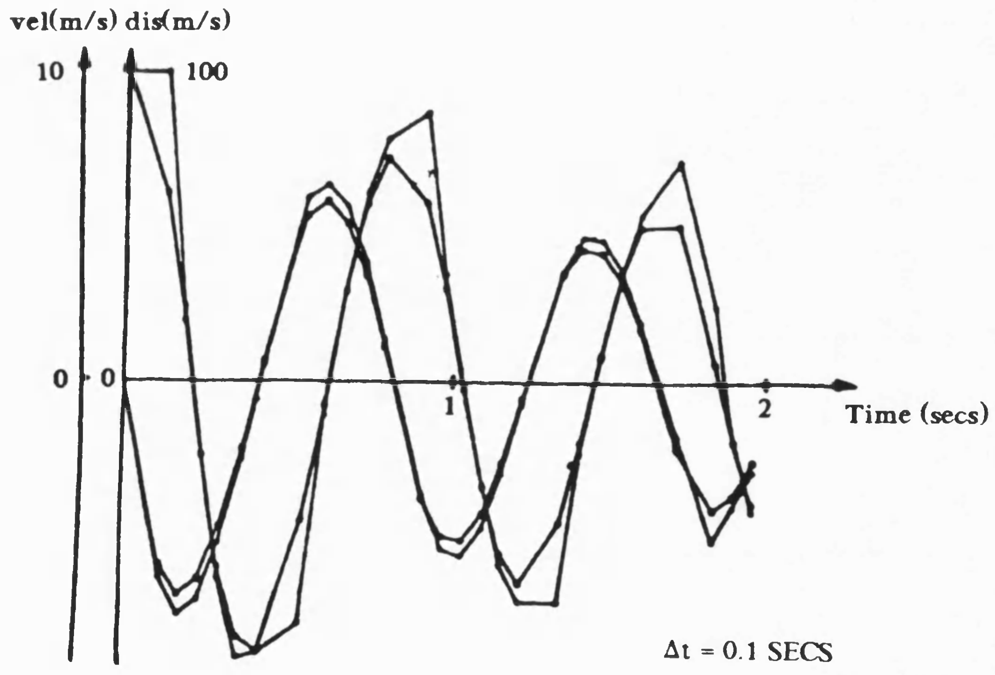


FIGURE 3.8 SOLUTION TO PROBLEM 2 FOUND BY LEUNG USING THE NEW METHOD

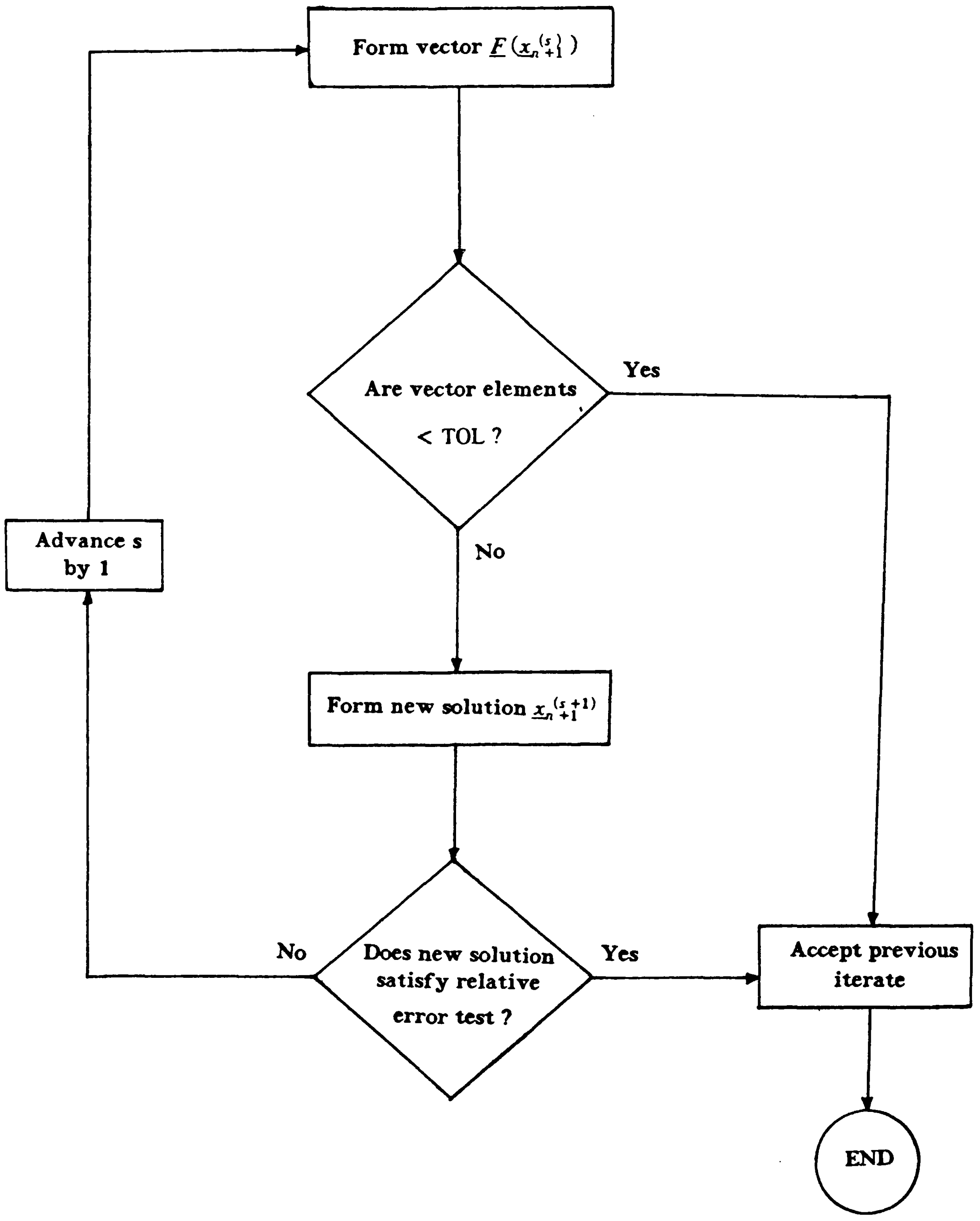

Form vector $\underline{F}(\underline{x}_{n+1}^{(s)})$
Are vector elements < TOL ?
Yes
No
Form new solution $\underline{x}_{n+1}^{(s+1)}$
Does new solution satisfy relative error test ?
No
Yes
Advance s by 1
Accept previous iterate
END

FIGURE 3.9 FLOW DIAGRAM ILLUSTRATING THE ACTION OF THE ITERATION SCHEME

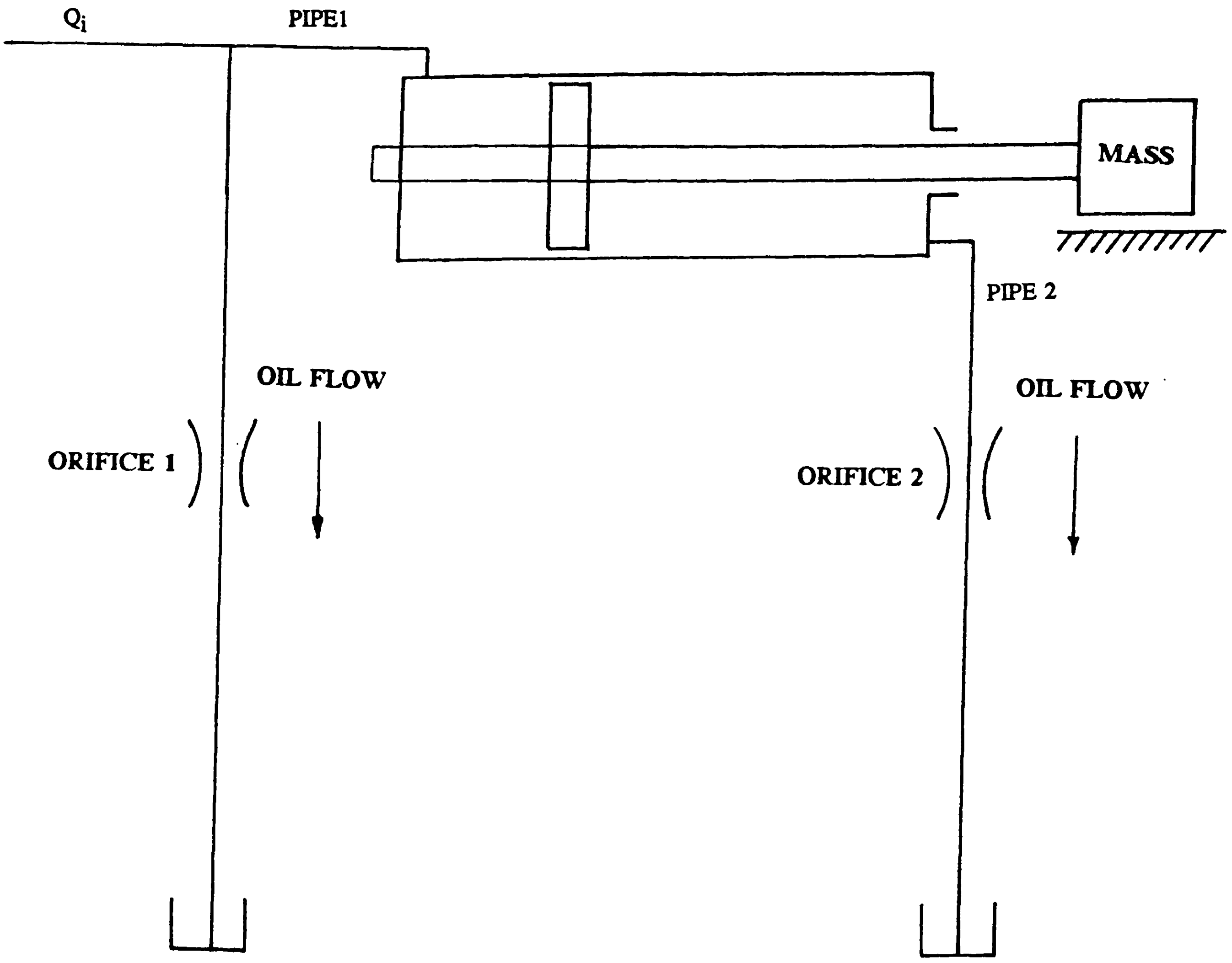


FIGURE 3.10 CIRCUIT DIAGRAM FOR TEST PROBLEM 3
- LINEAR ACTUATOR CIRCUIT

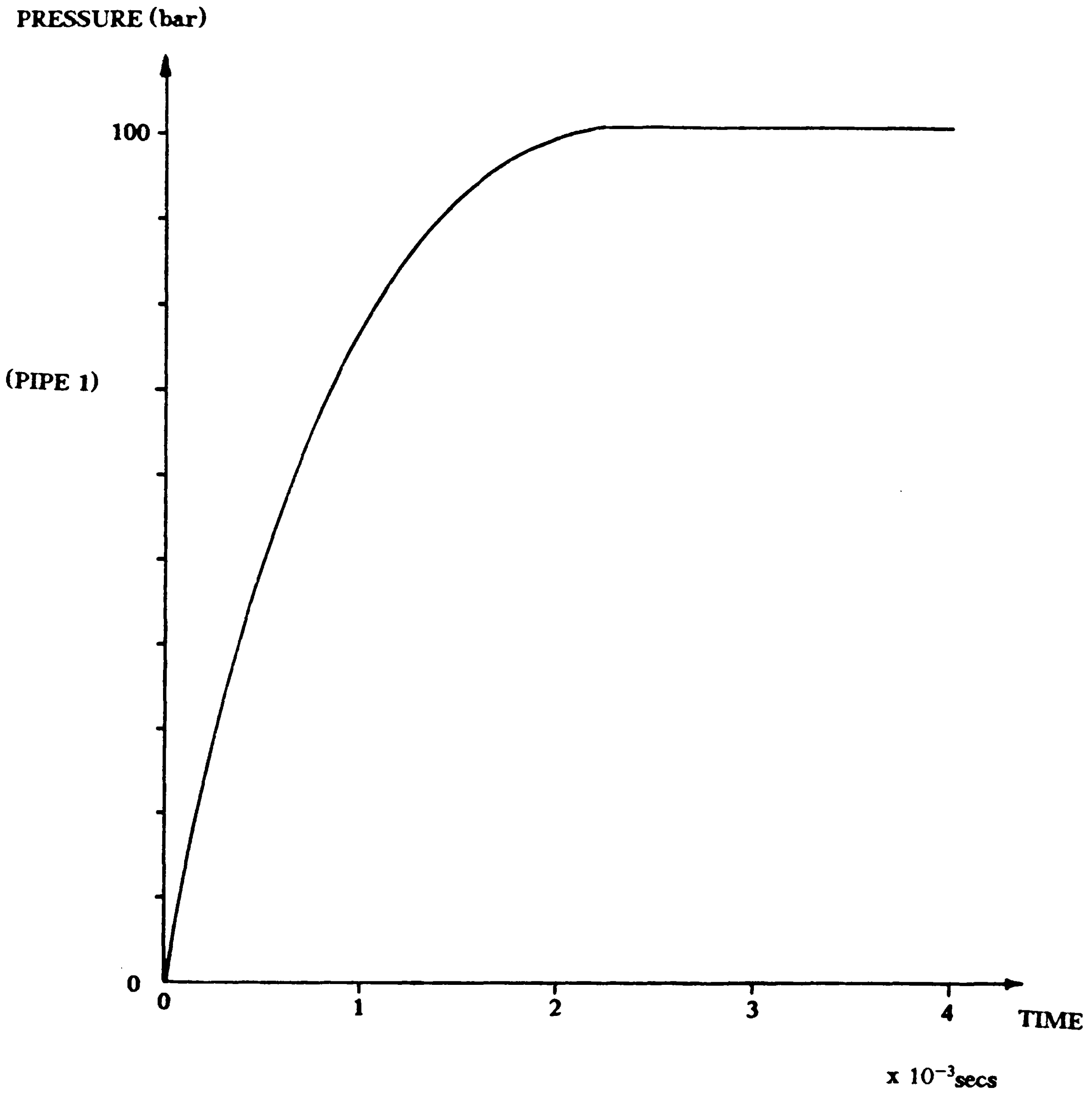


FIGURE 3.11 A TYPICAL RESPONSE OF THE THIRD ORDER ACTUATOR CIRCUIT FOR THE NON-OSCILLATORY PROBLEM

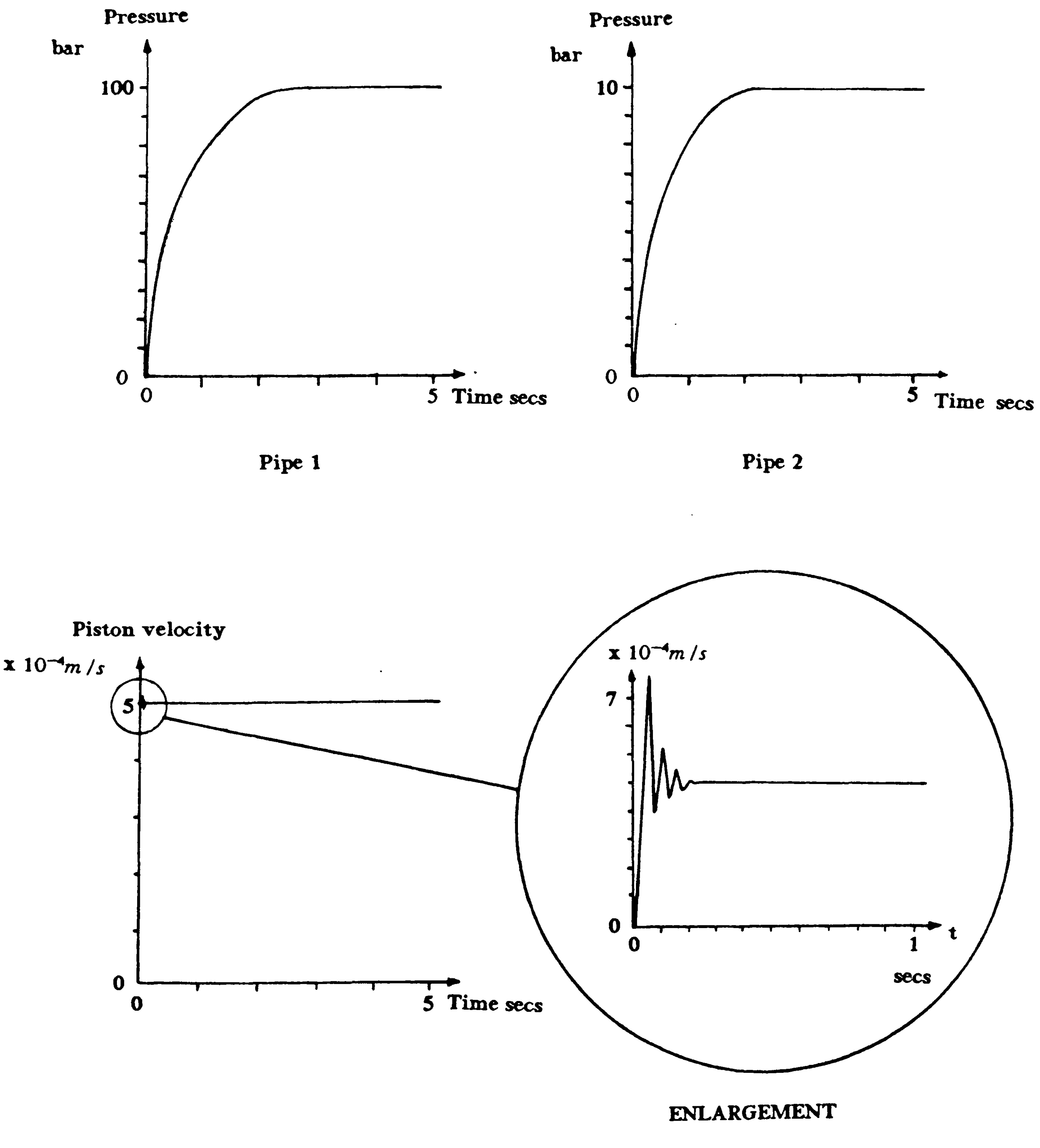


FIGURE 3.12 RESPONSES OF THE ACTUATOR CIRCUIT FOR THE OSCILLATORY PROBLEM

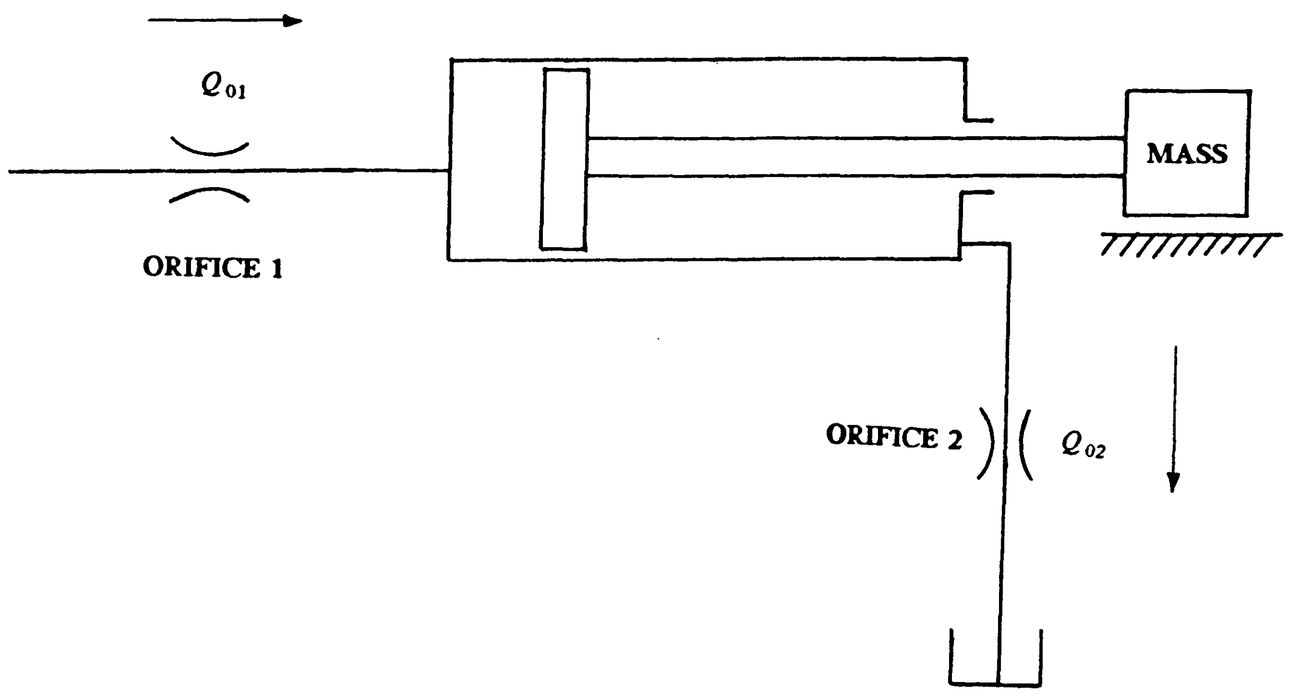
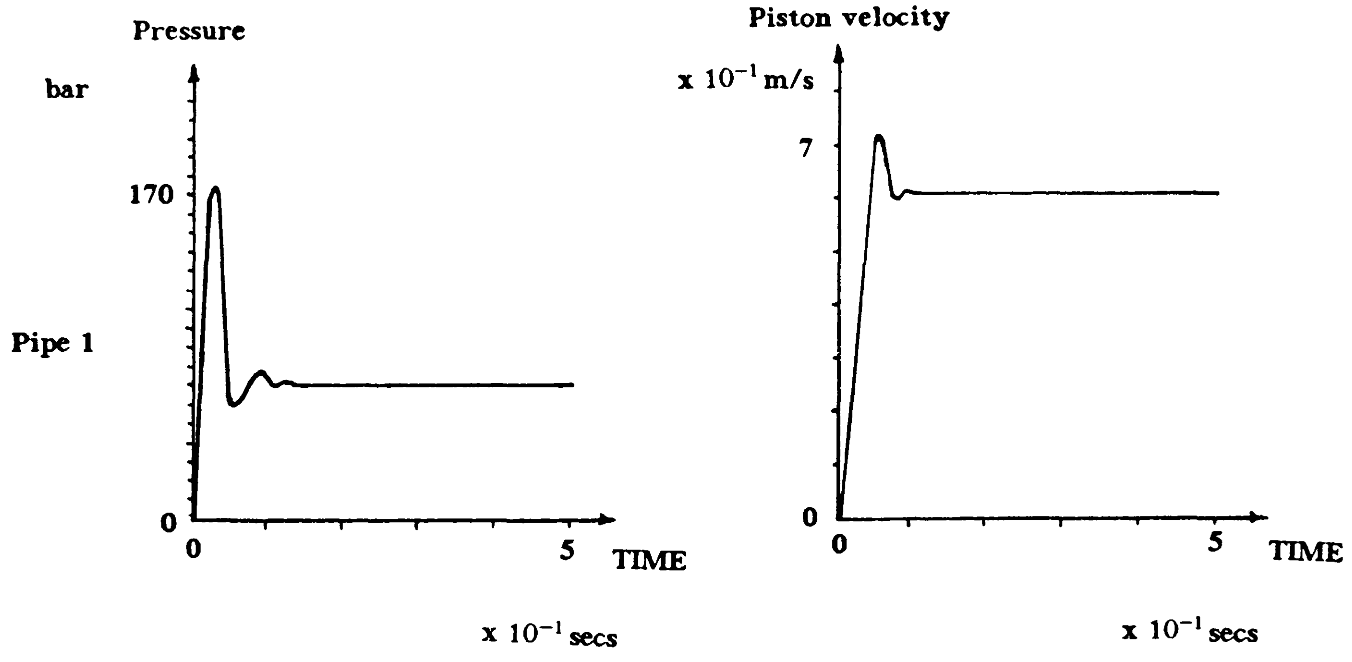


FIGURE 3.13 CIRCUIT DIAGRAM FOR PROBLEM 4 - NON-LINEAR PROBLEM



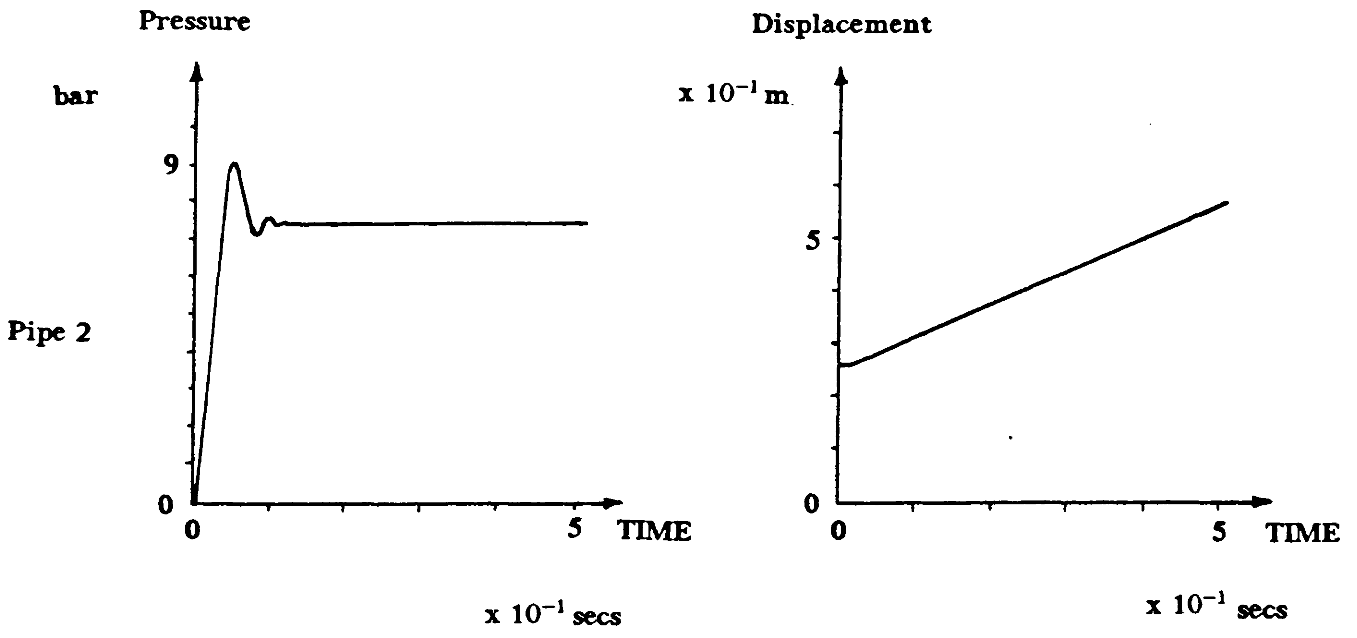


FIGURE 3.14 SOLUTION OF PROBLEM 4 USING NEW METHOD

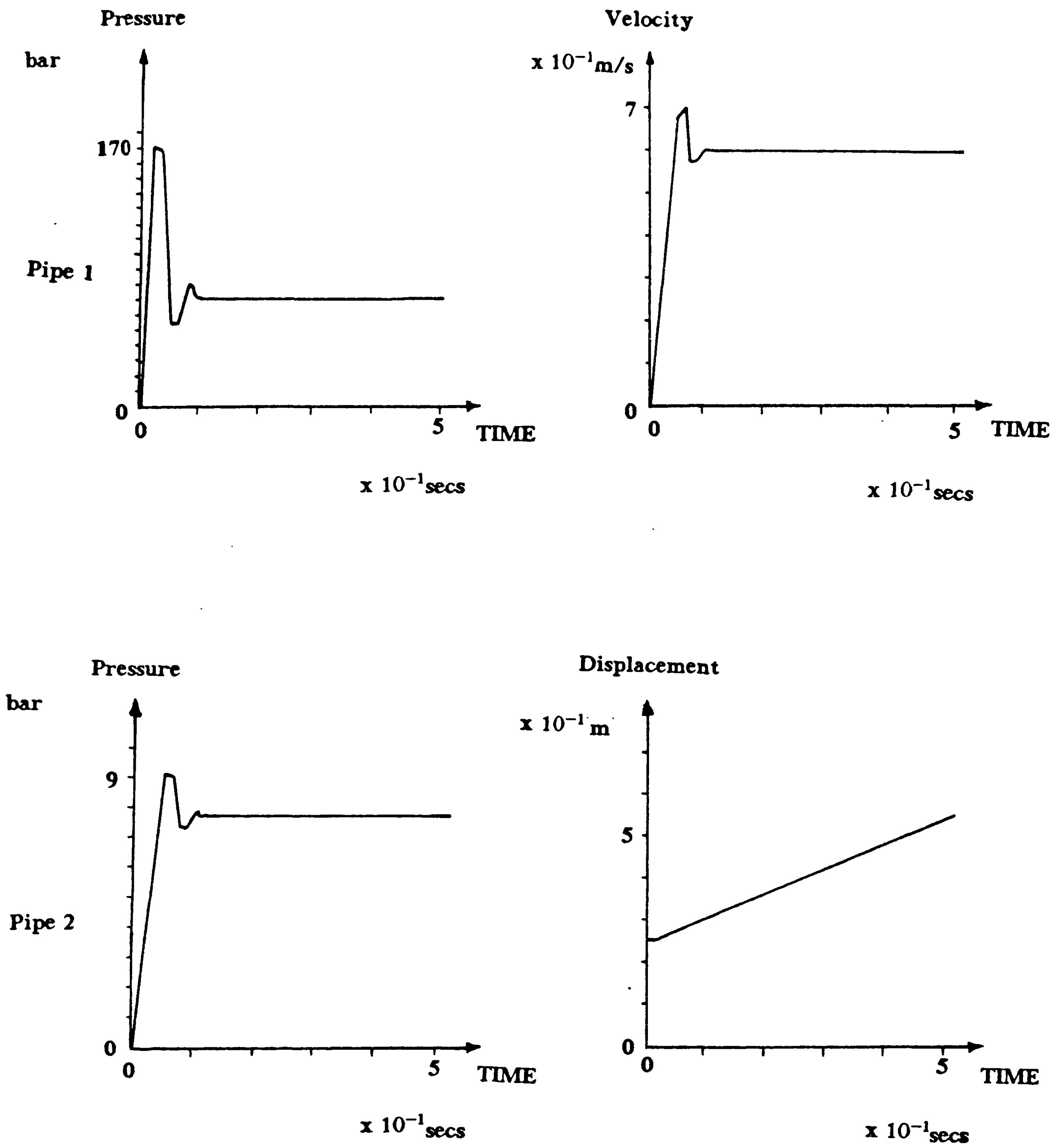


FIGURE 3.15 SOLUTION OF PROBLEM 4 USING GEAR'S METHOD

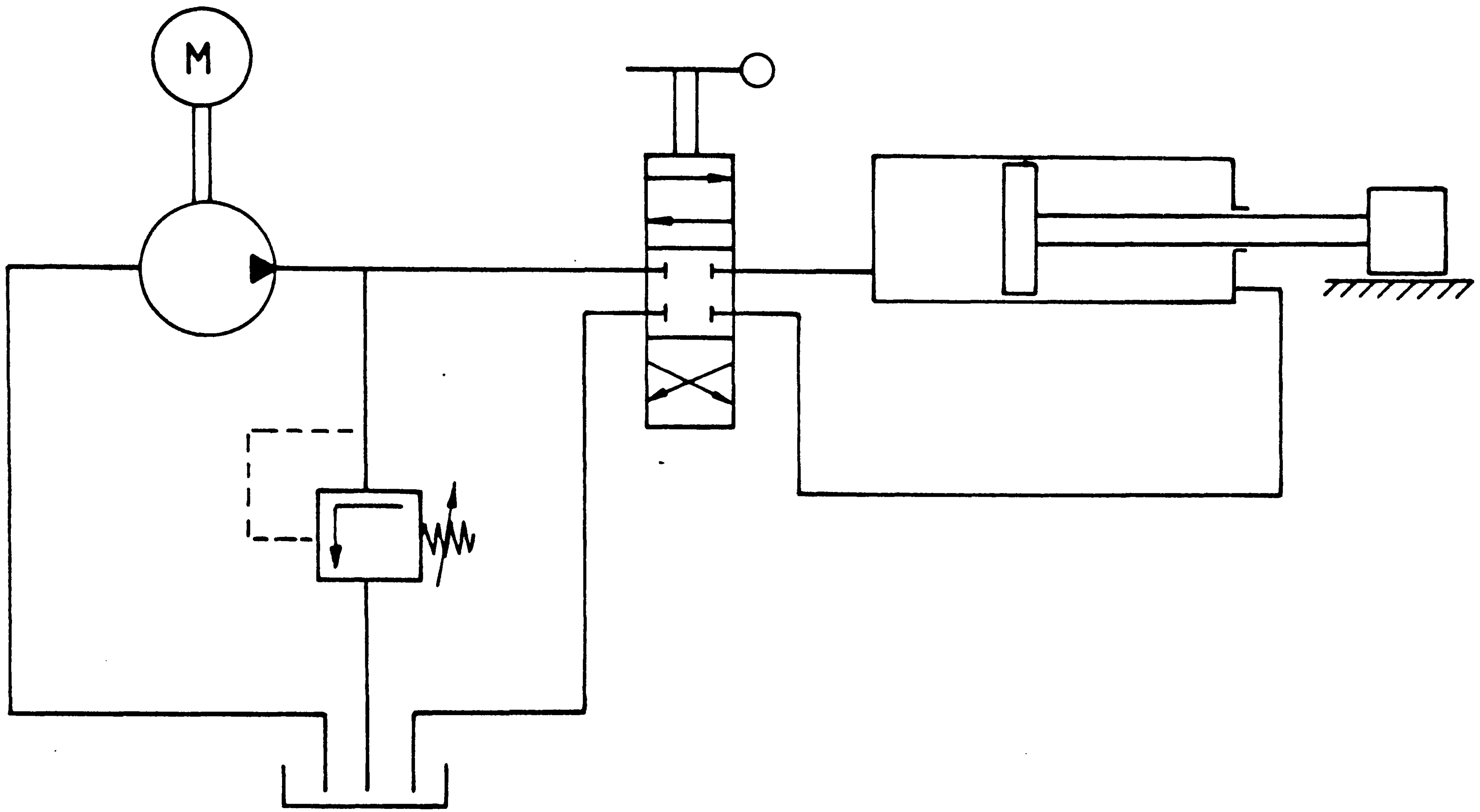


FIGURE 3.16 LINEAR ACTUATOR CIRCUIT FOR PROBLEM 5

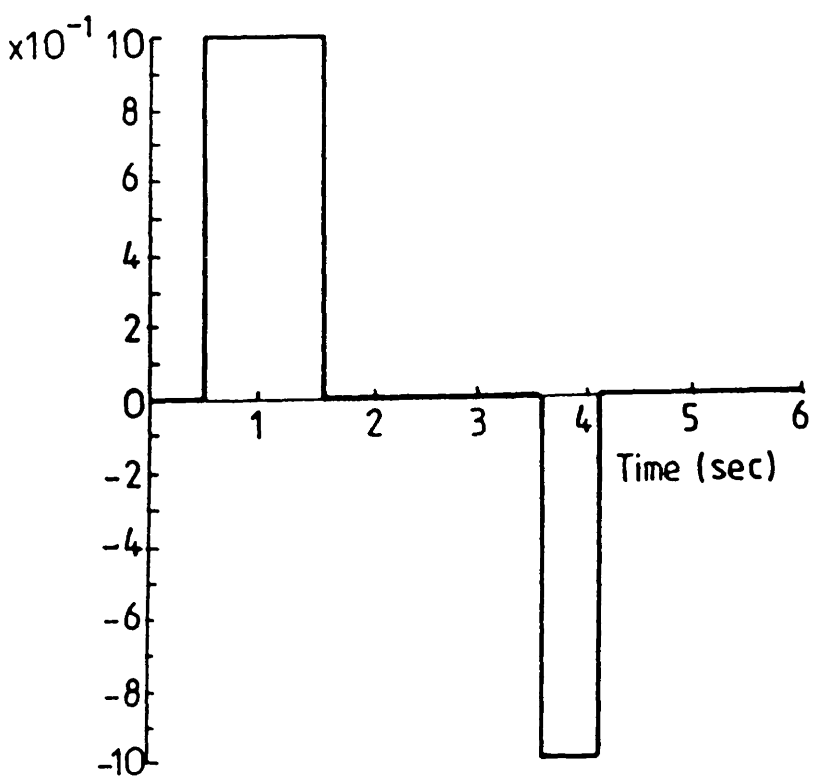


FIGURE 3.17 POSITION OF THE DIRECTIONAL CONTROL VALVE

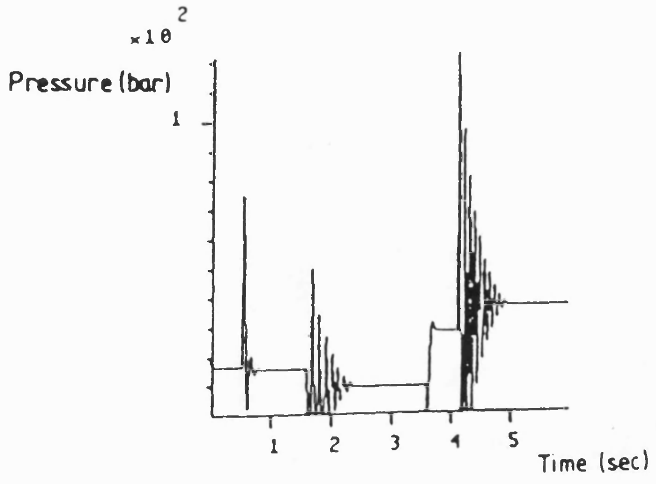


FIGURE 3.18 SIMULATED PISTON PRESSURE

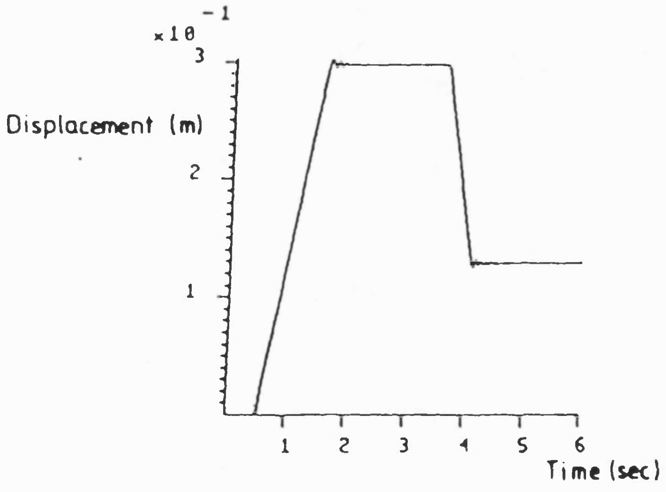


FIGURE 3.19 SIMULATED ACTUATOR DISPLACEMENT

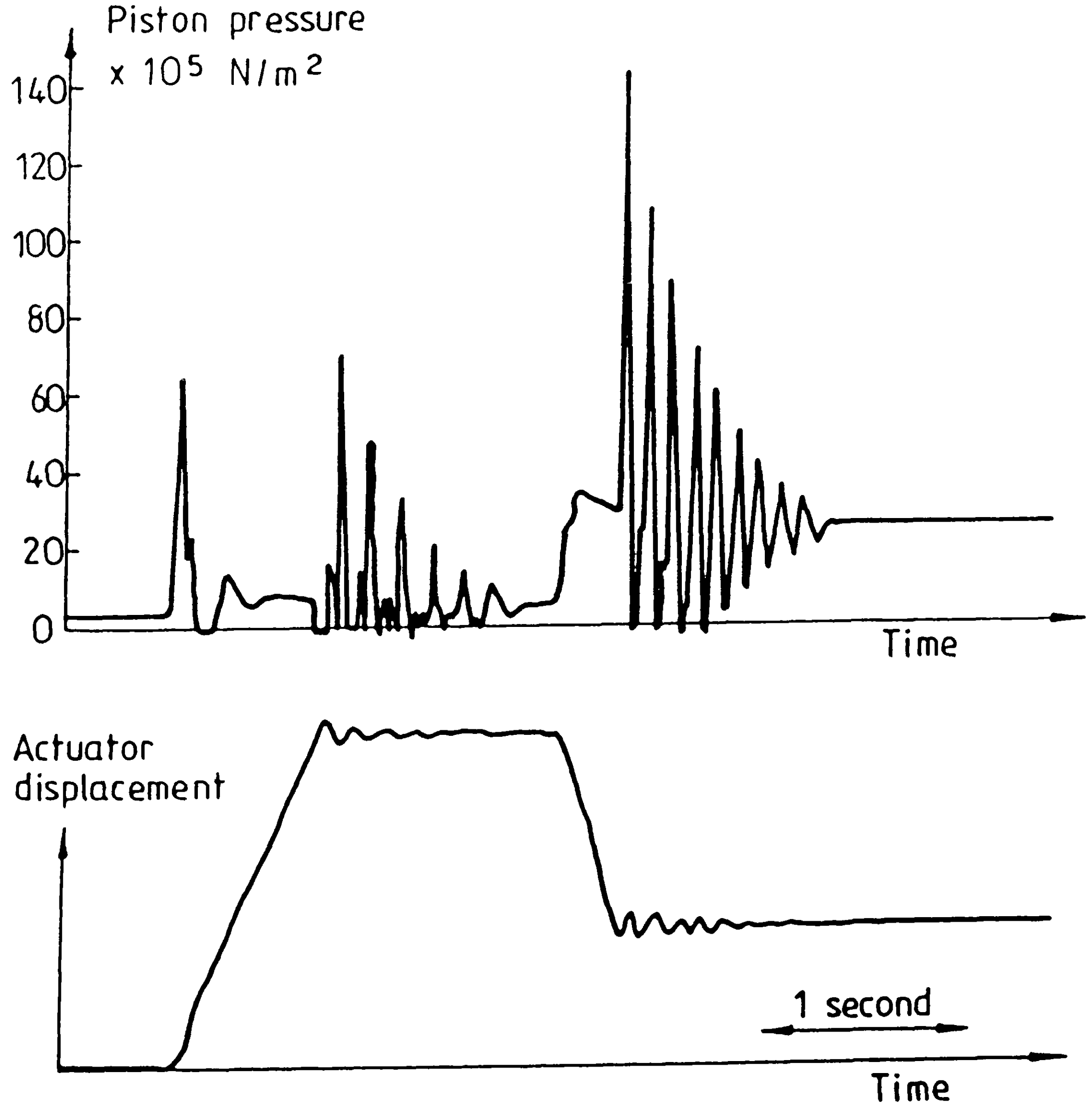


FIGURE 3.20 EXPERIMENTAL RESULTS FROM THE LABORATORY

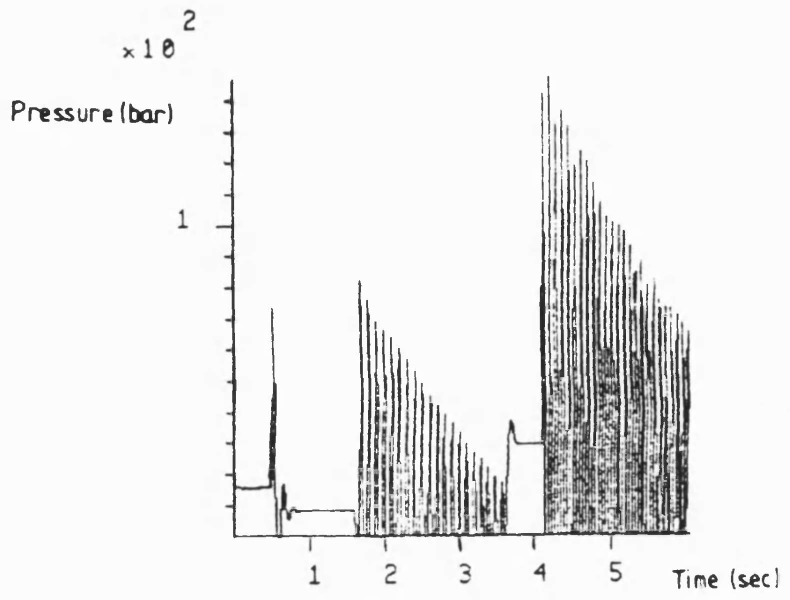


FIGURE 3.21 PISTON PRESSURE WITH CHANGED DATA FOR PROBLEM 5

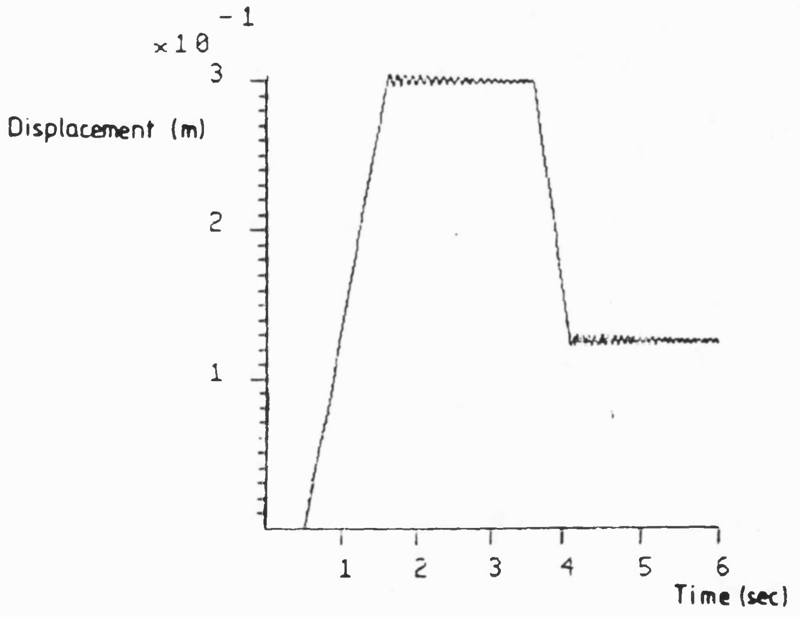


FIGURE 3.22 ACTUATOR DISPLACEMENT WITH CHANGED DATA

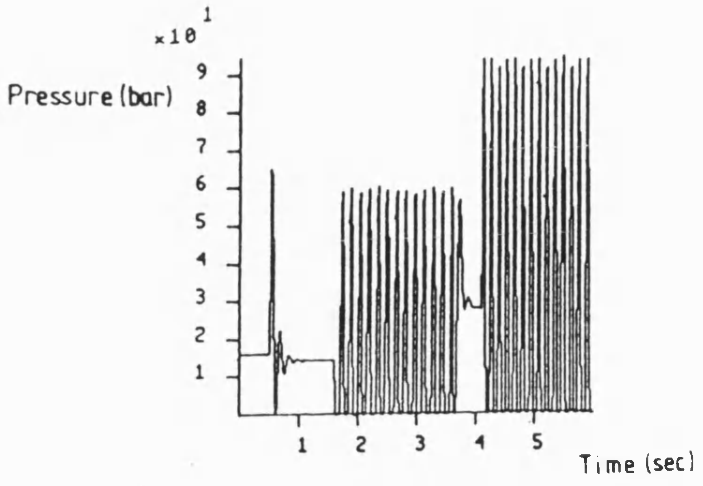


FIGURE 3.23 PISTON PRESSURE FOUND USING GEAR'S METHOD

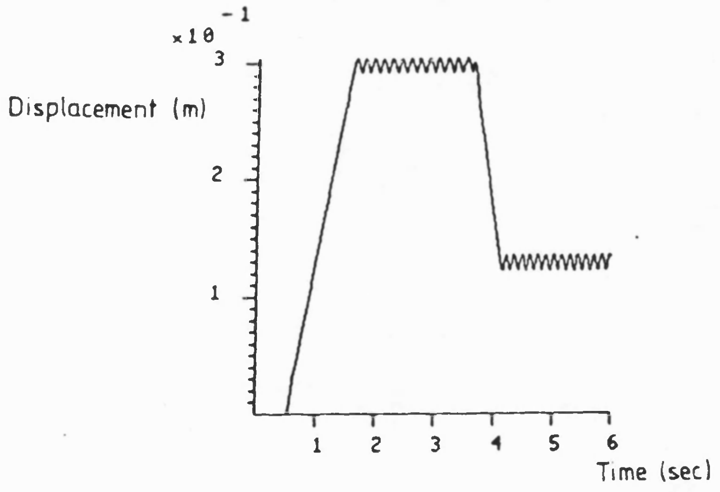


FIGURE 3.24 ACTUATOR DISPLACEMENT FOUND USING GEAR'S METHOD

# CHAPTER 4

**DETAILED CONTENTS** **Page**

# CHAPTER 4

# MATHEMATICAL ANALYSIS AND FURTHER TESTING OF THE NEW METHOD

## Introduction

**4.1** This chapter analyses in some depth the method that has been developed, and from this analysis a more robust, general purpose integration method is constructed. The chapter is split into two main areas; the first deals with the theoretical aspects of the analysis, and the second examines practical problems which are used to demonstrate the ideas introduced in the first half of the chapter.

**4.2** Firstly, the local error of the method, which is defined in section 2.33, is investigated. Then, a time step control for both the explicit method, and the predictor-corrector pair derived in chapter 3, is developed. Finally, the stability properties of the explicit method and the first implicit method, given by equation 3.7, are analysed, and the resultant stability regions are discussed. The results found from the stability analysis and the development of the time step control assist in deciding in which way the method is to be generalised so as to be suitable for solving non-specific sets of differential equations.

**4.3** After performing the theoretical analysis, problems which illustrate the ideas that have been introduced are studied. A selection of the problems previously met in the last chapter are solved again by the new method, this time with a time step control operating within the method. Systems which display the necessary properties required by the new method to ensure stability are studied, and this will demonstrate that the ideas met in the earlier part of the chapter have a practical application.

## The Order Of The Local Error Of The Explicit Method

**4.4** Referring to section 2.33, the local error of the method was found by comparing the method with a Taylor's series expansion made about the solution point. If the explicit method given by equation (3.6) is applied to the initial value problem

$$y' = a(t)y + b(t) \qquad y(\alpha) = \beta \tag{4.1}$$

where a and b are both time variant, then the numerical method is given by:

$$y_{n+1} = \frac{b_n}{a_n}(e^{a_n h} - 1) + y_n e^{a_n h}$$

Assuming that y(t) is a sufficiently smooth function, i.e. it is continuously differentiable several times, and applying the scheme; then assuming exact values at $t = t_n$, the method will satisfy

$$y_{n+1} = \frac{b(t_n)}{a(t_n)}(e^{a(t_n)h} - 1) + y(t_n)e^{a(t_n)h} \tag{4.2}$$

although the true solution will satisfy

$$y(t_{n+1}) = \frac{b(t_n)}{a(t_n)}(e^{a(t_n)h} - 1) + y(t_n)e^{a(t_n)h} + E_{n+1} \tag{4.3}$$

where $a(t_n)$, $b(t_n)$ and $y(t_n)$ denote the exact values at $t = t_n$, $y(t_{n+1})$ denotes the true solution at time $t = t_{n+1}$ and $y_{n+1}$ denotes the approximation to the solution at $t = t_{n+1}$. $E_{n+1}$ is the local error, the error occuring in one step of the scheme.

**4.5** Forming a Taylor's series expansion about $y(t_{n+1})$ gives:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2!}y''(t_n) + \frac{h^3}{3!}y'''(\xi) \quad t_n < \xi < t_{n+1} \tag{4.4}$$

Comparing equation (4.4) with (4.3) and subtracting will leave an expression for the local error, viz:

$$E_{n+1} = y(t_n) + hy'(t_n) + \frac{h^2}{2!}y''(t_n) + O(h^3) -$$
$$\frac{b(t_n)}{a(t_n)}(e^{a(t_n)h} - 1) - y(t_n)e^{a(t_n)h} \tag{4.5}$$

Now substituting equation (4.1) into the second term on the right hand side of equation (4.5) and expanding the exponential terms will give:

$$E_{n+1} = y(t_n) + h(a(t_n)y(t_n) + b(t_n)) + \frac{h^2}{2}y''(t_n) + O(h^3) +$$
$$\frac{b(t_n)}{a(t_n)}(1 - (1+a(t_n)h+\frac{(a(t_n)h)^2}{2}+\frac{(a(t_n)h)^3}{6}+ \cdots )) - y(t_n)e^{a(t_n)h} \tag{4.6}$$

which, when rearranged, will leave:

$$E_{n+1} = \frac{h^2}{2}y''(t_n) + O(h^3) - \frac{b(t_n)a(t_n)h^2}{2} -$$
$$\frac{b(t_n)a(t_n)^2h^3}{6} - \frac{y(t_n)a(t_n)^2h^2}{2} - \frac{y(t_n)a(t_n)^3h^3}{6} \tag{4.7}$$

i.e.

$$E_{n+1} = \frac{h^2}{2}(y''(t_n) - b(t_n)a(t_n) - y(t_n)a(t_n)^2) + O(h^3) \tag{4.8}$$

Again, using equation (4.1), this will reduce to:

$$E_{n+1} = \frac{h^2}{2}(y''(t_n) - a(t_n)y'(t_n)) + O(h^3) \tag{4.9}$$

This gives an estimation to the local error of the explicit method, and referring to section 2.37, then the method is first order accurate, i.e. the error expansion includes terms involving $h^2$.

## The Order Of The Local Error Of The First Implicit Method

**4.6** Using an analysis similar to that in section 4.4, then the local error of the implicit method given by equation 3.7 can be shown to be: [66]

$$E_{n+1} = \frac{h^2}{2}(a(t_n)y'(t_n) - y''(t_n)) + O(h^3) \tag{4.10}$$

and this again indicates a first order method.

## The Order Of The Local Error Of The Second Implicit Method

**4.7** The local error of the implicit method given by equation(3.11)can also be shown to be:

$$E_{n+1} = \frac{h^2}{2}(a(t_n)y'(t_n) - y''(t_n)) + O(h^3) \tag{4.11}$$

and so this method and the others are first order accurate. Consequently, referring to section 2.42, these three methods are both consistent and zero-stable, and so convergence of the numerical scheme will follow as $h \rightarrow 0$.

## Time Step Control For The New Method

**4.8** The most difficult problem in the application of an integration method is that of choosing an appropriate value for the steplength. A bound for the global error does not, in general, provide an adequate basis for choosing the step size, h. Instead, it is necessary to turn to the idea of finding an interval for h which ensures that the global error does not grow in a certain sense. It is essential to choose h such that the local error is acceptably small, but the application of a bound for the local error is hindered by the practical difficulty of finding a bound for $|y^{(p)}(t)|$ and $|y^{(p+1)}(t)|$, where in the case of the new method, $p = 1$, the order of accuracy of the method.

**4.9** However, when using a predictor-corrector method in an appropriate mode, it is possible to avoid estimating higher derivatives by using a combination of the solution given by the predictor and the corrector. The principal local error term of the combined scheme, which is accepted as as an approximation of the true local error, can be estimated using this strategy. Consequently, the steplength, h, will be chosen such that the estimated principal local error remains at each step less than a pre-assigned tolerance. Also, h must be chosen to ensure that the iteration scheme, used in conjunction with the corrector, will converge.

**4.10 Richardson's extrapolation.** With the application of the explicit method alone, it is not possible to form a readily computed estimate of the local error similar to the one that can be obtained with a predictor-corrector pair. However, there are alternative ways of estimating the error, and the one used with the explicit method arises from an application of the deferred approach to the limit, alternatively called Richardson's extrapolation [67]. Under the usual localising assumption that no previous errors have been made, the local error made in advancing the scheme from $t = t_n$ to $t = t_{n+1}$ can be written as:

$$y(t_{n+1}) - y_{n+1} = \psi(t_n, y(t_n))h^2 + O(h^3) \tag{4.12}$$

Now computing $y^*_{n+1}$, a second approximation to $y(t_{n+1})$, obtained by applying the same method at $t_{n-1}$ with steplength 2h, then using the same localising assumption it follows that:

$$y(t_{n+1}) - y^*_{n+1} = \psi(t_{n-1}, y(t_{n-1}))(2h)^2 + O(h^3) \tag{4.13}$$

which becomes:

$$y(t_{n+1}) - y^*_{n+1} = \psi(t_n, y(t_n))(2h)^2 + O(h^3) \tag{4.14}$$

on expanding $\psi(t_{n-1}, y(t_{n-1}))$ about $(t_n, y(t_n))$. Subtracting equation (4.12) from (4.14) will give:

$$y_{n+1} - y^{*}_{n+1} = (2^2 - 1)\psi(t_n, y(t_n))h^2 + O(h^3) \tag{4.15}$$

and so the principal local error term, which is taken as an estimate for the local error, may be written as:

$$\psi(t_n, y(t_n))h^2 = \frac{y_{n+1} - y^{*}_{n+1}}{(2^2 - 1)} \tag{4.16}$$

Hence, to apply Richardson's extrapolation, the solution is computed over two successive steps using steplength h, and then recomputed over the double step using steplength 2h. The difference between the values for the $y(t_{n+1})$ so obtained, when divided by three, is then an estimate to the local error. This estimate is usually quite adequate for step control purposes, but it involves a considerable increase in computational effort, and thus may not be as beneficial as would be hoped.

**4.11** Since an approximate value for the principal local error term can now be evaluated, a strategy can easily be adopted to control the steplength during the integration of a problem. This strategy is explained in section 4.14. However, because of the computational inefficiency which this method of estimating the local error leads to, since excessive work is required at each step to form an estimate, an efficient time step control mechanism has only been implemented with the predictor-corrector pair. As is demonstrated in the next section, with this pair, there is an economical and simple way of estimating the local error at each step, without repeating the integration at any point.

**4.12 Local error estimation for the predictor-corrector method.** Since the principal local error term for the explicit method is given by:

$$\frac{h^2}{2}(y''(t_n) - a(t_n)y'(t_n)) = h^2L_1$$

where $L_1$ is the local error function

and the principal local error term for either of the implicit methods is given by:

$$\frac{h^2}{2}(a(t_n)y'(t_n) - y''(t_n)) = -h^2L_1$$

then the explicit and either of the implicit methods make an ideal predictor-corrector pair. Not only is the order of accuracy of each method the same, which, as described in section 2.68, ensures that the local error of the combined pair is the same as that of the corrector alone; but it is easy to devise a strategy for estimating the local error of the corrector.

**4.13** In taking one step of the explicit and the implicit scheme, the relationships between the true value of the solution, the estimated values of the solution, and the local error are given by:

$$y(t_{n+1}) = y_{n+1}^{(p)} + h^2L_1 \tag{4.17}$$

and

$$y(t_{n+1}) = y_{n+1}^{(c)} - h^2L_1 \tag{4.18}$$

where (p) denotes the predicted value and (c) denotes the corrected value.

Subtracting equation (4.18) from (4.17) and rearranging will give an estimate to the local error, namely

$$\frac{|y_{n+1}^{(p)} - y_{n+1}^{(c)}|}{2} = |h^2L_1| \tag{4.19}$$

**4.14 Time step control for the predictor-corrector method.** Equation (4.19) will give an estimation of the local error at the first correction, and so it is possible to control the step size using this relationship. To do this the following strategy, which can also be applied to the explicit method, is employed.

To find an h such that

$$|h^2 L_1| = TOL \tag{4.20}$$

where TOL is a pre-set tolerance

then, defining $h^*$ as the present time step, with

$$|h^{*2} L_1| = ERR \tag{4.21}$$

where ERR is the error made in one step

and dividing equation (4.20) by (4.21) leaves

$$\frac{|h^2 L_1|}{|h^{*2} L_1|} = \frac{TOL}{ERR} \tag{4.22}$$

which will give the required h, with

$$h^2 = \frac{h^{*2} TOL}{ERR}$$

and hence

$$h = h^* \left( \frac{TOL}{ERR} \right)^{\frac{1}{2}} \tag{4.23}$$

The strategy used for controlling the time step, with the appropriate coding, can be found in Appendix B. The examples solved using the time step control can be found in sections 4.35 and 4.37.

## Stability Analysis Of The First Order Method

**4.15** The stability properties of the method are very important as stability is the determining factor in deciding whether a method is suitable for solving stiff or oscillatory sets of differential equations. When applying the method to the scalar test equation described in section 2.45, which is:

$$y' = \lambda y \quad y(0) = 1$$

then using the scheme described by equation (3.6), viz:

$$y_{n+1} = \frac{b_n}{a_n}(e^{a_n h} - 1) + y_n e^{a_n h}$$

will give:

$$y_{n+1} = y_n e^{\lambda h} \quad n = 0, 1, 2, \cdots \tag{4.24}$$

since $b_n = 0$ for all n, and $a_n = \lambda$ for all n

If $y_0$ is perturbed to $y_0 + \epsilon = z_0$, then the sequence will become $z_0, z_1, z_2, \cdots$ from the modified difference equation

$$z_{n+1} = z_n e^{\lambda h} \tag{4.25}$$

Subtracting equation (4.24) from (4.25) will give:

$$z_{n+1} - y_{n+1} = e^{\lambda h}(z_n - y_n) \quad n = 0, 1, 2, \cdots \tag{4.26}$$

and recursively applied, this leads to:

$$z_n - y_n = e^{\lambda nh}(z_0 - y_0) \quad n = 0, 1, 2, \cdots \tag{4.27}$$

The requirement is that the effect of the the initial perturbation will die away with increasing n; this will be guaranteed if $h\lambda < 0$. Hence, for an inherently stable scalar equation, i.e. where $\lambda < 0$, then the explicit method is absolutely stable for all values of h. with h positive. This is a good result for an explicit method. and it is easy to see that the result can be extended to cover the implicit methods given by equation (3.7) and (3.11).

**4.16 System stability analysis.** Referring to section 2.46, then to apply a similar stability analysis to a set of equations, it is necessary to consider the test system of m equations, given by:

$$\underline{y}' = A\underline{y} \quad \underline{y}(0) = \underline{\beta}$$

where the constant matrix A is assumed to have m linearly independent eigenvectors, and so may be written as:

$$P_m^{-1}AP_m = \Lambda$$

where:

$P_m$ is the modal (mxm) matrix consisting of eigenvectors of A
$\Lambda$ is the diagonal (mxm) matrix with the eigenvalues of A on the diagonal

However, this path cannot be pursued to obtain stability criteria for the new method since the method will not be applied to the formula given by the equation above,. but to the rewritten form given by:

$$\underline{y}' = D\underline{y} + (A - D)\underline{y} \qquad (4.28)$$

where D is the (mxm) matrix made up of the diagonal elements of A

**4.17** All numerical methods will seek to estimate the exponential matrix $e^{Ah}$ when used to solve the matrix equation given by equation (2.51) [68], since this will give the form of the true solution i.e. a method will try to provide

$$\underline{y}_{n+1} = e^{Ah}\underline{y}(t_n) \qquad (4.29)$$

However, there is no easy way of finding the exponential of a matrix, since the series expansion given by:

$$e^{Ah} = 1 + Ah + \frac{(Ah)^2}{2!} + \frac{(Ah)^3}{3!} + \cdots \tag{4.30}$$

shows that high powers of the matrix Ah are required, which would be impractical to evaluate, and difficult to approximate, particularly for large order matrices. Consequently, since the new method has been derived from a scalar ordinary differential equation, it effectively requires the decoupling of the systems to which it is applied, when used to solve sets of differential equations. Hence, to investigate its stability properties when applied to a set of equations, a different approach is required.

**4.18 Stability of the explicit method.** The approach taken here in examining the stability properties is to initially consider the case of the explicit method when applied to a mth order system of differential equations. This is then used as the base from which to establish the general stability properties, both for the explicit and the implicit method. Consider an mth order system of differential equations of the form

$$\begin{bmatrix} y'_1 \\ y'_2 \\ \cdot \\ \cdot \\ \cdot \\ y'_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdot & \cdot & \cdot & a_{1m} \\ a_{21} & a_{22} & a_{23} & \cdot & \cdot & \cdot & a_{2m} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{m1} & a_{m2} & a_{m3} & \cdot & \cdot & \cdot & a_{mm} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_m \end{bmatrix} \tag{4.31}$$

with initial conditions given by $\underline{y}(0) = \underline{\beta}$ where the coefficient "a"'s and "b"'s are time variant, but are assumed to be constant over one time step.

**4.19** Applying the explicit method given in equation (3.6) yields

$$y_{1_{n+1}} = \frac{b_{1_n} + a_{12_n}y_{2_n} + a_{13_n}y_{3_n} + \cdots + a_{1m_n}y_{m_n}}{a_{11_n}}(e^{a_{11_n}h} - 1) + y_{1_n}e^{a_{11_n}h}$$

$$y_{2_{n+1}} = \frac{b_{2_n} + a_{21_n}y_{1_n} + a_{23_n}y_{3_n} + \cdots + a_{2m_n}y_{m_n}}{a_{22_n}}(e^{a_{22_n}h} - 1) + y_{2_n}e^{a_{22_n}h}$$

$$\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}$$

$$y_{m_{n+1}} = \frac{b_{m_n} + a_{m1_n}y_{1_n} + a_{m2_n}y_{2_n} + \cdots + a_{mm-1_n}y_{m-1_n}}{a_{mm_n}}(e^{a_{mm_n}h} - 1) + y_{m_n}e^{a_{mm_n}h}$$

A similar process to that discussed in section 2.47 may now be used. To examine the way in which errors will propogate in the system of equations given above, it is necessary to introduce an error term into each of the equations, and then investigate the resultant behaviour of the solution as time advances.

4.20 Defining

$$z_i(0) = y_i(0) + \epsilon_i \quad i = 1, 2, \ldots\ldots, m \tag{4.32}$$

where $\epsilon_i$ is the error introduced into the solution

then the system of equations defined in the last section will become:

$$z_{1_{n+1}} = \frac{b_{1_n} + a_{12_n}z_{2_n} + a_{13_n}z_{3_n} + \cdots + a_{1m_n}z_{m_n}}{a_{11_n}}(e^{a_{11_n}h} - 1) + z_{1_n}e^{a_{11_n}h}$$

$$z_{2_{n+1}} = \frac{b_{2_n} + a_{21_n}z_{1_n} + a_{23_n}z_{3_n} + \cdots + a_{2m_n}z_{m_n}}{a_{22_n}}(e^{a_{22_n}h} - 1) + z_{2_n}e^{a_{22_n}h}$$

$$\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}$$

$$z_{m_{n+1}} = \frac{b_{m_n} + a_{m1_n}z_{1_n} + a_{m2_n}z_{2_n} + \cdots + a_{mm-1_n}z_{m-1_n}}{a_{mm_n}}(e^{a_{mm_n}h} - 1) + z_{m_n}e^{a_{mm_n}h}$$

Subtracting this set of equations from those in section 4.19 will leave expressions that describe the propagation of the error terms, viz:

$$\epsilon_{1_{n+1}} = \frac{a_{12_n}\epsilon_{2_n} + a_{13_n}\epsilon_{3_n} + \cdots + a_{1m_n}\epsilon_{m_n}}{a_{11_n}}(e^{a_{11_n}h} - 1) + \epsilon_{1_n}e^{a_{11_n}h}$$

$$\epsilon_{2_{n+1}} = \frac{a_{21_n}\epsilon_{1_n} + a_{23_n}\epsilon_{3_n} + \cdots + a_{2m_n}\epsilon_{m_n}}{a_{22_n}}(e^{a_{22_n}h} - 1) + \epsilon_{2_n}e^{a_{22_n}h}$$

$$\begin{array}{ccc} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{array}$$

$$\epsilon_{m_{n+1}} = \frac{a_{m1_n}\epsilon_{1_n} + a_{m2_n}\epsilon_{2_n} + \cdots + a_{mm-1_n}\epsilon_{m-1_n}}{a_{mm_n}}(e^{a_{mm_n}h} - 1) + \epsilon_{m_n}e^{a_{mm_n}h}$$

Expressing this last set of equations in matrix form will give:

$$\underline{\epsilon}_{n+1} = M_e\underline{\epsilon}_n \tag{4.33}$$

where:

$$M_e = \begin{vmatrix} e^{a_{11_n}h} & (e^{a_{11_n}h}-1)\frac{a_{12_n}}{a_{11_n}} & \cdots & (e^{a_{11_n}h}-1)\frac{a_{1m_n}}{a_{11_n}} \\ (e^{a_{22_n}h}-1)\frac{a_{21_n}}{a_{22_n}} & e^{a_{22_n}h} & \cdots & (e^{a_{22_n}h}-1)\frac{a_{2m_n}}{a_{22_n}} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ (e^{a_{mm_n}h}-1)\frac{a_{m1_n}}{a_{mm_n}} & (e^{a_{mm_n}h}-1)\frac{a_{m2_n}}{a_{mm_n}} & \cdots & e^{a_{mm_n}h} \end{vmatrix}$$

A stable solution to the problem will be ensured if the eigenvalues of the matrix $M_e$ all have modulus less than 1 [64]. The problem lies in determining these eigenvalues, and in relating the matrix $M_e$ to the original system matrix.

**4.21 Diagonally dominant systems.** One important result is immediately forthcoming for the explicit method. Examining the stability matrix given by equation (4.33), with $h > 0$, the following holds.

If

$$a_{ii} < 0 \qquad i = 1, 2, 3, \ldots, m \tag{4.34}$$

and

$$\sum_{j=1 j\neq i}^{m} \left| \frac{a_{ij}}{a_{ii}} \right| < 1 \qquad i = 1, 2, 3, \ldots, m \tag{4.35}$$

where the condition in equation (4.35) ensures that the system is strictly diagonally dominant, then, since

$$\| M_e \|_\infty = \max_i [e^{a_{ii}h} + | e^{a_{ii}h} - 1 | \sum_{j=1 j\neq i}^{m} \left| \frac{a_{ij}}{a_{ii}} \right| ] \tag{4.36}$$

and

$$0 < e^{a_{ii}h} < 1$$

the infinity norm of the stability matrix will satisfy

$$\| M_e \|_\infty < \max_i [e^{a_{ii}h} + (1 - e^{a_{ii}h})] = 1 \tag{4.37}$$

Consequently, because

$$\| M_e \|_\infty < 1$$

the spectral radius of $M_e$, i.e. the eigenvalue of $M_e$ with largest modulus, fulfils the inequality

$$\rho(M_e) < 1 \qquad (4.38)$$

Hence, all the eigenvalues of $M_e$ have modulus less than unity; and consequently, for any system of ordinary differential equations which have a strictly diagonally dominant system matrix, with the $a_{ii}$ negative for all i, the explicit method is absolutely stable for all step sizes $h > 0$. The same result will also hold for the implicit method given by equation (3.7).

**4.22 General stability properties of the new method.** For the explicit method, stability is ensured if the original system matrix is strictly diagonally dominant with negative diagonal entries. These are not the onlycircumstances under which the method is stable, but they do provide sufficient conditions for stability. In general, it appears impossible to define the necessary conditions under which the method will behave satisfactorily, apart from stating that the eigenvalues of the stability matrix given in equation (4.33) must all have modulus less than unity. Most numerical methods, in monitoring the local error, also determine the way in which the global error is affected. The explicit method, with a time step control, is capable of solving most problems quite satisfactorily, but does tend to require a very small time step with some sets of equations, particularly mathematically stiff systems with large off-diagonal elements in the system stopping diagonal dominance.

**4.23** Consequently, for the type of problem for which stability presents difficulties to the explicit method, it is worthwhile using a method with improved stability, and here the implicit method given by equation (3.7) is considered. It has been found that larger time steps can be used to solve a problem when the implicit method is implemented with the explicit method as the predictor, than by using the explicit method alone to solve the same problem. This will be verified experimentally later in the chapter.

**4.24 Stability properties of the implicit method.** Referring to the stability analysis performed on the explicit method in section 4.18, then if the same analysis is carried out on the implicit method, the resultant matrix equation can be written as:

$$H\underline{\epsilon}_{n+1} = C\underline{\epsilon}_n \tag{4.39}$$

where

$$H = \begin{bmatrix} 1 & (1-e^{a_{11_{n+1}}h})\frac{a_{12_{n+1}}}{a_{11_{n+1}}} & \cdots & (1-e^{a_{11_{n+1}}h})\frac{a_{1m_{n+1}}}{a_{11_{n+1}}} \\ (1-e^{a_{22_{n+1}}h})\frac{a_{21_{n+1}}}{a_{22_{n+1}}} & 1 & \cdots & (1-e^{a_{22_{n+1}}h})\frac{a_{2m_{n+1}}}{a_{22_{n+1}}} \\ \vdots & \vdots & \cdots & \vdots \\ (1-e^{a_{mm_{n+1}}h})\frac{a_{m1_{n+1}}}{a_{mm_{n+1}}} & (1-e^{a_{mm_{n+1}}h})\frac{a_{m2_{n+1}}}{a_{mm_{n+1}}} & \cdots & 1 \end{bmatrix}$$

and

$$C = \begin{bmatrix} e^{a_{11_{n+1}}h} & 0 & 0 & \cdots & 0 \\ 0 & e^{a_{22_{n+1}}h} & 0 & \cdots & 0 \\ 0 & 0 & e^{a_{33_{n+1}}h} & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & e^{a_{mm_{n+1}}h} \end{bmatrix}$$

Provided that H is non-singular, equation (4.39) can be rewritten as

$$\underline{\epsilon}_{n+1} = H^{-1}C\underline{\epsilon}_n \tag{4.40}$$

A stable solution will be obtained if the modulus of the eigenvalues of the (mxm) matrix $H^{-1}C$ are all less than unity. Unfortunately, there is no straightforward way of relating the eigenvalues of this error matrix to the original system eigenvalues, but further manipulation of the error matrix does lead to some important results.

**4.25** Rewriting the matrix H for the general case in the form

$$H = N + C \tag{4.41}$$

where:

$$N = \begin{bmatrix} (1-e^{a_{11_{n+1}}h})\dfrac{a_{11_{n+1}}}{a_{11_{n+1}}} & (1-e^{a_{11_{n+1}}h})\dfrac{a_{12_{n+1}}}{a_{11_{n+1}}} & \cdots & \cdot \\ (1-e^{a_{22_{n+1}}h})\dfrac{a_{21_{n+1}}}{a_{22_{n+1}}} & (1-e^{a_{22_{n+1}}h})\dfrac{a_{22_{n+1}}}{a_{22_{n+1}}} & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ (1-e^{a_{mm_{n+1}}h})\dfrac{a_{m1_{n+1}}}{a_{mm_{n+1}}} & (1-e^{a_{mm_{n+1}}h})\dfrac{a_{m2_{n+1}}}{a_{mm_{n+1}}} & \cdots & (1-e^{a_{mm_{n+1}}h})\dfrac{a_{mm_{n+1}}}{a_{mm_{n+1}}} \end{bmatrix}$$

then the elements of the (mxm) matrix N are given by:

$$N_{ij} = \frac{a_{ij_{n+1}}}{a_{ii_{n+1}}}(1 - e^{a_{ii_{n+1}}h}) \tag{4.42}$$

and the elements of the (mxm) matrix C are given by:

$$C_{ij} = \begin{cases} 0 & i \neq j \\ e^{a_{ii_{n+1}}} & i = j \end{cases} \tag{4.43}$$

**4.26** Equation (4.39) can now be written as:

$$(N + C)\underline{\varepsilon}_{n+1} = C\underline{\varepsilon}_n \tag{4.44}$$

and pre-multiplying by $C^{-1}$, which exists since $e^{\alpha} \neq 0$ for all $\alpha$, will lead to:

$$(C^{-1}N + I)\underline{\varepsilon}_{n+1} = \underline{\varepsilon}_n \tag{4.45}$$

Hence;

$$\varepsilon_{n+1} = (I + C^{-1}N)^{-1}\varepsilon_n \tag{4.46}$$

Defining the matrix G as

$$G = -C^{-1}N \tag{4.47}$$

then equation (4.46) will become

$$\varepsilon_{n+1} = (I - G)^{-1}\varepsilon_n \tag{4.48}$$

The method will be stable provided that the errors decrease throughout the computation. The errors will decay if all the eigenvalues of G lie outside the circle of unit radius lying within the complex h$\lambda$ plane as shown in Figure 4.1, where $\lambda$ refers to the eigenvalues of G. The stability region of the implicit method, with reference to the G matrix, resembles the shape of the Backward Euler Method. The G matrix is given by:

$$G = \begin{vmatrix} (1-e^{-a_{11_{n+1}}h})\frac{a_{11_{n+1}}}{a_{11_{n+1}}} & (1-e^{-a_{11_{n+1}}h})\frac{a_{12_{n+1}}}{a_{11_{n+1}}} & \cdots & \cdot \\ (1-e^{-a_{22_{n+1}}h})\frac{a_{21_{n+1}}}{a_{22_{n+1}}} & (1-e^{-a_{22_{n+1}}h})\frac{a_{22_{n+1}}}{a_{22_{n+1}}} & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ (1-e^{-a_{mm_{n+1}}h})\frac{a_{m1_{n+1}}}{a_{mm_{n+1}}} & (1-e^{-a_{mm_{n+1}}h})\frac{a_{m2_{n+1}}}{a_{mm_{n+1}}} & \cdots & (1-e^{-a_{mm_{n+1}}h})\frac{a_{mm_{n+1}}}{a_{mm_{n+1}}} \end{vmatrix}$$

where the individual elements of G are defined by:

$$G_{ij} = \frac{a_{ij_{n+1}}}{a_{ii_{n+1}}}(1 - e^{-a_{ii_{n+1}}h}) \tag{4.49}$$

**4.27** A similar expression can be formulated for the explicit method. Equation (4.33) can be written as

$$\underline{\epsilon}_{n+1} = (G + I)\underline{\epsilon}_n \tag{4.50}$$

where the individual elements of this G matrix are given by:

$$G_{ij} = \frac{a_{ij_n}}{a_{ii_n}}(e^{a_{ii_n}h} - 1) \tag{4.51}$$

Figure 4.2 shows the stability region for the explicit method in terms of the eigenvalues of the matrix G. This region is similar to the stability region for Euler's method.

**4.28** To investigate the stability properties of the implicit method, it is necessary to analyse the G matrix which is defined for this method. One way is to use Gerschgorin's theorem [69]. If the original system matrix A is assumed to be strictly diagonally dominant with negative diagonal entries, then the elements of -G are given by:

$$-G_{ij} = \frac{a_{ij_{n+1}}}{a_{ii_{n+1}}}(e^{-a_{ii_{n+1}}h} - 1) \tag{4.52}$$

and applying Gerschgorin's theorem, the eigenvalues of -G lie in the union of the circles with centre $e^{-a_{ii_{n+1}}h} - 1$ ( > 0) and radius given by:

$$(e^{-a_{ii_{n+1}}h} - 1)\sum_{j=1}^{m} \left| \frac{a_{ij_{n+1}}}{a_{ii_{n+1}}} \right|$$

Since the radius of the circles is less than $(e^{-a_{ii_{n+1}}h} - 1)$, -G will have eigenvalues with positive real parts, and so G will have eigenvalues with negative real parts. Consequently, the implicit method will be stable for this case, since the eigenvalues of the G matrix lie within the stability region. As an aside, with the diagonal entries all greater than zero, it can be shown that the eigenvalues of the G matrix all lie within the circle corresponding to instability. However, with the real parts of the eigenvalues of the G matrix all greater than zero, it is unlikely that the system being solved is inherently stable, and so it is relative stability that should be sought [56].

4.29 For the implicit method, the required result is that, for all inherently stable engineering systems, then the G matrix will always have eigenvalues with negative real parts. Referring to Figure 4.1, this will ensure that the method is stable. At present, for a certain subsection of problems, inherent stability of the original problem will ensure the numerical stability of the method. The table below indicates the results found; the standard matrix theorems which are needed to substantiate the ideas can be found in Nering [25] and Varga [69].

4.30 First, it is necessary to rewrite the matrix G as

$$G = DA$$

where A is the original system matrix and

$$D = \mathrm{diag}\{d_1, d_2, d_3, \cdots, d_m\}, \quad d_i = \frac{(1 - e^{-a_{ii_{n+1}}h})}{a_{ii_{n+1}}} \quad (h > 0)$$

It can be seen that

$$d_i > 0 \quad \text{and} \quad |d_i| \to 0 \text{ as } |a_{ii_{n+1}}| \to \infty \qquad (4.53)$$

**Cases For Which The Implicit Method Is Stable**

| Cases | Do eigenvalues of A having negative real parts imply that eigenvalues of G have negative real parts? |
|---|---|
| $d_i$ all equal | yes |
| $a_{ii_{n+1}} < 0$ , $i = 1, 2, 3, \ldots, m$ A strictly diagonally dominant | yes |
| A symmetric | yes |
| A triangular | yes |
| $m = 2$, no other restriction on A | yes |

**4.31** The one case proved here will be the last one. In order to ensure that the quadratic equation $\lambda^2 + S_1\lambda + S_2 = 0$ has roots with negative real parts it is necessary to show that $S_1 > 0$ and $S_2 > 0$. For the matrix A, this characteristic equation yields:

$$S_1^A = -(a_{11_{n+1}} + a_{22_{n+1}}) \tag{4.54}$$

and

$$S_2^A = \det A \tag{4.55}$$

So, if A has eigenvalues with negative real parts, then

$$(a_{11_{n+1}} + a_{22_{n+1}}) < 0 \tag{4.56}$$

and

$$\det A > 0 \tag{4.57}$$

The same coefficients for the matrix G are given by:

$$S_1^G = -(d_1 a_{11_{n+1}} + d_2 a_{22_{n+1}}) \qquad (4.58)$$

and

$$S_2^G = d_1 d_2 \det A \qquad (4.59)$$

Referring to equations (4.53), (4.54) and (4.55) it can be seen that $S_1^G > 0$ and $S_2^G > 0$ if $S_1^A > 0$ and $S_2^A > 0$. Hence the required result holds, i.e. the eigenvalues of G have negative real parts.

**4.32 Counter-example.** For the case $m = 3$, then the necessary and sufficient conditions for the characteristic equation $\lambda^3 + S_1\lambda^2 + S_2\lambda + S_3$ to have roots with negative real parts are:

$$S_1 > 0 \qquad (4.60)$$

$$S_3 > 0 \qquad (4.61)$$

$$S_1 S_2 - S_3 > 0 \qquad (4.62)$$

For this case there exists a counter-example which proves that, in general, the inherent stability of the original system does not imply that the implicit method is stable for all values of h.

For the matrix A, given by

$$A = \begin{bmatrix} -1 & -1 & \alpha \\ 1 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix}$$

then the characteristic equation is:

$$\lambda^3 + 2\lambda^2 + (2 - \alpha)\lambda + 1 = 0$$

The $S_i^A$'s are given by the relations:

$$S_1^A = -(a_{11_{n+1}} + a_{22_{n+1}} + a_{33_{n+1}}) \tag{4.63}$$

$$S_2^A = \det \begin{bmatrix} a_{11_{n+1}} & a_{12_{n+1}} \\ a_{21_{n+1}} & a_{22_{n+1}} \end{bmatrix} + \det \begin{bmatrix} a_{11_{n+1}} & a_{13_{n+1}} \\ a_{31_{n+1}} & a_{33_{n+1}} \end{bmatrix} + \det \begin{bmatrix} a_{22_{n+1}} & a_{23_{n+1}} \\ a_{32_{n+1}} & a_{33_{n+1}} \end{bmatrix} \tag{4.64}$$

$$S_3^A = -\det A \tag{4.65}$$

For a stable system, since $S_1^A = 2 > 0$ and $S_3^A = 1 > 0$, it is required that

$$S_1^A S_2^A - S_3^A > 0$$

i.e. that $2(2 - \alpha) > 1$, or $\frac{3}{2} > \alpha$

For the G matrix corresponding to the implicit method, then the coefficient values, the $S_i^{G}$'s, are given by

$$S_1^G = 2(e^h - 1)$$

$$S_2^G = (e^h - 1)^2(1 - \alpha) + h(e^h - 1)$$

$$S_3^G = h(e^h - 1)^2$$

For a stable system, since it is clear that the conditions given in equation (4.60) and (4.61) are satisfied, the requirement is that

$$\frac{(e^h - 1)}{h}(1 - \alpha) > -\frac{1}{2}$$

i.e.

$$\alpha < 1 + \frac{h}{2(e^h - 1)} \tag{4.66}$$

Hence there is a crucial range for $\alpha$, which is $[1, \frac{3}{2}]$, and this will ensure that the eigenvalues of G have negative real parts. Consequently, if $\alpha$ is chosen so as not to satisfy the inequality (4.66) for a particular value of h, then conditions must be placed on h in order to ensure the stability of the implicit method. Therefore, the stability of the original system in this case does not guarantee that the numerical scheme is stable. This result indicates that the implicit method has a finite stability region within the left hand side of the $\lambda h$ plane.

**Introduction To Illustrative Examples**

The following examples have been chosen to illustrate the theoretical aspects of the new method that have been discussed.

**4.33** To demonstrate the effectiveness of monitoring the local error, the two problems solved using the time step control have already been solved by the method, without a time step control, in chapter 3. An explanation of the routine controlling the time step can be found in Appendix B.

**4.34** To demonstrate the theoretical stability properties, the explicit method is applied to both a diagonally dominant, and a non-diagonally dominant system; the stability matrix for both of these problems is then studied. Then, the explicit method is applied to a stiff problem and compared with the explicit-implicit pair applied to the same problem, in order to show the improved stability properties of the implicit method. Finally, several examples are chosen to demonstrate further the stability properties of the implicit method.

## Time Step Control

**4.35 Example 1 - Linear actuator circuit.** This problem is described in section 3.29, and the differential equations being solved are given by equations (3.30), (3.31) and (3.32). The data values remain the same and these are given in table 3.2. The problem being solved is moderately stiff and non-oscillatory and provides a good test, both for the new method, and for the time step control. The way in which the predictor-corrector pair is applied can also be found in sections 3.30 - 3.32. The results for the problem have been obtained by using two tolerances, $1 \times 10^{-3}$ and $1 \times 10^{-4}$, which control the integration time step by limiting the largest mixed local error per integration step. The initial time step is chosen as $1 \times 10^{-3}$ so as to be of comparative order to the smallest time constant. Figure 4.3 shows the manner in which the step sizes change during the integration process. Once the fast transient has decayed, the step size increases to a value that gives an acceptable approximation to the slowly-varying transient. This value is larger than $1 \times 10^{-3}$, the step size that the method needs when used as a fixed step method in order to ensure sufficiently accurate results.

**4.36** The overall C.P.U. times taken by the variable time step method were 7.9 and 8.1 seconds for the tolerances of $1 \times 10^{-3}$ and $1 \times 10^{-4}$ respectively. These values demonstrate that there is a saving to be made by using a time step control with the method. However, because the single step method does not monitor the local error, and is reliant only on the steplength input by the user, and the tolerance value used by the iteration scheme, it is possible to choose a step size for which the method is bordering on instability for some or all of the integration, but still provides reasonable results. This step size may give run times which over-amplify the true performance of the method.

**4.37 Example 2 - Fifth order system.** A full description for this problem can be found in section 3.40, with the system equations being given in section 3.41. The data values remain the same, and these are given in tables 3.4 and 3.5. The integration again takes place over 6 seconds of real time. For this problem, the number of steps used are too numerous to be represented graphically, and a discussion of the overall C.P.U. time used by the method will provide sufficient information. When solving the initial problem, to complete the simulation on the interval [0 , 6] seconds, then with a time step control, the new method took 5 minutes and 21.24 seconds, in comparison with Gear's method which took 3 minutes and 41.2 seconds. The initial time step chosen for the new method was $1 \times 10^{-5}$ and throughout the simulation, the largest time step used by the method was of the order $1 \times 10^{-3}$, and the smallest of the order $1 \times 10^{-7}$. Gear's method also used a similar range of time steps.

**4.38** For the problem which had a low viscous friction coefficient, a higher bulk modulus and smaller pipe volumes, leading to high mathematical stiffness and very oscillatory results, then the new variable step method proved to require less time than Gear's method to provide identical results. These results are shown in Figures 3.21 and 3.22. The C.P.U. time required was 28 minutes and 19.56 seconds in comparison with Gear's method which took 30 minutes and 49.87 seconds to give reasonable results. For this problem, Gear's method sank to a time step of the order $1 \times 10^{-11}$, in comparison to the new method which only sank to a step of the order $1 \times 10^{-8}$.

**4.39** Table 4.1 shows the full set of C.P.U. timing results obtained by solving these two problems using the new method with a time step control. The results obtained show the merits of monitoring the local error at each time step, and the computational efficiency that is involved with doing so. A great deal of experimental work has been done with the time step control, in an effort to ensure that the optimal efficiency of the method to which it is applied is achieved. This work is involved with the choosing of the tolerance that is used by the step control to monitor the local error, and in deciding the precise strategy for determining the time step to be used over the next integration step.

**Stability Properties**

**4.40 Example 1 - Diagonally dominant system - Accumulator-actuator circuit.** The circuit diagram for this problem is shown in Figure 4.4. The input flow to the system is assumed to be constant, with the accumulator providing a constant supply pressure to two small actuators. The actuators drive two masses. The control valves for the actuators are assumed to open suddenly, and to operate with negligible pressure drop. The flow taken by the two actuators is assumed to be small in comparison to the system flow.

**4.41 System equations.** For the accumulator, then

$$Pv^n = \text{constant} \tag{4.67}$$

where:

P is the pressure in the system
v is the volume of gas in the accumulator
n is the polytropic index of the gas

Hence

$$P'v^n + nv^{n-1}v'P = 0$$

and so

$$P' = -\frac{nP}{v}v'$$

Applying this to the system will give

$$P' = -\frac{nP}{v}Q_i \tag{4.68}$$

where $Q_i$ is the main flow

The differential equations describing the acceleration of the masses are given by:

$$u_1' = P\frac{A_1}{M_1} - \frac{f_1 u_1}{M_1} \tag{4.69}$$

$$u_2' = P\frac{A_2}{M_2} - \frac{f_2 u_2}{M_2} \tag{4.70}$$

where:

$u_{1,2}$ are the velocities of mass 1 and 2 respectively
$M_{1,2}$ are the masses of the actuator loads
$f_{1,2}$ are the viscous friction coefficients for the two actuators

Writing the differential equations in matrix form will give:

$$\begin{bmatrix} P' \\ u'_1 \\ u'_2 \end{bmatrix} \begin{bmatrix} -\frac{n}{v}Q_i & 0 & 0 \\ \frac{A_1}{M_1} & -\frac{f_1}{M_1} & 0 \\ \frac{A_2}{M_2} & 0 & -\frac{f_2}{M_2} \end{bmatrix} \begin{bmatrix} P \\ u_1 \\ u_2 \end{bmatrix} \tag{4.71}$$

**4.42** The data values for this problem are given in table 4.2, and when substituted into equation (4.71), they give the following system matrix:

$$\begin{bmatrix} -2.76 \text{ x } 10^{-2} & 0 & 0 \\ 4.91 \text{ x } 10^{-6} & -9.82 \text{ x } 10^{1} & 0 \\ 2.46 \text{ x } 10^{-5} & 0 & -4.91 \text{ x } 10^{2} \end{bmatrix} \tag{4.72}$$

This matrix is easily seen as being diagonally dominant. The eigenvalues of this matrix are:

$\lambda_1 = -2.76 \text{ x } 10^{-2}$
$\lambda_2 = -9.82 \text{ x } 10^{1}$
$\lambda_3 = -4.91 \text{ x } 10^{2}$

and the stiffness ratio is given by:

$S = 1.78 \text{ x } 10^{4}$

which indicates a problem of moderate stiffness.

The error matrix for this problem, when the explicit method is applied, is:

$$M_e = \begin{vmatrix} e^{a_{11_n}h} & (e^{a_{11_n}h} - 1)\frac{a_{12_n}}{a_{11_n}} & (e^{a_{11_n}h} - 1)\frac{a_{13_n}}{a_{11_n}} \\ (e^{a_{22_n}h} - 1)\frac{a_{21_n}}{a_{22_n}} & e^{a_{22_n}h} & (e^{a_{22_n}h} - 1)\frac{a_{23_n}}{a_{22_n}} \\ (e^{a_{33_n}h} - 1)\frac{a_{31_n}}{a_{33_n}} & (e^{a_{33_n}h} - 1)\frac{a_{32_n}}{a_{33_n}} & e^{a_{33_n}h} \end{vmatrix} \tag{4.73}$$

where the $a_{ij}$ are given in equation (4.72)

Taking a typically large value for h, for example h = 1, then M will become

$$\begin{vmatrix} 9.73 \times 10^{-1} & 0 & 0 \\ 5.0 \times 10^{-8} & 2.25 \times 10^{-43} & 0 \\ 5. \times 10^{-8} & 0 & e^{-491} \end{vmatrix} \tag{4.74}$$

which has the eigenvalues given by:

$\mu_1 = 9.73 \times 10^{-1}$
$\mu_2 = 2.25 \times 10^{-43}$
$\mu_3 = e^{-491}$

Hence the modulus of all three eigenvalues is less than unity, ensuring that any error introduced will decay as the scheme marches forward in time. For other values of h the same result will hold. The problem was solved using the new method with various fixed time steps, to see if instability was demonstrated for any value of h. However, for the results obtained instability was not apparent for any value of h, although the accuracy of the results is affected because of the low order of the method.

**4.43 Example 2 - Non-diagonally dominant system - Orifice-motor circuit.** This example is a hydraulic circuit for which the system matrix of the differential equations that describe the behaviour of the circuit is non-diagonally dominant. The circuit diagram is shown in Figure 4.5, and the system equations are given below. A constant flow passes through an orifice restrictor and then drives a hydraulic motor. The pump and the motor are assumed to exhibit no slip flow or torque losses in order to simplify the analysis. The o.d.e.'s are:

$$\frac{dP_1}{dt} = \frac{B}{v_1}(Q_1 - Q_2) \tag{4.75}$$

$$\frac{dP_2}{dt} = \frac{B}{v_2}(Q_2 - Q_m) \tag{4.76}$$

$$\frac{d\omega_m}{dt} = \frac{P_2 D_m}{J} - \frac{T_l}{J} - \frac{f\omega_m}{J} \tag{4.77}$$

where:

$P_{1,2}$ are the oil pressures in the pipes either side of the orifice
$Q_{1,2}$ are the oil flow rates in the pipes either side of the orifice
$Q_m$ is the motor flow rate
B is the bulk modulus of the hydraulic oil
$v_{1,2}$ are the volumes of the pipes either side of the orifice
J is the moment of inertia
$\omega_m$ is the angular velocity of the motor
f is the viscous friction coefficient
$D_m$ is the motor displacement
$T_l$ is the load torque

The flow rates $Q_2$ and $Q_m$ are given by:

$$Q_2 = k_o(P_1 - P_2) \tag{4.78}$$

and

$$Q_m = \omega_m D_m \tag{4.79}$$

where $k_o$ is the flow coefficient for the orifice

Substituting equations (4.78) and (4.79) into equations (4.75) and (4.76) will, on rearranging, give:

$$\frac{dP_1}{dt} = -\frac{Bk_o}{V_1}P_1 + \frac{B}{V_1}(Q_1 + k_oP_2) \tag{4.80}$$

and

$$\frac{dP_2}{dt} = -\frac{Bk_o}{V_2}P_2 + \frac{B}{V_2}(k_oP_1 - \omega_m D_m) \tag{4.81}$$

Putting equations (4.80), (4.81) and (4.77) into matrix form will lead to

$$\begin{bmatrix} P_1' \\ P_2' \\ \omega_m' \end{bmatrix} = \begin{bmatrix} -\frac{Bk_o}{V_1} & \frac{Bk_o}{V_1} & 0 \\ \frac{Bk_o}{V_2} & -\frac{Bk_o}{V_2} & -\frac{BD_m}{V_2} \\ 0 & \frac{D_m}{J} & -\frac{f}{J} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ \omega_m \end{bmatrix} + \begin{bmatrix} \frac{BQ_1}{V_1} \\ 0 \\ -\frac{T_l}{J} \end{bmatrix} \tag{4.82}$$

The data values for this problem are given in table 4.3. Substituting these values into equation (4.82) will lead to the system matrix

$$\begin{bmatrix} -1.24 & 1.24 & 0 \\ 1.24 \times 10^2 & -1.24 \times 10^2 & -4.2 \times 10^8 \\ 0 & 1.5 \times 10^{-6} & -2.4 \times 10^{-2} \end{bmatrix} \tag{4.83}$$

which is not diagonally dominant. The eigenvalues of the system are given by:

$\lambda_1 = -2.07$
$\lambda_2 = -1.2 \times 10^2$
$\lambda_3 = -3.13$

If the explicit method is used to solve this problem, then the stability matrix for the method is given by equation (4.73), where the coefficient values can be obtained from the matrix given in equation (4.83), and hence, taking a typically large h, e.g. h = 1, the stability matrix will become:

$$\begin{bmatrix} 2.88 \times 10^{-1} & 7.12 \times 10^{-1} & 0 \\ 1 & 1.00 \times 10^{-54} & -3.38 \times 10^{6} \\ 0 & 1.48 \times 10^{-6} & 9.76 \times 10^{-1} \end{bmatrix} \tag{4.84}$$

The eigenvalues for this matrix are given by:

$$\lambda_1 = 1.71 \times 10^{-1}$$
$$\lambda_2 = 5.47 \times 10^{-1} + 2.02\,i$$
$$\lambda_3 = 5.47 \times 10^{-1} - 2.02\,i$$

$|\lambda_2| = |\lambda_3| = 2.095$, and since this is greater than one, a stable solution to the problem using the explicit method may not be possible, since errors could grow throughout the integration. It is not possible to tell when the solution will become unstable, but the deciding factor here is that the explicit method should not be used.

**4.44 Example 3.** The next problem compares the performance of the explicit method with the predictor-corrector pair in solving a mathematically stiff problem. The example is a well known stiff test problem [32], and has been solved, first by the explicit method, and then by the explicit and implicit methods acting as a pair. It is a non-linear problem that comes from the field of chemical kinetics, and was originally published by Robertson [70]. It involves three equations, these being:

$$y_1' + 0.04y_1 - 10^4 y_2 y_3 = 0 \tag{4.85}$$

$$y_2' - 0.04y_1 + 10^4 y_2 y_3 + 3 \times 10^7 y_2^2 = 0 \tag{4.86}$$

$$y_3' - 3 \times 10^7 y_2^2 = 0 \tag{4.87}$$

The initial conditions are

$$y_1(0) = 1, \; y_2(0) = 0, \; y_3(0) = 0$$

Also,

$$y_1(t) + y_2(t) + y_3(t) = 1, \quad \text{for all } t \tag{4.88}$$

The problem is solved between 0 and 100 seconds.

Writing equations (4.85), (4.86) and (4.87) as a system will give:

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \end{bmatrix} = \begin{bmatrix} -.04 & 10^4 y_3 & 0 \\ 0.04 & -3 \times 10^7 y_2 - 10^4 & \\ 0 & 3 \times 10^7 y_2 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{4.89}$$

and the Jacobian matrix for the system is given by:

$$-\begin{bmatrix} 0.04 & -10^4 y_3 & -10^4 y_2 \\ -.04 & 10^4 y_3 + 6 \times 10^7 y_2 & 10^4 y_2 \\ 0 & -6 \times 10^7 y_2 & 0 \end{bmatrix} \tag{4.90}$$

This matrix has one zero eigenvalue and two real negative eigenvalues which are functions of time. Figure 4.6 shows the varying time constants given by:

$$\tau(t) = \mathrm{Re}(\frac{-1}{\lambda(t)})$$

and the stiffness ratio, S(t), which varies from $O(10^4)$ to $O(10^5)$.

**4.45 Applying the new integration method.** Because of the relationship given by equation (4.88), it is possible to rewrite equations (4.85), (4.86) and (4.87) as two differential equations by substituting for $y_3$ in equations (4.85) and (4.86). The new system will be:

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -0.04 - 10^4 y_2 & 10^4 - 10^4 y_2 \\ 0.04 + 10^4 y_2 & -10^4 + 10^4 y_2 - 3 \times 10^7 y_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{4.91}$$

and

$$y_3(t) = 1 - y_1(t) - y_2(t)$$

The system matrix for this problem, given by equation (4.91), is initially non-diagonally dominant, as is the system matrix of the original full set of equations. Hence stability for the explicit method cannot be guaranteed. The coefficients for the new explicit method are:

$$a_{1_n} = (-0.04 - 10^4 y_{2_n}) \quad b_{1_n} = (10^4 - 10^4 y_{2_n}) y_{2_n}$$

$$a_{2_n} = 10^4 + 10^4 y_{2_n} - 3 \times 10^7 y_{2_n} \quad b_{2_n} = (0.04 + 10^4 y_{2_n}) y_{1_n}$$

The coefficient values for the implicit method are similar to those for the explicit method, with the subscripts n replaced by subscripts n+1, on the right hand side of the equalities given in the equations above.

**4.46 Results for the two methods.** The results have been obtained using two tolerances, $1 \times 10^{-3}$ and $1 \times 10^{-4}$. This tolerance, for both the methods, is used to control the integration time step by limiting the largest relative error per integration step. The tolerance is also used by the Newton-Raphson iteration scheme in conjunction with the implicit method to ensure that the iterates have converged to an acceptably accurate value. The initial step size is chosen as $h_0 = 1 \times 10^{-4}$ so as to be of comparable order to the smallest time constant. Figure 4.7 shows the manner in which the step size increases during the integration for the explicit method and Figure 4.8 shows the manner in which the step size increases for the implicit method. Over the range $t = [0, 1 \times 10^{-3}]$, then the rapidly varying transient is accurately approximated using a small step size. Once this transient has decayed sufficiently, the step size increases to a value that gives an acceptably accurate approximation to the slowly varying transient. The amount of computation used by the two methods in solving this problem is summarised in table 4.4. The explicit method does require smaller time steps in order to ensure that the local error criteria is satisfied, and this indicates that the stability properties of the method are a handicap for problems of this type. The results for the predictor-corrector pair demonstrates the improved stability properties of the implicit method which holds with the theory developed earlier in the chapter.

**4.47** As a comparison, this problem has also been solved by Hall and Watt [32] using a weighted combination of the Forward Euler and the Backward Euler method. The method they used is:

$$y_{n+1} = y_n + h(0.55f(t_{n+1},y_{n+1}) + 0.45f(t_n,y_n)) \qquad (4.92)$$

This method is A-stable, and leads to the formula

$$y_{n+1} = \frac{(1 + 0.45h\lambda)}{(1 - 0.55h\lambda)} y_n$$

when applied to the test equation

$$y' = \lambda y \quad \lambda < 0 \quad y(0) = 1$$

Using L'hopital's rule, it can be seen that

$$\frac{y_{n+1}}{y_n} = \frac{(1 + 0.45h\lambda)}{(1 - 0.55h\lambda)} \rightarrow \frac{-9}{11} \text{ as } |h\lambda| \rightarrow \infty$$

and so the method is A-stable for all h with $\lambda < 0$. The order of the method is one with the principal error term being $\frac{1}{20}h^2y(\xi)''$. Again, the problem was solved using two values of the tolerance parameter, $1 \times 10^{-3}$ and $1 \times 10^{-4}$. Figure 4.9 shows how the step size changes over the integration and table 4.5 gives a summary of the amount of computation involved in solving the problem using the method given by equation (4.92). The method is better than the explicit form of the new method for solving this problem, but not as good as the predictor-corrector pair.

The next set of examples illustrate the stability properties of the implicit method. The first problem is the non-diagonally dominant system discussed in section 4.43.

**4.48 Example 4 - Non-diagonally dominant system.** The resultant system, with reference to equation (4.82), is given by:

$$\begin{bmatrix} P_1' \\ P_2' \\ P_3' \end{bmatrix} = \begin{bmatrix} -\frac{Bk}{v_1} & \frac{Bk}{v_1} & 0 \\ \frac{Bk}{v_2} & -\frac{Bk}{v_2} & -\frac{BD_m}{v_2} \\ 0 & 0 & -\frac{f}{J} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} + \begin{bmatrix} \frac{BQ_1}{v_1} \\ 0 \\ -\frac{T_1}{J} \end{bmatrix}$$

Substituting the data values given in table 4.4 will lead to the system matrix

$$\begin{bmatrix} -1.24 & 1.24 & 0 \\ 1.24 \times 10^2 & -1.24 \times 10^2 & -4.2 \times 10^8 \\ 0 & 1.5 \times 10^{-6} & -2.4 \times 10^{-2} \end{bmatrix}$$

and the problem is inherently stable since the system eigenvalues are:

$$\lambda_1 = -2.07$$
$$\lambda_2 = -1.2 \times 10^2$$
$$\lambda_3 = -3.13$$

If the implicit method is used to solve this problem, then the stability matrix for the method is given by:

$$[I - G]^{-1} \qquad (4.93)$$

where I is the (3x3) identity matrix, and

$$G = \begin{bmatrix} (1-e^{-a_{11_{n+1}}h})\frac{a_{11_{n+1}}}{a_{11_{n+1}}} & (1-e^{-a_{11_{n+1}}h})\frac{a_{12_{n+1}}}{a_{11_{n+1}}} & (1-e^{-a_{11_{n+1}}h})\frac{a_{13_{n+1}}}{a_{11_{n+1}}} \\ (1-e^{-a_{22_{n+1}}h})\frac{a_{21_{n+1}}}{a_{22_{n+1}}} & (1-e^{-a_{22_{n+1}}h})\frac{a_{22_{n+1}}}{a_{22_{n+1}}} & (1-e^{-a_{22_{n+1}}h})\frac{a_{23_{n+1}}}{a_{22_{n+1}}} \\ (1-e^{-a_{33_{n+1}}h})\frac{a_{31_{n+1}}}{a_{33_{n+1}}} & (1-e^{-a_{33_{n+1}}h})\frac{a_{32_{n+1}}}{a_{33_{n+1}}} & (1-e^{-a_{33_{n+1}}h})\frac{a_{33_{n+1}}}{a_{33_{n+1}}} \end{bmatrix}$$

A sufficient condition for the method to be stable is that all the eigenvalues of the G matrix have negative real parts. This will ensure that the errors decrease throughout the integration process. Evaluating the G matrix, taking h = 1, will give:

$$G = \begin{bmatrix} -2.47 & 2.47 & 0 \\ 9.98 \text{ x } 10^{53} & -9.98 \text{ x } 10^{53} & -3.37 \text{ x } 10^{60} \\ 0 & 1.52 \text{ x } 10^{-6} & -2.4 \text{ x } 10^{-2} \end{bmatrix} \tag{4.94}$$

The necessary and sufficient conditions for this matrix to have three eigenvalues with negative real parts are:

$$S_1^G > 0$$

$$S_3^G > 0$$

$$S_1^G S_2^G - S_3^G > 0$$

where the $S_i$'s are defined in section 4.32.

Now

$$S_1^G = 9.98 \text{ x } 10^{53} > 0$$

$$S_2^G = 5.14 \text{ x } 10^{54}$$

$$S_3^G = 1.26 \text{ x } 10^{55} > 0$$

and

$$S_1^G S_2^G - S_3^G = 5.13 \text{ x } 10^{108} - 1.26 \text{ x } 10^{55} > 0$$

Consequently, the eigenvalues of the G matrix have negative real parts, and so the implicit method will be stable for h = 1, although the explicit method is not stable for this same value of h.

**4.49 Example 5 - Symmetric matrix.** This example takes a symmetric system with negative eigenvalues and shows that this is enough to ensure that the G matrix has negative eigenvalues. If the system matrix is given by:

$$A = \begin{bmatrix} -2 & -1 & -1 \\ -1 & -3 & -2 \\ -1 & -2 & -4 \end{bmatrix}$$

then the eigenvalues of this matrix are

$\lambda_1 = -1.31$
$\lambda_2 = -6.05$
$\lambda_3 = -1.64$

Taking h = 1, the G matrix for this problem is given by:

$$G = \begin{bmatrix} -6.39 & -3.19 & -3.19 \\ -6.36 & -19.09 & -12.72 \\ -13.40 & -26.80 & -53.60 \end{bmatrix}$$

and the eigenvalues of this matrix are:

$\lambda_{1_G} = -4.94$
$\lambda_2^G = -62.70$
$\lambda_3^G = -11.44$

Hence the eigenvalues of the stability matrix for the implicit method are all less than one in modulus, and so the implicit method is suitable to be used to solve this problem, from the point of view of stability.

**4.50 Example 6 - Lower triangular matrix.** This example presents a triangular system matrix, in this case a lower triangular matrix, with negative eigenvalues, and shows that for this system, the stability matrix of the implicit method has only eigenvalues with modulus less than one.

If the system matrix is given by:

$$A = \begin{bmatrix} -2 & 0 & 0 \\ -3 & -3 & 0 \\ -1 & -2 & -4 \end{bmatrix}$$

then the eigenvalues for the system are

$\lambda_1 = -2.0$
$\lambda_2 = -3.0$
$\lambda_3 = -4.0$

If h is taken as 1, then the G matrix for the implicit method is given by:

$$G = \begin{bmatrix} -6.39 & 0 & 0 \\ -19.09 & -19.09 & 0 \\ -13.40 & -26.80 & -53.60 \end{bmatrix}$$

and the eigenvalues for this matrix are

$\lambda_1^G = -6.39$ $\lambda_2^G = -19.09$ $\lambda_3^G = -53.60$

This again illustrates the theoretical results obtained in section 4.30, and the last three examples highlight the cases for which the inherent stability of the original system will ensure the stability of the numerical method.

## Conclusions

4.51 Although there are no general conclusions that can be drawn with regards to the stability properties of either of the methods with relation to the original system matrix, it is apparent that the stability properties of the implicit method are far superior to those of the explicit method, which is to be expected since implicit methods in general have much larger stability regions than corresponding explicit methods. Also, for the examples that have been chosen so far, numerical instability has not been demonstrated provided that a sufficiently small time step and error tolerance have been used when applying the methods. One important result that has been established is the numerical stability of the explicit method when it is applied to strictly diagonally dominant systems. This leads to an interesting branch of work that will be investigated in chapter 7. This work involves applying the explicit method to solve sets of partial differential equations which will reduce to diagonally dominant sets of stiff ordinary differential equations when discretised in both the time and the spatial directions.

4.52 The new method has been analysed, and a time step control has also been developed for use with both the explicit and implicit forms. Although the method is only first order accurate, it does have desirable stability properties, and consequently may prove to be a useful integration method for solving sets of differential equations that are mathematically stiff. Having tested and studied the new method, the next chapter looks at its implementation inside of the HASP package as a general purpose integrator.

| | Predictor-Corrector Pair | | Gear's Method | | |
|---|---|---|---|---|---|
| Test Problem | Tolerance (secs) | C.P.U. (secs) | Tolerance (secs) | C.P.U (secs) | Maximum Stiffness Ratio |
| Problem 1 t=[0,5] secs | $1 \times 10^{-3}$<br>$1 \times 10^{-4}$ | 7.9<br>8.1 | $1 \times 10^{-5}$ | 21 | $1 \times 10^{3}$ |
| Problem 2 Case A t=[0,6] secs | $1 \times 10^{-4}$ | $3.41 \times 10^{2}$ | $1 \times 10^{-5}$ | $2.212 \times 10^{2}$ | $1 \times 10^{3}$ |
| Problem 2 Case B t=[0,6] secs | $1 \times 10^{-4}$ | $1.700 \times 10^{3}$ | $1 \times 10^{-6}$ | $1.849 \times 10^{3}$ | $1 \times 10^{5}$ |

**TABLE 4.1 C.P.U. TIMES FOR PROBLEMS 1 AND 2 USING A TIME STEP CONTROL**

| Symbol | Description | Value |
|---|---|---|
| $A_1$ | Area of the piston in actuator 1 | $4.91 \times 10^{-4} m^2$ |
| $A_2$ | Area of the piston in actuator 2 | $4.91 \times 10^{-4} m^2$ |
| $f_{1,2}$ | Viscous friction coefficients | $9.82 \times 10^3 N/m/sec$ |
| $M_1$ | Mass to be moved by actuator 1 | $1.0 \times 10^2$ Kg |
| $M_2$ | Mass to be moved by actuator 2 | $2.0 \times 10^1$ Kg |
| v | Gas volume | $5.8 \times 10^{-2} m^3$ |
| n | polytropic index of gas | 1.6 |
| $Q_i$ | Main flow | $1 \times 10^{-2}$ $m^3/sec$ |
| P | Oil pressure in pipe - initial value | $1 \times 10^7$ $N/m^2$ |
| $u_{1,2}$ | actuator velocities - initial values | 0 m/sec |

**TABLE 4.2 DATA VALUES USED FOR THE ACCUMULATOR-ACTUATOR CIRCUIT**

| Symbol | Description | Value |
|---|---|---|
| B | Bulk modulus of the hydraulic oil | $1.4 \times 10^9$ N/m$^2$ |
| f | Viscous friction coefficient | $4.8 \times 10^{-1}$ Nm/rad/sec |
| k | Orifice flow coefficient | $1.315 \times 10^{-14}$ m$^3$/sec/N/m$^2$ |
| $T_l$ | Load torque | $3 \times 10^2$ Nm |
| $Q_1$ | Input flow rate | $3.5 \times 10^{-3}$ m$^3$/sec |
| J | The inertia of the load | $2. \times 10^1$ Kgm$^2$ |
| $v_1$ | Volume of pipe 1 | $1 \times 10^{-2}$ m$^3$ |
| $v_2$ | Volume of pipe 2 | $1 \times 10^{-4}$ m$^3$ |
| $D_m$ | Motor displacement | $3. \times 10^{-5}$ m$^3$/rad |
| $\omega_m$ | Angular velocity of the motor - initial value | $1. \times 10^2$ rads/sec |
| $P_1$ | oil pressure in pipe 1 - initial value | $5 \times 10^6$ N/m$^2$ |
| $P_2$ | oil pressure in pipe 2 - initial value | $4 \times 10^6$ N/m$^2$ |

**TABLE 4.3 DATA VALUES FOR ORIFICE-MOTOR CIRCUIT**

|  | Accuracy parameter $\epsilon$ | $10^{-3}$ |  |  | $10^{-4}$ |  |  |
|---|---|---|---|---|---|---|---|
| Method | Integration range | 0-1 | 0-10 | 0-100 | 0-1 | 0-10 | 0-100 |
| Explicit | No. of successful steps | 230 | 292 | 391 | 1606 | 2666 | 3613 |
|  | No. of rejected steps | 10 | 14 | 17 | 14 | 18 | 21 |
| Method | No. of times steps size doubled | 6 | 8 | 9 | 6 | 8 | 11 |
| Implicit | No. of successful steps | 31 | 34 | 43 | 140 | 180 | 203 |
|  | No. of rejected steps | 1 | 2 | 3 | 4 | 2 | 6 |
| Method | No. of times step size doubled | 10 | 12 | 13 | 10 | 12 | 14 |

**TABLE 4.4 PERFORMANCE OF NEW METHOD ON STABILITY EXAMPLE 3**

| Accuracy parameter $\epsilon$ | $10^{-3}$ |  |  | $10^{-4}$ |  |  |
|---|---|---|---|---|---|---|
| Integration range | 0-1 | 0-10 | 0-100 | 0-1 | 0-10 | 0-100 |
| No. of successful steps | 41 | 55 | 65 | 111 | 160 | 214 |
| No. of rejected steps | 1 | 1 | 2 | 3 | 3 | 3 |
| No. of times steps size doubled | 12 | 15 | 20 | 12 | 15 | 18 |

**TABLE 4.5 PERFORMANCE OF METHOD FROM HALL & WATT ON STABILITY EXAMPLE 3**

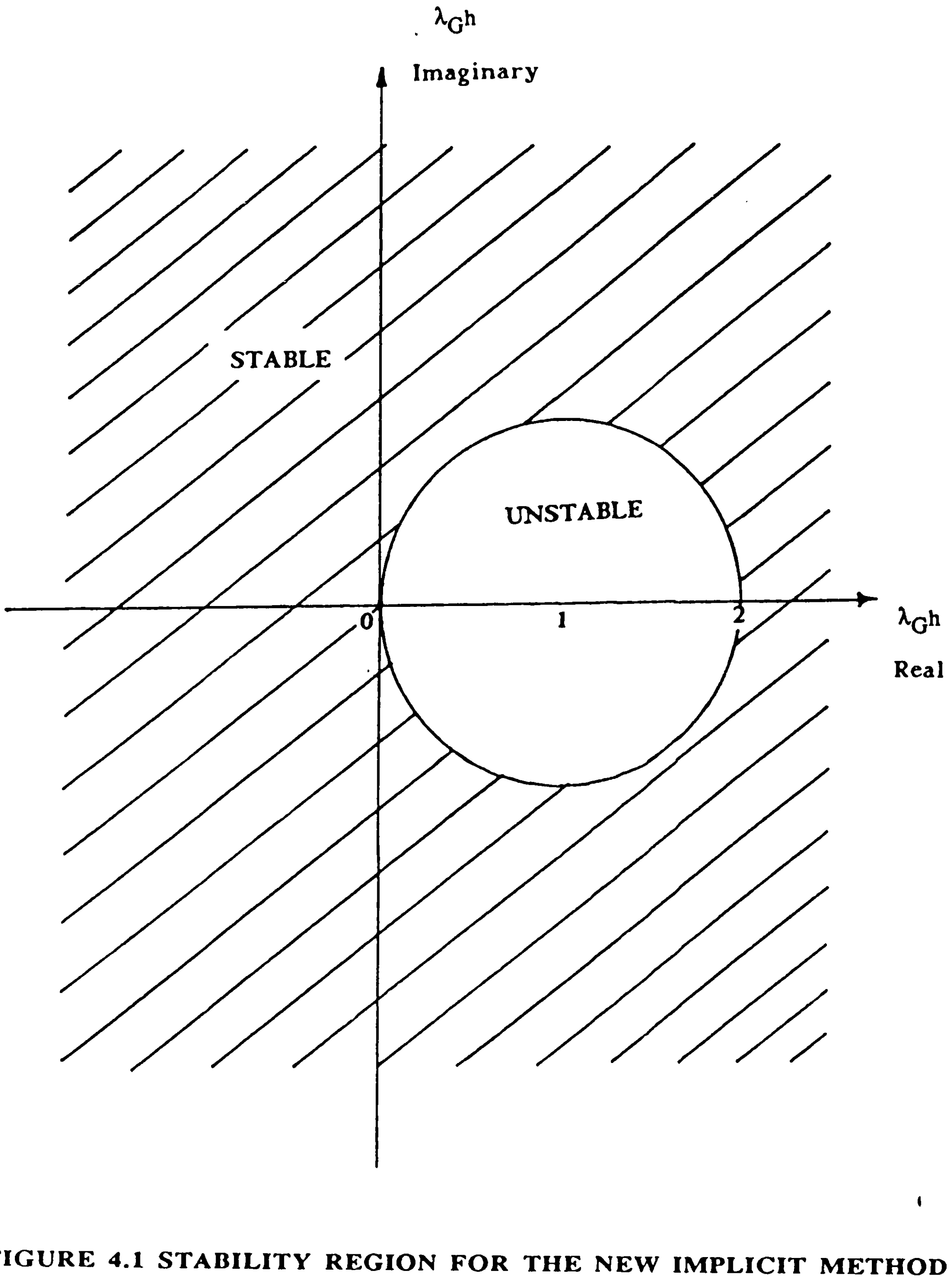


FIGURE 4.1 STABILITY REGION FOR THE NEW IMPLICIT METHOD

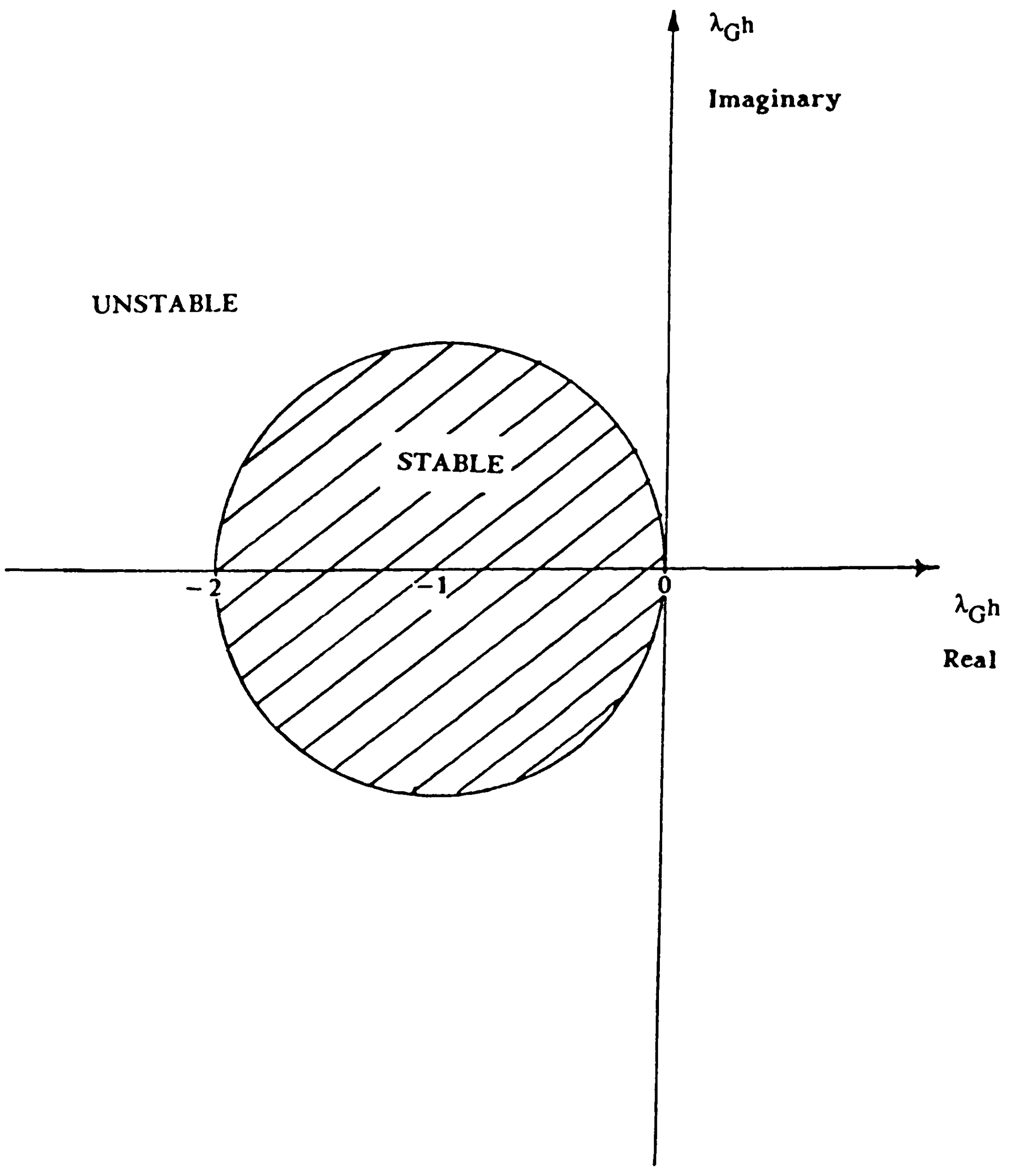


**FIGURE 4.2 STABILITY REGION FOR THE NEW EXPLICIT METHOD**

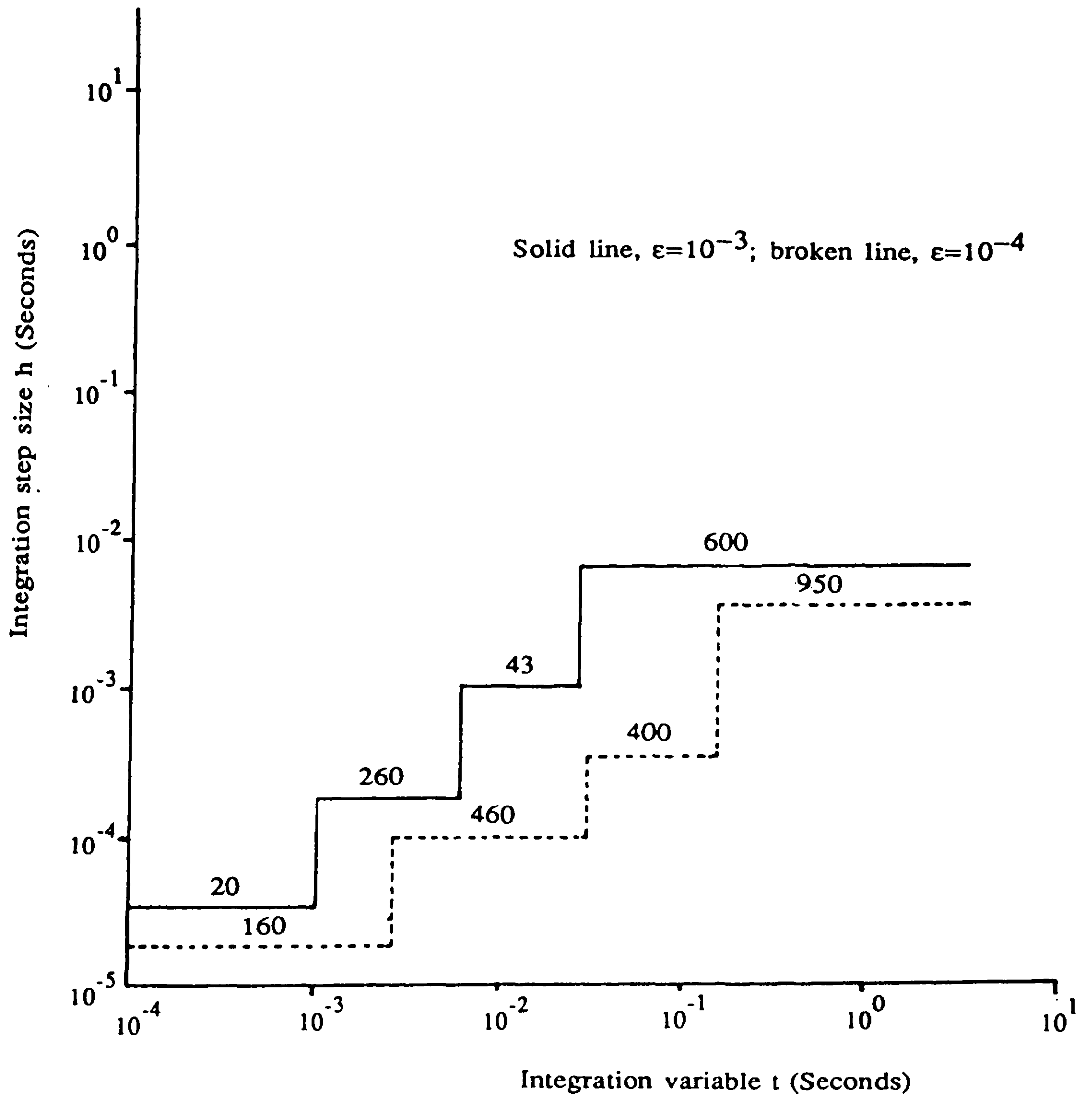


FIGURE 4.3 VARIATION OF STEP SIZE AND NUMBER OF STEPS IN SOLUTION OF PROBLEM 1 WITH TIME STEP CONTROL

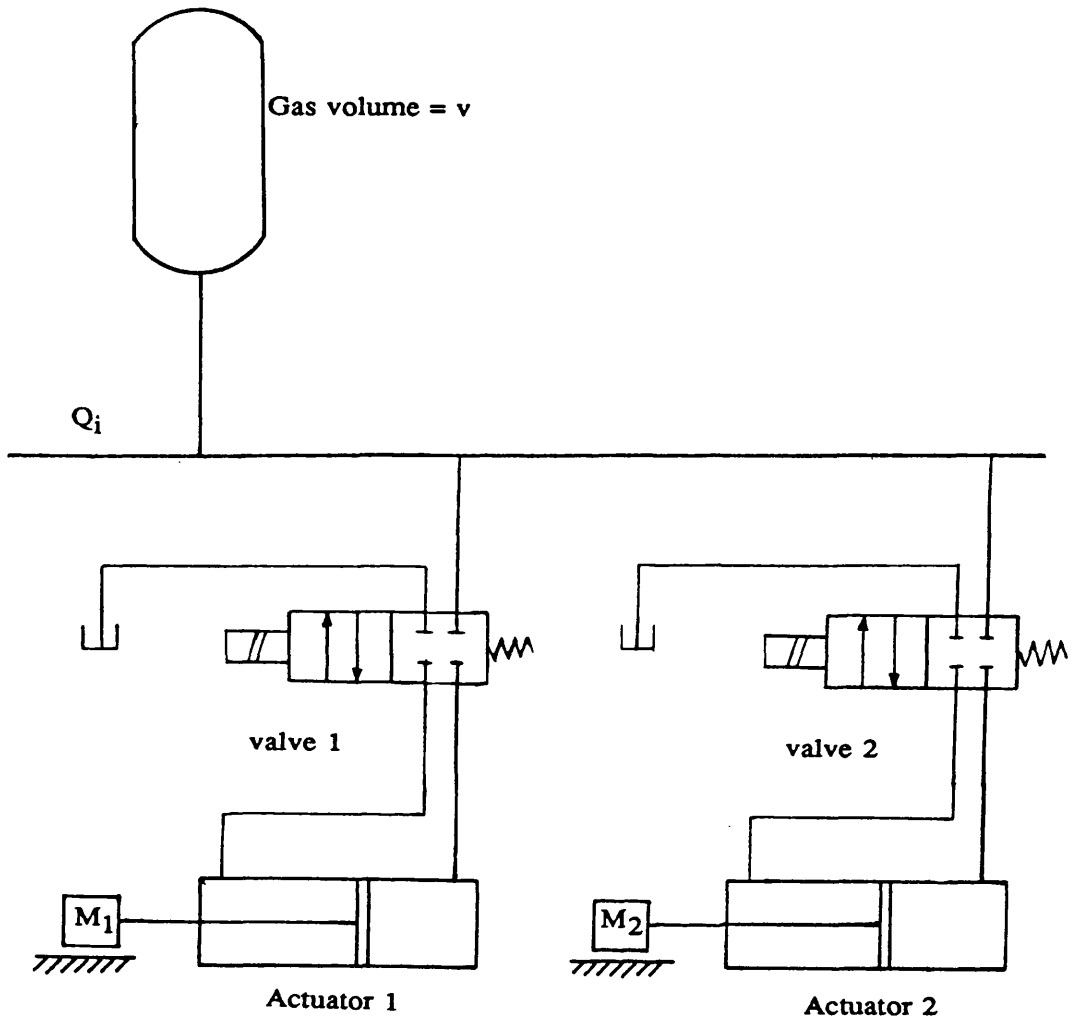


FIGURE 4.4 ACCUMULATOR-ACTUATOR CIRCUIT FOR STABILITY EXAMPLE 1

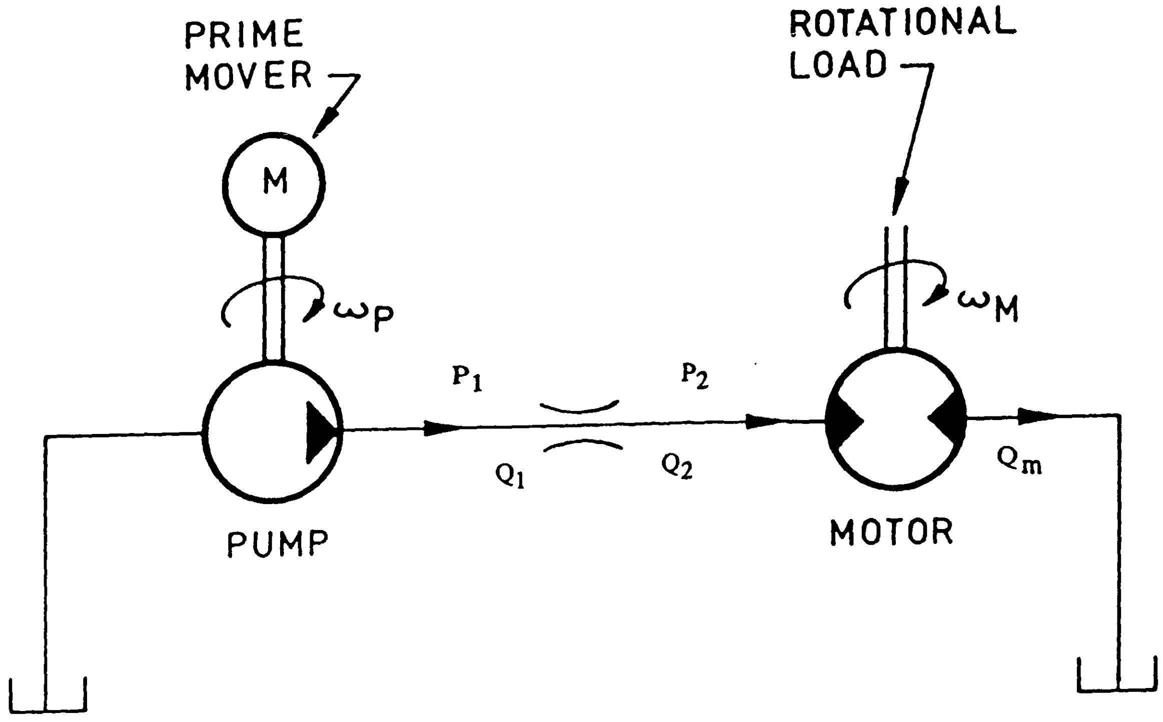


FIGURE 4.5 ORIFICE-MOTOR CIRCUIT FOR STABILITY EXAMPLE 2

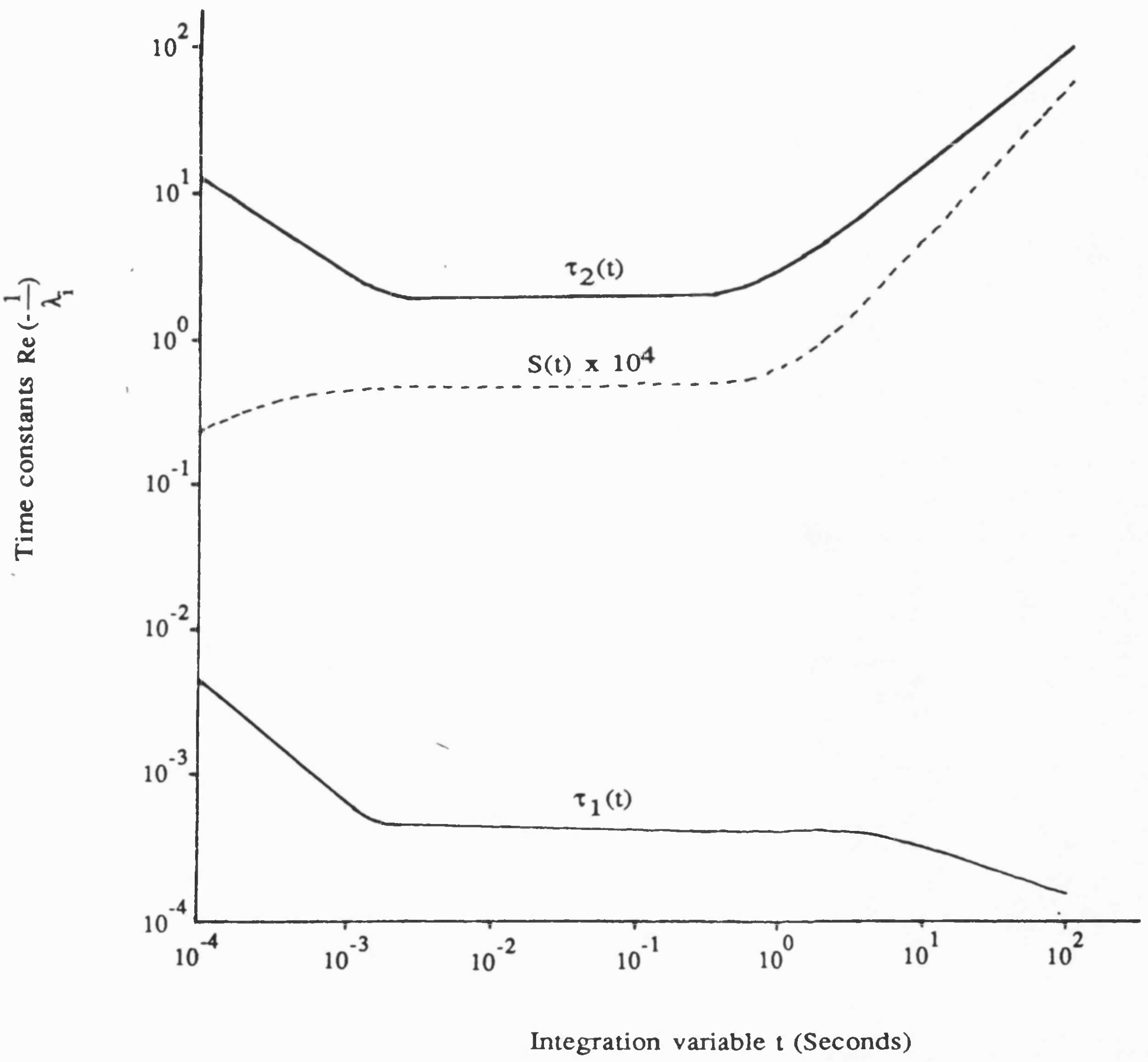


**FIGURE 4.6 STIFFNESS RATIO S(t) AND TIME CONSTANTS τ(t) OF STABILITY EXAMPLE 3**

Solid line, $\varepsilon=10^{-3}$; broken line, $\varepsilon=10^{-4}$

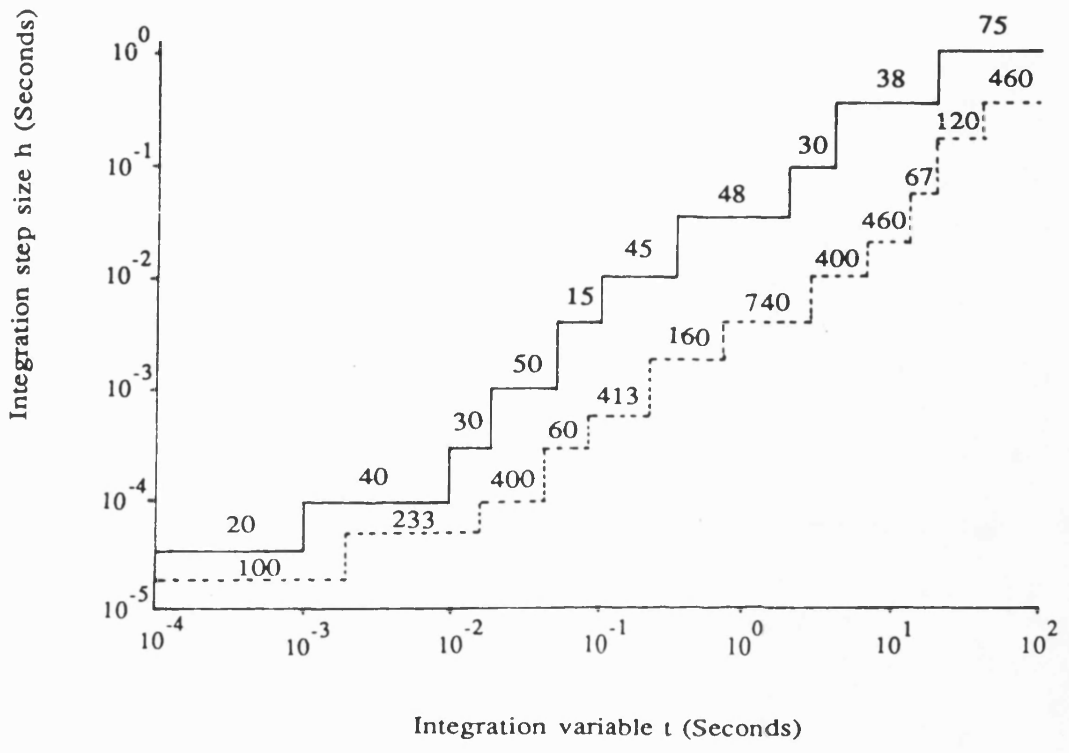


**FIGURE 4.7 VARIATION OF STEP SIZE AND NUMBER OF STEPS IN SOLUTION OF EXAMPLE 3 USING EXPLICIT METHOD**

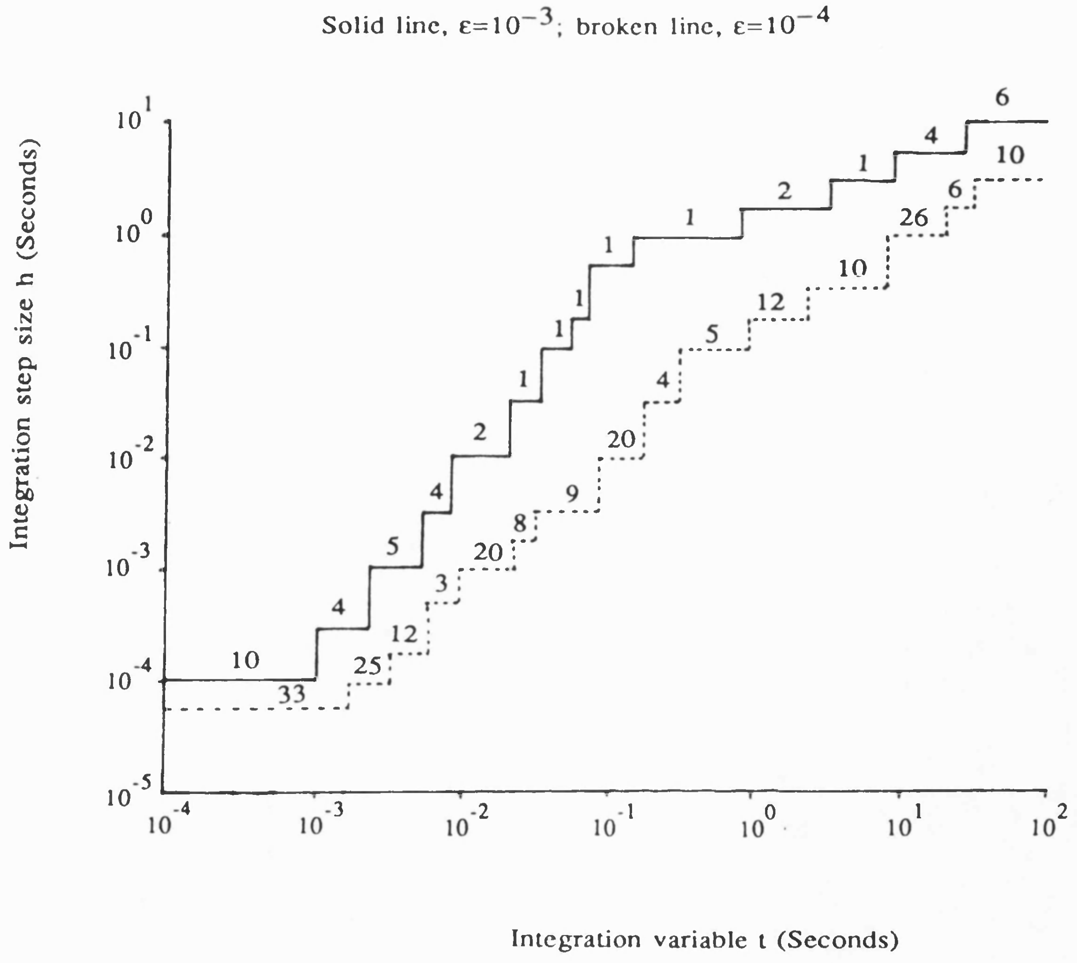


FIGURE 4.8 VARIATION OF STEP SIZE AND NUMBER OF STEPS IN SOLUTION OF EXAMPLE 3 USING IMPLICIT METHOD

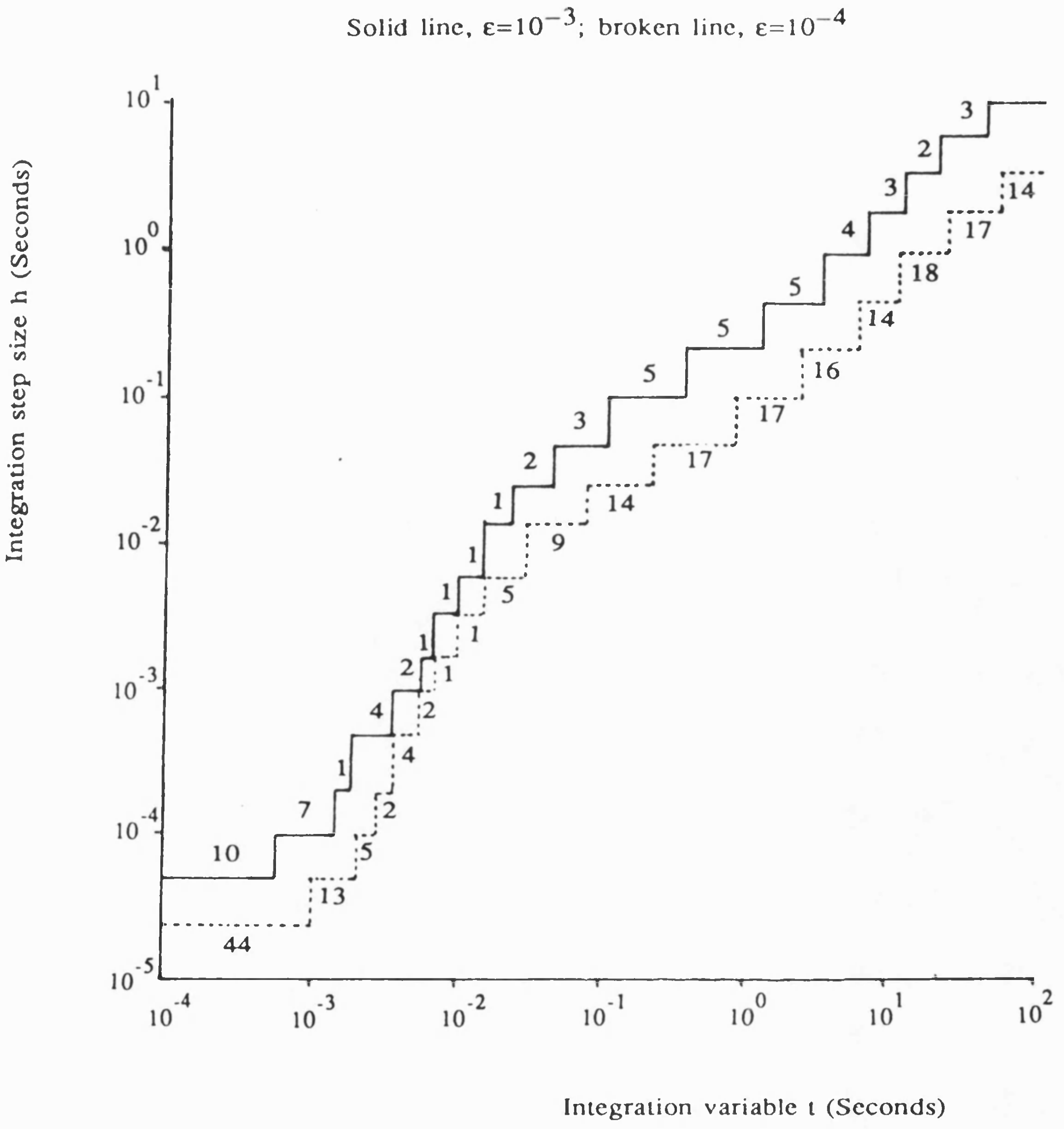


**FIGURE 4.9 VARIATION OF STEP SIZE AND NUMBER OF STEPS IN SOLUTION OF EXAMPLE 3 USING METHOD SUGGESTED IN HALL & WATT**

# CHAPTER 5

**DETAILED CONTENTS** **Page**

# CHAPTER 5

# IMPLEMENTATION AND TESTING OF THE NEW INTEGRATION METHOD WITH THE HYDRAULIC AUTOMATIC SIMULATION PACKAGE

## Introduction

**5.1** Having developed and analysed in detail the new method, the next step is to implement a generalised form of the integration routine inside the HASP package, and hence assess the behaviour of the method when it is used in this way. This chapter explains the structure and the workings of the HASP simulation package, and describes how a numerical integration method must be implemented. The generalisation of the predictor-corrector pair as a complete numerical integrator is discussed, as is the placing of the new method inside the package. The changes that must be made, to allow the use of the new method rather than Gear's method, are explained, and examples of necessary changed computer coding within HASP are given. Finally, dynamic simulations of Fluid Power circuits have been performed, and the performance of the new method with these circuits is examined, and compared with the performance of Gear's algorithm, which has previously been used by the package. The simulation results obtained by the different methods are presented and discussed.

**Description Of HASP**

**5.2** The Hydraulic Automatic Simulation Package consists of a set of computer programs, and has been developed to aid the design of hydraulic systems. The package is intended for use by engineeers who do not have a high degree of computational skill, or expertise in mathematical modelling, and consequently the package is dependent on many automatic processes.

**5.3** The package consists of a library of individual mathematical models, each written as a FORTRAN subroutine representing a discrete physical component of a Fluid Power system; together with a program generator and a numerical integration routine. An overall view of the HASP package is shown in Figure 5.1. The component models form individual blocks that are placed into position by the program generator to represent a hydraulic circuit. Having used the program generator, the user is left with a unique simulation program corresponding to a unique hydraulic circuit. The simulation programs are considered as temporary, whereas the program generator is permanent, and with the integration routine, constitutes the most important part of the package. The integration routine is also permanent, and must be able to solve any differential equations that arise from describing the behaviour of a hydraulic circuit. To describe to the reader how the HASP simulation package operates, a hydraulic circuit will be considered, and the action taken by HASP in its solution will be studied.

5.4 In order to use the package, the user must initially construct a computer block diagram of the hydraulic circuit whose behaviour is being investigated. This block diagram approach was first developed by Rolfe [71] and is representative of the way in which individual models of hydraulic components are connected together in a circuit. Considering for example the hydraulic actuator circuit shown in Figure 5.2, that has been studied in a previous chapter. A computer block diagram, indicating the individual component models necessary to represent the system behaviour, is shown in Figure 5.3. The block diagram contains boxes, each representing a model, which are connected together by lines that are termed links [4]. These links do not represent physical components such as pipes or shafts, but describe an exchange of information between models. A four character name is used for each model and is written in the box. The first two letters of the model name represent a particular class of model, the second two, a particular model in that class. For example, a pipe model will commence with the characters PI and the remaining two characters will depend on which pipe model is required. An illustration of a pipe model is shown in Figure 5.4. This receives inputs from up to eight system components on eight links and outputs the pressure which will be needed by other component models for calculation purposes. A simulation program of the hydraulic circuit can now be produced since the model information and linking shown on the block diagram is subsequently entered into a simple data file for input to the program generator.

**Program Generator**

**5.5** The program generator is the most important part of the package and exists alongside a comprehensive library of component models. Each model is an individual subroutine and the program generator automatically links together the necessary routines in order to create a unique simulation program for each individual hydraulic circuit. The program generation stage of the simulation will write a complete set of FORTRAN files, which are fully explained later in the chapter, and place them in the correct calling sequence. The only other permanent component in the completed program is the integration subroutine, which is written so as be to suitable for use with any circuit. This will be discussed in detail later in the chapter.

**5.6** Once the block diagram has been constructed, then the controlling segments of the simulation can be generated by the user. The program generator is invoked and the questions posed are answered. The questions issued by the generator are entirely concerned with the definition of the block diagram produced by the user, and the production of a file to store that information for possible retrieval, amending and regenerating in the future. Assuming that the circuit data defined by the user is acceptable to the program generator, several routines are written. Specifically, four FORTRAN source files are produced, namely: MAIN, OUT, AUX and CONTRL, which form the main segment and three controlling subroutines of the simulation program, together with a selector file which effectively instructs the component library which models are to be attached to the controlling segments. These routines are introduced later.

5.7 Again, considering the example shown as a hydraulic circuit in Figure 5.2. The first step the user must take, having selected appropriate HASP component models and formed the schematic block diagram of the circuit, is to convert the linking diagram into a table of information acceptable to the program generator. The table of data corresponding to Figure 5.3 is shown below

```
10
PU 0001  08  09  12
PM0001   12
PI  0501  02  09  10
PC 0101  10  11
TK 0301  05  08  11
DC 1T01  01  02  03  04  05
PI  0601  03  06  01
AL 0001  06  07  13  01  02
PI  0602  04  07  02
DE 0101  01
```

5.8 The first line indicates that there are ten component models in the circuit. The remaining lines list the component models in an arbitrary order and define the links between them. Each line consists of a four character mnemonic for the component model, the two digit identifier to indicate multiple occurrences of the same model, and the external links in the form of the two digit numbers separated by single blank spaces.

5.9 At this stage the program generator is employed. Since the information outlined above is defined in a simple interactive manner, the user will have the chance to correct typing and logical errors. The generator employs a sophisticated algorithm to check the validity of defined data and display diagnostic error messages if problems with the entered data are encountered. When each component model is written, the information required by that model as input is stored in a file called the 'component models attribute' file, and this is interrogated at various stages of the generation procedure in order to aid in checking the validity of the data, and also to set up the order and form of the call statements to the component model subroutines. Provided the defined data is acceptable, the four FORTRAN files and one component selector file are produced. At this point, the program generator has completed its task, and it is now necessary to link the generated segments to selected segments from the component model library.

**5.10** When the routines written by the generator have been linked together, along with the integration routine, which is discussed later, then a complete simulation program is produced. This program is run interactively, and the user responds to questions posed by the computer. When a simulation is performed for the first time, all the dimensions and performance data must be supplied for each model in turn, and this is termed the input stage. The simulation is then performed for a period of real time specified by the user, after which the results may be examined. It is also possible to view the results during a simulation. Subsequent simulations of the same circuit can be performed by the user changing data information for selected items. The program generation stage is only repeated if the user wishes to change the structure of the circuit. The reader is again referred to the main features of the HASP system shown in Figure 5.1.

## The Constituent Parts Of The HASP Simulation Package Written By The Program Generator

**5.11** The action of the four FORTRAN files and the component model selector file which the program generator automatically writes, are now described. The overall simulation program is formed when the compiled versions of these files are linked to the compiled versions of the relevant component models and the integration subroutine, and is called CAD.TSK. The four FORTRAN files, mentioned earlier, form the body of the simulation program. MAIN is the main calling program, CONTRL is for the collection and transfer of data, AUX calls the integration subroutine and component models in a determined order, and OUT deals with the output from a simulation. The programs, written by the program generator for the example introduced earlier in the chapter, can be found in Appendix C.

**5.12 Main program MAIN.** When writing well structured FORTRAN programs, it is usual practice to have a main body of code which calls the constituent parts of the program in a direct manner. MAIN is the main body of the simulation program CAD.TSK which calls the other routines controlling the individual workings of the simulation program. In particular, MAIN calls two segments, each of which recursively calls other routines. Indirectly, one of these segments, CONTRL, is used to input the dimensional and performance data values, and the other segment is the integration subroutine required to perform the simulation.

**5.13 Subroutine CONTRL.** The subroutine CONTRL is called from the main segment MAIN and this routine allows each component model parameter input section to be accessed individually in order to supply the dimensional and performance data details, which must be supplied by the user before the simulation commences. Data for each component in the hydraulic circuit being simulated are entered from the input routines of each individual model. For example, the piston area, initial displacement and velocity of the actuator are defined by the user inputting values to the actuator model input routine. The complete data for all components are then collated within CONTRL, and in turn transfered to the model calculation routines via MAIN.

**5.14 Subroutine AUX.** During a simulation, the numerical integration subroutine does not call individual component models directly. To ensure the flexibility and general application of the integrator, a separate routine is called instead. This routine is called AUX, and is written by the program generator for each hydraulic circuit that is to be simulated. AUX contains the subroutine calls to the component model calculation routines in a specified order, pre-determined by the nature of the models and their position in the circuit. AUX enables algebraic information to be exchanged between component models, derivatives of state variables to be transmitted from models to integrator, and state variables to be transmitted from the integrator to models requiring them. An integer flag, called LIMIT, which is explained in detail later, also operates inside of AUX, and this is passed to and from the integrator and the component models. Its values can change the action of AUX.

**5.15 Subroutine OUT.** OUT is a routine which is used to store the simulation results. The results are automatically stored in a file called CADRES.DAT, and this file can be accessed by the user via HASP to obtain graphical results, or alternatively a numerical printout during the simulation. The graphics routine, although simple, does provide the basic information generally required by the user. The numerical output, in contrast, is particularly useful for a detailed examination of the results.

**5.16 Component model selector file CAD.OPT.** Since MAIN calls either the model input routines via CONTRL, or the model calculation routines via the integration routine and AUX, a file has been written which automatically selects the required component model subroutines from the library when the linking process takes place to form the input and simulation portions of the simulation program. The program generator writes a unique CAD.OPT for each individual simulation, and this file is used to link the constituent parts that the generator has written, to the relevant routines from the model library and the integration routine.

**Numerical Integrator Inside HASP**

**5.17** Figure 5.5 shows a schematic diagram of the integration process inside HASP. The integration method requires information from AUX, and in the case of a classical integration method, this will be in the form of state variable derivatives. When the numerical solution has been found at the next time step, or if more information is required, the integrator passes the latest state variable values back to AUX which then calls the model calculation subroutines in an appropriate order so as to collect new information for the integrator, and hence the process is repeated.

**5.18** When Gear's method is called by MAIN, then the call statement is:

```
CALL GEARKC(X,XEND,Y,N,AUX,TOL,TEST,TAB,OUT,F,DFDY,DIF,IFAIL)
```

Gear's method was developed by C.W. Gear [63], and was adapted for use with the HASP package by K. Caney and W. Richards. It was devised specifically for solving stiff sets of differential equations. The integration method, which is variable order and variable step, effectively chooses the order of the method dependent on the complexity and stiffness of the problem being solved. The method employs a Newton-Raphson iteration scheme in the solution of its implicit corrector equations, but does not evaluate a new Jacobian at each time step provided the iterates converge within three iterations. Consequently, the method is efficient in its re-evaluation of this matrix. However Gear's method is not suitable for solving discontinuous problems [72], since it is a multi-step method, and special simulation techniques are employed at the points of discontinuity in the form of cubic smoothing. An explanation of cubic smoothing is given later in the chapter, and a full description of the action of Gear's method inside HASP is given by Tomlinson [4].

**5.19** Here, a detailed explanation of the variables in the argument list of the call statement to GEARKC, and their action in the integrator, or in AUX, is given. These arguments are important since they are recognised by the models and the generated FORTRAN subroutines, and must be employed by each integration method that is implemented inside HASP.

| | |
|---|---|
| X | - the time in seconds that the simulation has reached |
| XEND | - overall simulation time defined by the user in seconds |
| Y | - array that stores the state variables. Its size is dependent on the number of system equations |
| N | - the number of state variables |
| TOL | - the tolerance used for the time step control, and the convergence criteria with the iteration scheme. |
| TEST | - the variable used to define the type of error test that is used with the iteration scheme. |
| TAB | - The print interval of the results in seconds |
| F, DIF | - Dummy arguments used in the evaluation of the Jacobian |
| DFDY | - contains the elements of the Jacobian |
| IFAIL | - integer used to indicate a failure inside of the integrator |

**5.21 Integer LIMIT.** Besides these arguments, there is another important integer variable called LIMIT. This variable aids in testing to ensure that no physical conditions have been violated by the integrator. Once an integration step has been completed by the integrator, in that both the convergence requirement and the local error criteria have been satisfied, one further test is necessary before the step can be accepted. The integrator is not able to decide whether it has violated a physical condition, such as when an actuator apparently extends beyond the limit of its travel, and so the models must check that no physical condition has been violated. LIMIT is used by the integrator in order to communicate with the component model subroutines, and takes several possible values depending on the stage of the integration. At the beginning of the simulation LIMIT is set to 0; during a prediction or correction stage, LIMIT is set to 1, and at the completion of an integration stage, LIMIT is set to 2. Having been set to 2, a final call is made to each of the model subroutines via AUX in order to ensure that no physical condition has been violated. If there is a violation, the integration step is rejected and LIMIT is set to 3 by the model that detects the violation. When LIMIT returns from the models, through AUX, and back to the integrator, and the integrator detects that the value of LIMIT is 3, then the integration step is repeated with the integration step length halved.

## Generalisation And Implementation Of The New Method

**5.22** The remainder of the chapter deals with the writing of the new method in a form suitable for use inside the HASP package. This will include the formulation of a Jacobian matrix for the iteration scheme using a perturbation technique and other modifications that must be made to the method so as to ensure its trouble-free implementation. Besides the changes that must be made to the method, components of the HASP package must also be adapted for use with the new integrator. These adaptations, to the program generator and to the models, are discussed, and the resultant routines that are produced are presented.

**5.23 Perturbation technique for formulating the Jacobian matrix.** When applying the corrector form of the new method, the Newton iteration scheme that is needed to solve it will require the formulation of a Jacobian matrix. Unlike the examples found in chapter 3, where specific Jacobian matrices were evaluated for each of the problems, it is necessary to devise a means for approximating, in general, a Jacobian matrix regardless of the particular problem. The process that has been used to do this is derived from the first principles definition of a partial derivative, and this is now described

**5.24** When applying the corrector, the form of the solution is given by the numerical scheme:

$$y_{in+1} = \frac{b_{in+1}}{a_{in+1}} \left( e^{a_{in+1}h} - 1 \right) + y_{in} e^{a_{in+1}h} \qquad i = 1,2,\ldots,N \qquad (5.1)$$

where N is the number of differential equations being solved. In matrix form, this is given by:

$$\underline{y}_{n+1} = D_1 \underline{y}_n + D_2 \underline{z} \qquad (5.2)$$

where

$$\underline{y} = \left[y_1, y_2, \ldots, y_N\right]^T, \quad \underline{z} = \left[\frac{b_{1n+1}}{a_{1n+1}}, \frac{b_{2n+1}}{a_{2n+1}}, \ldots, \frac{b_{Nn+1}}{a_{Nn+1}}\right]^T$$

$$D_1 = \begin{bmatrix} e^{a_{1n+1}h} & 0 & 0 & \ldots & 0 \\ 0 & e^{a_{2n+1}h} & 0 & \ldots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \ldots & e^{a_{Nn+1}h} \end{bmatrix}$$

$$D_2 = \begin{bmatrix} e^{a_{1n+1}h}-1 & 0 & 0 & \ldots & 0 \\ 0 & e^{a_{2n+1}h}-1 & 0 & \ldots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \ldots & e^{a_{Nn+1}h}-1 \end{bmatrix}$$

Forming $\underline{F}(\underline{y}) = \underline{0}$, then the Newton iterations will be given by:

$$\underline{y}_{n+1}^{(s+1)} = \underline{y}_{n+1}^{(s)} - [\underline{F}'(\underline{y}_{n+1}^{(s)})]^{-1} [\underline{F}(\underline{y}_{n+1}^{(s)})] \tag{5.3}$$

Consequently, the Jacobian of $\underline{F}(\underline{y})$ is required, and this will be

$$I - D_2 \frac{\partial \underline{z}}{\partial \underline{y}} \tag{5.4}$$

Defining

$$X_{ij} = \partial\left(\frac{b_{in+1}}{a_{in+1}}\right)/\partial y_j, \text{ evaluated at the point } (t_{n+1}, \underline{y}_{n+1}^{(s)})$$

then the full Jacobian will become:

$$\begin{bmatrix} 1-(e^{a_{1n+1}h}-1)X_{11} & (1-e^{a_{1n+1}h})X_{12} & \cdots & (1-e^{a_{1n+1}h})X_{1N} \\ (1-e^{a_{2n+1}h})X_{21} & 1-(e^{a_{2n+1}h}-1)X_{22} & \cdots & (1-e^{a_{2n+1}h})X_{2N} \\ \cdot & \cdot & \cdot\cdot\cdot & \cdot \\ \cdot & \cdot & \cdot\cdot\cdot & \cdot \\ \cdot & \cdot & \cdot\cdot\cdot & \cdot \\ (1-e^{a_{Nn+1}h})X_{N1} & (1-e^{a_{Nn+1}h})X_{N2} & \cdots & 1-(e^{a_{Nn+1}h}-1)X_{NN} \end{bmatrix}$$

To form estimates to $X_{ij}$ it is necessary to return to the standard definition of a partial derivative.

**5.25** For the function $f_1(t_1,t_2,t_3,\dots,t_n)$, then the partial derivative in the $t_1$ direction is defined as:

$$\frac{\partial f_1}{\partial t_1} = \lim_{h \to 0} \frac{f_1(t_1+h,t_2,t_3,\dots,t_n) - f_1(t_1,t_2,t_3,\dots,t_n)}{h}$$

and in general, the partial derivatives for this function are defined as

$$\frac{\partial f_1}{\partial t_i} = \lim_{h \to 0} \frac{f_1(t_1,t_2,t_3,\dots,t_i+h,\dots,t_n) - f_1(t_1,t_2,t_3,\dots,t_n)}{h}$$

The way in which numerical integration methods form an estimate to these derivatives is to perturb the variable, with respect to which the partial derivative is required, and then evaluate the new function value with this perturbed variable. From this is subtracted the original value of the function, and the resultant is divided by the perturbation made, i.e. to form an approximation to the partial derivative $\partial f_1/\partial t_1$, then the process would be:

form

$$f_1(\text{new}) = f_1(t_1+kt_1,t_2,t_3,\dots,t_n)$$

$$f_1(\text{old}) = f_1(t_1,t_2,t_3,\dots,t_n)$$

then

$$\frac{\partial f_1}{\partial t_1} \approx \frac{f_1(\text{new}) - f_1(\text{old})}{(t_1+kt_1) - t_1} = \frac{f_1(\text{new}) - f_1(\text{old})}{kt_1}$$

and consequently, provided the perturbation is small enough, then a reliable estimation to the derivative can be formed.

**5.26** For the Jacobian matrix, the partial derivatives are formed in a similar way to that described in the last section. Since both $a_i$ and $b_i$, $i=1,2,\dots,N$, are functions of $y_i$, then to form the $X_{ij}$ it is necessary to perturb the $y_j$ and then re-evaluate the function

$$\frac{b_{in+1}}{a_{in+1}}$$

In full, to formulate the partial derivative, $X_{ij}$, then the process is to find

In full, to formulate the partial derivative, $X_{ij}$, then the process is to find

$$\frac{b_{in+1}(\text{new})}{a_{in+1}(\text{new})} = \frac{b_{in+1}(y_1,y_2,\dots,y_j+ky_j,\dots,y_N)}{a_{in+1}(y_1,y_2,\dots,y_j+ky_j,\dots,y_N)}$$

an

$$\frac{b_{in+1}(\text{old})}{a_{in+1}(\text{old})} = \frac{b_{in+1}(y_1,y_2,\dots,y_N)}{a_{in+1}(y_1,y_2,\dots,y_N)}$$

from which can be formulated $X_{ij}$, since

$$X_{ij} = \frac{\dfrac{b_{in+1}(\text{new})}{a_{in+1}(\text{new})} - \dfrac{b_{in+1}(\text{old})}{a_{in+1}(\text{old})}}{ky_j}$$

The value of k must be chosen carefully to ensure a sufficiently accurate estimation to the derivative.

5.27 The general integration algorithm for the new method is called from within MAIN. The call statement is:

```
CALL FOM (T,TEND,Y,N,AUX,TOL,ITEST,TAB,OUT,IFAIL)
```

The arguments are identical to those described in section 5.20, although DIF, DFDY and F have been excluded as they were not deemed to be necessary in the argument list. Appendix C gives a complete documented listing and description of the generalised new method, and the following text describes the main considerations in the implementation of the method. When Gear's method makes a call to AUX, then the calling statement is of the form

```
CALL AUX (F,Y,T,N,LIMIT)
```

since the function values given by F are required to form the solution at the next time step. However, for the new method, it is not the function values that are required, but coefficient values, which is explained in section 3.12. Consequently, from within the subroutine FOM, the call statements made to AUX are given by

```
      CALL AUX (AAA,Y,T,N,LIMIT)
```

and the coefficient values can be passed back to the integrator from AUX in the array AAA(2N). This process is explained in detail later in the chapter, when the modifications that must be made to the program generator for the implementation of the method are discussed.

**5.28** The following piece of code illustrates the formulation of a Jacobian matrix, when the coefficient values can be passed through AUX to the integrator

```
      LIMIT = 1
      DO 10 I = 1,N
         YY(I) = Y1(I) + EPS*Y1(I)
         CALL AUX(BBB,YY,T,N,LIMIT)
         A1(I)  = BBB(I)
         B1(I)  = BBB(N+I)
         YY(I) = Y1(I)
10    CONTINUE
```

This piece of code perturbs the predicted state variable values, Y1(N), and then calls AUX to determine the new coefficient values for each perturbed state variable. The array BBB(2N) will hold all the coefficient values on its return from AUX, and these are distributed to A1(N) and B1(N). The next piece of code is :

```
      DO 20 I = 1,N
         DIVIS1 = B(I)/A(I)
         DIVIS2 = B1(I)/A1(I)
         DERIV(I) = (DIVIS2 - DIVIS1)/(EPS*Y1(I))
20    CONTINUE
```

The evaluated Jacobian is stored in DERIV(N). Precautions are taken, which this piece of code does not illustrate, to ensure that no divisions by zero are attempted, e.g. when A(I) = 0. These can be seen in the fully commented version of the program given in Appendix C, which also gives a complete description of the program action.

**5.29 Linear interpolation.** A linear interpolation process is employed by the subroutine implementing the new method, to ensure that no information is lost, whatever the value of the print interval input by the user. Since the method is variable step, it is unlikely that at each time step the differential equations have been integrated exactly on the print interval. Consequently, the process that is employed by GEARKC, and by the new method, is a decision making piece of code that determines when results should be transfered to OUT, and if this is not exactly on a print interval, then linear interpolation is employed to determine the result at the stipulated print interval point. The following algorithm further illustrates this idea.

The method has completed another integration step. The print interval is given by TAB, the time when the results should be outputted is given by TTAB, and the time up to which the integration has been completed is given by T. The final simulation time is given by TEND

.

```
IF ( T = TTAB) THEN
    OUTPUT RESULTS VIA HASP ROUTINE OUT
    TTAB = TTAB + TAB
ENDIF
IF ( T > TTAB) THEN
    USE LINEAR INTERPOLATION TO FORM AN ESTIMATE TO
    THE SOLUTION AT TTAB
    OUTPUT RESULTS AT TTAB VIA HASP ROUTINE OUT
    TTAB = TTAB + TAB
ENDIF
IF ( T .GE. TEND) THEN STOP
ELSE CONTINUE WITH INTEGRATION
```

**5.30 Modifications to the models and program generator.** Once the method had been written in a general form suitable for use with HASP, its implementation inside the package was relatively trouble-free. The problems that remained were in the modification of the program generator and the models. The program generator must be modified to ensure that when the new method is used, rather than Gear's method, to solve a problem, then the necessary changes to the generated FORTRAN files, and the program selector file are made automatically. The changes to the models are made so that the necessary coefficient values are supplied to the integrator rather than derivative values. A flow chart describing the general structure of the algorithm adopted in implementing the new method is shown in Figure 5.6. This is given to complement the structure of the integration process in a typical HASP simulation program which is shown in Figure 5.7. The component model calculation routines, shown in Figure 5.7, interfaced with AUX, are determined by the hydraulic circuit that is being simulated.

**5.31** To illustrate best the way in which information must be passed from models to the new integrator, the example shown in Figure 5.2, for which the block computer diagram appears in Figure 5.3, is again considered. After a brief description of the mathematical models involved in the simulation of this circuit, the actuator model will be isolated, and the formulation of the coefficient values for the new method from this model explained.

**5.32** The mathematical models used in the circuit are as follows:

TK03 - represents the three port hydraulic tank. It supplies a user-defined pressure to the adjacent models which is independent of both flow rate and tank volume

PM00 - represents the prime mover. It supplies a user-defined angular velocity to the pump. The effects of speed droop, maximum torque and motoring are ignored

PU00 - represents the fixed displacement hydraulic pump. The analysis of the behaviour of the pump is based upon the Wilson model [3]. Pump dynamics are ignored

PC01 - represents the supply system relief valve. The model supplies flowrate to the adjacent models based upon a user-defined pressure/flow characteristic. The characteristic is assumed to be linear, and leakage, saturation and valve dynamics are ignored

|  |  |
|---|---|
| DE01 | - valve controller which defines the fractional displacement of the directional control valve DC1T |
| DC1T | - represents a 4-way, 3-position directional control valve. The model also incorporates the orifices on the supply and feed ports of the directional control valve |
| PI05 | - represents the dynamic response of a frictionless pipe. The model also accounts for the effects of air and cavitation |
| PI0601 | - represents the length of pipe connecting the directional control valve to the linear actuator. This dynamic model accounts for the effects of air release and cavitation, and also accounts for volume change signalled by the movement of the actuator |
| PI0602 | - represents the length of pipe returning from the actuator to the directional control valve |
| AL00 | - represents a general purpose linear actuator model |

**5.33** When the hydraulic circuit is modelled, five first order ordinary differential equations are formed for the integrator to solve. This corresponds to five state variables and subsequently, the five derivative values are produced by the three pipe models and the actuator model. In the implementation of the new method, the output from these models must be reformed into the coefficient values that the method requires. The linear actuator is now considered and the necessary changes discussed.

**5.34 Linear actuator model.** The linear actuator model has been developed in order to simulate the behaviour of a linear actuator with a constant load, and has provision for end stops. The actuator may be inclined at any angle between plus and minus 90 degrees, and a schematic diagram of a linear actuator with load is shown in Figure 5.8. The model requires the solution of a second order ordinary differential equation representing the acceleration of the actuator. Consequently, two state variables are computed, one for actuator displacement, and the other for velocity. The model makes use of two signals to provide output of incremented volume change to pipes connected to the piston and rod ends of the actuator.

5.35 The equations that govern the behaviour of the actuator are given by:

$$\frac{du}{dt} = \frac{1}{M}\left( P_{in}A_1 - P_{out}A_2 - fu - f_w u^2 - mgsin\theta - kx - F_c sgn(u) \right) \qquad (5.5)$$

$$\frac{dx}{dt} = u \qquad (5.6)$$

where:

| | | |
|---|---|---|
| $P_{in}$ | - | fluid pressure at piston end |
| $P_{out}$ | - | fluid pressure at rod end |
| $A_1$ | - | piston end area |
| $A_2$ | - | rod end area |
| $f$ | - | viscous friction coefficient |
| $f_w$ | - | windage loss coefficient |
| $u$ | - | velocity of piston |
| $M$ | - | mass of piston and load |
| $\theta$ | – | angle of inclination of the actuator to the vertical |
| $k$ | - | spring stiffness |
| $x$ | - | displacement of piston |
| $F_c$ | - | coulomb friction force |

The equations of motion given here do not clarify the effects of stiction or explain the end stop logic used in the model, but they are sufficient to explain how the information from the actuator is passed to the new integrator.

**5.36 Changes to the linear actuator model.** For a classical integration method, such as Gear's, the values of the derivatives given in equations (5.5) and (5.6) are passed back to the method. Rewriting these equations into a form suitable for the new method will give:

$$\frac{du}{dt} = \frac{1}{M}(-f - f_W u)u + \frac{1}{M}(P_{in}A_1 - P_{out}A_2 - mg\sin\theta - kx - F_c \mathrm{sgn}(u)) \tag{5.7}$$

and

$$\frac{dx}{dt} = -x + (x+u) \tag{5.8}$$

Equation (5.8) is formed using the technique described in section 3.22. Consequently, the coefficient values can be passed back to the integrator via AUX and these will be:

$$\mathrm{ACOEFF}(1) = \frac{1}{M}(-f - f_w u)$$

$$\mathrm{ACOEFF}(2) = -1$$

$$\mathrm{BCOEFF}(1) = \frac{1}{M}(P_{in}A_1 - P_{out}A_2 - mg\sin\theta - kx - F_c \mathrm{sgn}(u))$$

$$\mathrm{BCOEFF}(2) = x + u$$

and so the required information is now available for AUX to return to the new integration method.

**5.37** Similar changes must also be made to the pipe models, but since the differential equations in these models rely on information from other models, it is necessary to ensure that the relevant data is exchanged before the coefficient values can be calculated. Once the models have been modified, the next step is to change certain portions of the coding in the program generator. The changes made in the generator itself are not discussed, since the explanation of massive portions of coding make this prohibitive. The changes can be found in the directory [CAPLEN.HASP.INHASP] on the VAX 750 in the School of Engineering [73]. The resultant programs due to these changes are now discussed.

**5.38 Changes to the program generator.** When the system linking details have been entered, and the program generator has passed them as being acceptable, the user is asked which integration method is required to solve the problem. Originally, there was no choice, since the only method was Gear's method, but with the recent work carried out on the integrator, this choice has been extended to several methods. If the user has indicated that the required method is the new method, then the portion of the program generator that writes the segments MAIN, AUX, CONTRL, OUT, and CAD.OPT must ensure that the programs written allow the new method to be used. The modifications that are required have been carried out by the author, and, as a demonstration, then for the circuit that has been considered throughout this chapter, the resultant routines written by the generator for use with the new method can be found in Appendix C.

## Implications For The Implementation Of The Integrator

**5.39** The techniques used for modelling discontinuities and certain types of non-linearities are now discussed, since many Fluid Power systems demonstrate these mathematical problems, and the way in which they are dealt with is important in understanding the implications for the implementation of the integrator.

**5.40 Modelling discontinuities and non-linearities.** Since discontinuous problems provide difficulties for multi-step methods, special techniques have been developed in the component models to deal with the discontinuities that arise. Normally, at a discontinuity, a multi-step method, such as Gear's method, must stop and restart, so as not to violate the theoretical conditions that must be satisfied in order to ensure a unique solution. Unlike single-step methods, for a multi-step method, restarts are difficult since the method relies on information at previous time steps, both to form a solution at the new time step, and to control the integration time step. One way of overcoming the problem of discontinuities for multi-step methods is to "smooth" the discontinuous function, and this approach has been adopted in HASP. The technique employed in the models is to use a cubic smoothing polynomial to connect the two branches of a discontinuous function in such a manner that the function and its gradient are continuous across the region of transition. A full description of the formulation and implementation of the cubic smoothing polynomial function is given by Hull [3], but here, in illustration, a discontinuous example is considered and the method of approach adopted is discussed.

**5.41** Figure 5.9 shows the operating characteristic of a typical component model, e.g. the pressure/flow characteristic of a pressure relief valve. This type of function will create difficulties for Gear's integration method. Figure 5.10 demonstrates how the sharp corners of the function are smoothed to ensure that GEARKC can solve the problem adequately. The way in which a model must be written to encorporate cubic smoothing, and recognise when it is needed, is also discussed by Hull [3]. However it has been found by the author, and by Wang [74], that the problems which this present when writing some models exceed the usefulness of the process. Models are much easier to write when conditional IF statements in the models are used in conjunction with single-step methods to overcome the problem arising at points of discontinuity. As already discussed in chapter 2, since single-step methods are self-starting and require only information at the last time step, discontinuities present less difficulties when these methods are used to solve problems of this nature. Consequently, in terms of user friendliness, and time-saving, if the user has to write model subroutines, then to save the effort involved in including cubic smoothing logic, a single-step method should be employed by the package. As well as implementing the new method, the author has been involved with implementing Runge-Kutta methods inside HASP, these being single-step methods. Runge-Kutta methods will be discussed in some detail in chapter 8

**5.42** Besides discontinuities, there are many other models that can, in certain circumstances, provide severe difficulties to the integrator. An example is the simple orifice restrictor, where the relationship between flow rate Q, and differential pressure $\Delta P$ is given by the parabolic law:

$$Q = k_1 \sqrt{\mid \Delta P \mid} \, \mathrm{sgn}(\Delta P)$$

The gradient of this function is given by:

$$\frac{dQ}{d\Delta P} = 0.5(\mid \Delta P \mid)^{-0.5}$$

Hence, when the differential pressure is zero, the gradient of this function is infinite, and as the orifice usually connects two sections of pipe, zero differential pressure can rise to infinite stiffness. Since no integration method can cope with such a condition, much work has been done to find the best way of overcoming this particular problem.

**5.43** One way of overcoming the problem of the infinite gradient at the origin for the orifice law is achieved using a practical engineering consideration. The flow through the orifice is assumed to be potential flow, and as in practice, the relationship is laminar in the region of the origin, then a linear variation of flow rate with differential pressure is used. Figure 5.11 shows graphically the representation of this model, and the only problem remaining is to determine the interval each side of the origin in which to apply the linearisation assumption.

**5.44** Another way of approaching this difficulty has been studied by Bowns and Rolfe [75], and has proved to be successful. Their method has been to use an iterative technique to solve the continuity differential equation for the pressures in the sections of pipe either side of the orifice, in addition to using the integrator to solve the system differential equations. This technique is similar to an approach adopted by Bowns and Wang [74], which has been found to be particularly useful for solving stiff problems.

## Application Of The New Method Inside HASP

**5.45** Having placed the new method successfully in the package, the results obtained by the method when applied to two test circuit are now presented. The first of the test circuits is the circuit shown in Figure 5.2, which has already been discussed in chapter 3, and the second of the circuits is shown in Figure 5.12, with the computer block diagram for the problem being shown in Figure 5.13. A steady flow is fed through an orifice into the actuator from a fixed displacement pump, and the return flow to tank from the actuator is also made via an orifice. Here again, five state variables occur in the simulation. The data values for the second of the two problems are presented in the same form as to the user of the HASP package, and the data values for both problems are given in tables 5.1, 5.2, 5.3 and 5.4. It was felt that these two circuits would provide a good test for the new method, since they both lead to stiff, osillatory and non-linear problems

**5.46 Results for test circuit 1.** Typical results for this problem found by the new method being tested are given in Figures 5.14 and 5.15. As a comparison, the same results given by Gear's method are shown in Figures 5.16 and 5.17. These results do not differ to any noticeable extent from the results given by the new method. One important factor in comparing the two methods is the C.P.U. time taken for the simulation of the problems, and the comparisons in C.P.U. time between the two methods are shown in table 5.5. The problem has also been solved using the Runge-Kutta Merson method, and the results were the same as those given by the new method, with the C.P.U. time required to solve the problem being similar to that required by the new method.

**5.47 Results for test circuit 2.** Typical results for this problem given by the new method are shown in Figures 5.18 and 5.19. The comparative results given by Gear's method in the solution of the same problem are shown in Figures 5.20 and 5.21, and prove to be identical to those given by the new method being tested. The C.P.U. time taken by the two methods in solving the problem is shown in table 5.5. The new method does not prove to be as fast as Gear's method for solving this problem, or the last problem, and considering the modifications that must be made to the component models to allow the use of the first order method, it is worthwhile considering an alternative form of testing the new method to study its potential as a stiff solver.

## Conclusions

**5.48** Now that the new method has been implemented inside of the HASP package, its performance in solving the Test problems, and other circuits that have been presented to it, has been good. However, as regards the C.P.U. time taken, the savings that were originally hoped for have not been made, and the main reason for this is the low order of accuracy of the method. When solving stiff or oscillatory sets of equations, then the method performs as well as, if not better than, Gear's method; but when the solution has reached steady state, then the new method is not able to compete with the fifth or sixth order method, and the accompanying large time steps, that Gear uses. Another point to be made concerns the number of function calls and Jacobian evaluations made by each of the methods. These are important since they can highlight the difficulties that a method has with solving a problem, since, for example, continued Jacobian evaluations may be a result of numerical instability. However, no measurement of these processes has been made in the solution of a problem. Consequently, to finally test the methods suitability for solving stiff sets of differential equations, it was decided to employ a package developed specifically for that reason. The package has been developed by W.H. Enright and J.D. Pryce [17], and its principal objective is to provide testing tools which can be used in assessing the efficiency and reliability of a numerical method, without requiring modifications to the method, and without the tools themselves affecting the performance of the method. The work involved in using this package is presented in the next chapter. Work has also been undertaken to improve the order of accuracy of the new method, and this is presented in chapter 7.

| Symbol | Description | Value |
|---|---|---|
| $A_1$ | Area of the actuator piston - piston side | $2.03 \times 10^{-3} m^2$ |
| $A_2$ | Area of the actuator piston - rod side | $1.54 \times 10^{-3} m^2$ |
| $B_1$ | Bulk modulus of the hydraulic oil in pipe 1 | $1.2 \times 10^9$ N/m$^2$ |
| $B_2$ | Bulk modulus of the hydraulic oil in pipe 1 | $7 \times 10^8$ N/m$^2$ |
| $B_3$ | Bulk modulus of the hydraulic oil in pipe 1 | $7 \times 10^8$ N/m$^2$ |
| f | Viscous friction coefficient | $5 \times 10^3$N/m/sec |
| $kE_1$ | discharge coefficient for d.c.v. | $4.24 \times 10^{-7}$ m$^3$/sec/N/m$^2$ |
| $kE_2$ | discharge coefficient for d.c.v. | $4.30 \times 10^{-7}$ m$^3$/sec/N/m$^2$ |
| M | Mass to be moved by the actuator | $8.7 \times 10^2$ Kg |
| $v_1$ | Volume of pipe 1 | $1.7 \times 10^{-3} m^3$ |
| $v_2$ | Initial combined volume of actuator and pipe 2 | $4.62 \times 10^{-4} m^3$ |
| $v_3$ | Initial combined volume of actuator and pipe 3 | $1.94 \times 10^{-3} m^3$ |

TABLE 5.1 DATA VALUES USED FOR EXAMPLE 1

| Symbol | Description | Value |
|---|---|---|
| $P_1$ | Oil pressure in pipe 1 - initial value | $7 \times 10^6$ N/m$^2$ |
| $P_2$ | Oil pressure in pipe 2 - initial value | $1.7 \times 10^6$ N/m$^2$ |
| $P_3$ | Oil pressure in pipe 3 - initial value | $2.1 \times 10^6$ N/m$^2$ |
| x | actuator displacement - initial value | 0 m |
| u | actuator velocity - initial value | 0 m/sec |
| $\omega$ | Angular speed of the pump | $1.57 \times 10^2$ rads/sec |
| D | Pump displacement | $3.5 \times 10^{-6}$ m$^3$/rad |
| $\mu$ | Viscosity of the hydraulic oil | $6.974 \times 10^{-2}$ Nsec/m$^2$ |
| $C_A$ | Slip loss due to differential pressure | $7 \times 10^{-9}$ |
| n | Polytropic index of air | 1.4 |
| $d_0$ | Fraction of air dissolved in the hydraulic fluid at S.T.P | 0.1 |
| $k_r$ | Relief valve coefficient | $5 \times 10^{-11}$ m$^3$/sec/N/m$^2$ |
| $P_c$ | Relief valve cracking pressure | $7 \times 10^6$ N/m$^2$ |

**TABLE 5.2 ADDITIONAL DATA VALUES FOR EXAMPLE 1**

**COMPONENT NUMBER 1**

EFFORT DUTY CYCLE (DEDT) NUMBER 1

| | |
|---|---|
| 1 OUTPUT VARIABLE IS PRESSURE | |
| 2 NUMBER OF STAGES TO DUTY CYCLE = | 1 |
| 3 STAGE 1 OUTPUT LEVEL AT BEGINNING OF STAGE = | 2.1000D+02 |
| 4 STAGE 1 OUTPUT LEVEL AT END OF STAGE = | 2.1000D+02 |
| 5 STAGE 1 TIME IN SECONDS AT WHICH STAGE ENDS = | 1.0000D+02 |
| 6 TRANSITION INTERVAL = | 1.0000D-02 |

**COMPONENT NUMBER 2**

ORIFICE RESTRICTOR (OR3Z) NUMBER 1

| | |
|---|---|
| 1 STATED FLOW IN L/S = | 8.3333D+00 |
| 2 CORRESPONDING PRESSURE DROP IN BAR = | 7.0000D+01 |
| 3 LAMINAR BOUNDARY IN BAR = | 1.0000D+00 |

**COMPONENT NUMBER 3**

FRICTIONLESS PIPE (PI06) NUMBER 1

| | |
|---|---|
| 1 INTERNAL DIAMETER OF PIPE IN MM = | 2.5230D+01 |
| 2 LENGTH OF PIPE IN M = | 1.0000D+00 |
| 3 PIPE VOLUME IN LITRES = | 4.9995D-01 |
| 4 BULK MODULUS OF PIPE SYSTEM IN BAR = | 1.8000D+04 |
| 5 AIR SATURATION PRESSURE IN BAR = | 0.0000D+00 |
| 6 PROPORTION OF DISSOLVED AIR = | 1.0000D-01 |
| 7 INITIAL PRESSURE IN BAR = | 0.0000D+00 |

**COMPONENT NUMBER 4**

LINEAR ACTUATOR (AL00) NUMBER 1

| | |
|---|---|
| 1 ACTUATOR DIAMETER IN MM = | 5.0800D+01 |
| 2 ROD DIAMETER IN MM = | 2.5000D+01 |
| 3 TOTAL MASS MOVED IN KG = | 1.0000D+03 |
| 4 STICTION IN N = | 2.0000D+02 |
| 5 COULOMB FRICTION IN N = | 1.0000D+02 |
| 6 VISCOUS FRICTION COEFFICIENT IN N/<M/S> = | 4.0000D+03 |

**TABLE 5.3 DATA VALUES FOR EXAMPLE 2**

| | |
|---|---|
| 7 WINDAGE LOSS COEFFICICENT IN N/((M/S)**2) = | 0.0000D+00 |
| 8 GRAVITATIONAL FORCE IN N = | 7.0710D+03 |
| 9 ACTUATOR STROKE IN M = | 5.9520D-01 |
| 10 INITIAL DISPLACEMENT IN M = | 0.0000D+00 |
| 11 INITIAL VELOCITY IN M/S = | 0.0000D+00 |
| 12 SPRING STIFFNESS IN M/S = | 0.0000D+00 |
| 13 PISTON LEAKAGE COEFFICIENT IN <L/S>/BAR = | 0.0000D+00 |
| 14 END STOP SPRING IN N/M = | 1.0000D+05 |
| 15 DAMPING FACTOR (CRITICAL - 2) = | 1.0000D+01 |

**COMPONENT NUMBER 5**

FRICTIONLESS PIPE (PI06) NUMBER 2

| | |
|---|---|
| 1 INTERNAL DIAMETER OF PIPE IN MM = | 2.5230D+01 |
| 2 LENGTH OF PIPE IN M = | 1.0000D+00 |
| 3 PIPE VOLUME IN LITRES = | 4.9995D-01 |
| 4 BULK MODULUS OF PIPE SYSTEM IN BAR = | 1.8000D+04 |
| 5 AIR SATURATION PRESSURE IN BAR = | 0.0000D+00 |
| 6 PROPORTION OF DISSOLVED AIR = | 1.0000D-01 |
| 7 INITIAL PRESSURE IN BAR = | 0.0000D+00 |

**COMPONENT NUMBER 6**

ORIFICE RESTRICTOR (OR3Z) NUMBER 2

| | |
|---|---|
| 1 STATED FLOW IN L/S = | 8.3333D+00 |
| 2 CORRESPONDING PRESSURE DROP IN BAR = | 7.0000D+01 |
| 3 LAMINAR BOUNDARY IN BAR = | 1.0000D+00 |

**COMPONENT NUMBER 7**

FIXED LEVEL TANK (TK00) NUMBER 1

1 PRESSURE IN BAR = 0.0000D+00

**TABLE 5.4 ADDITIONAL DATA VALUES FOR EXAMPLE 2**

| | New Method | | Gear's Method | | |
|---|---|---|---|---|---|
| Example | Tolerance (secs) | C.P.U. (secs) | Tolerance (secs) | C.P.U (secs) | Maximum Stiffness Ratio |
| 1<br>t=[0,6] secs | $1 \times 10^{-4}$ | $2.72 \times 10^{2}$ | $1 \times 10^{-5}$ | $2.212 \times 10^{2}$ | $1 \times 10^{3}$ |
| 2<br>t=[0,0.2] secs | $1 \times 10^{-4}$ | $6.78 \times 10^{1}$ | $1 \times 10^{-5}$ | $4.98 \times 10^{1}$ | $1 \times 10^{3}$ |

**TABLE 5.5 C.P.U. TIMES FOR THE NEW METHOD INSIDE OF HASP**

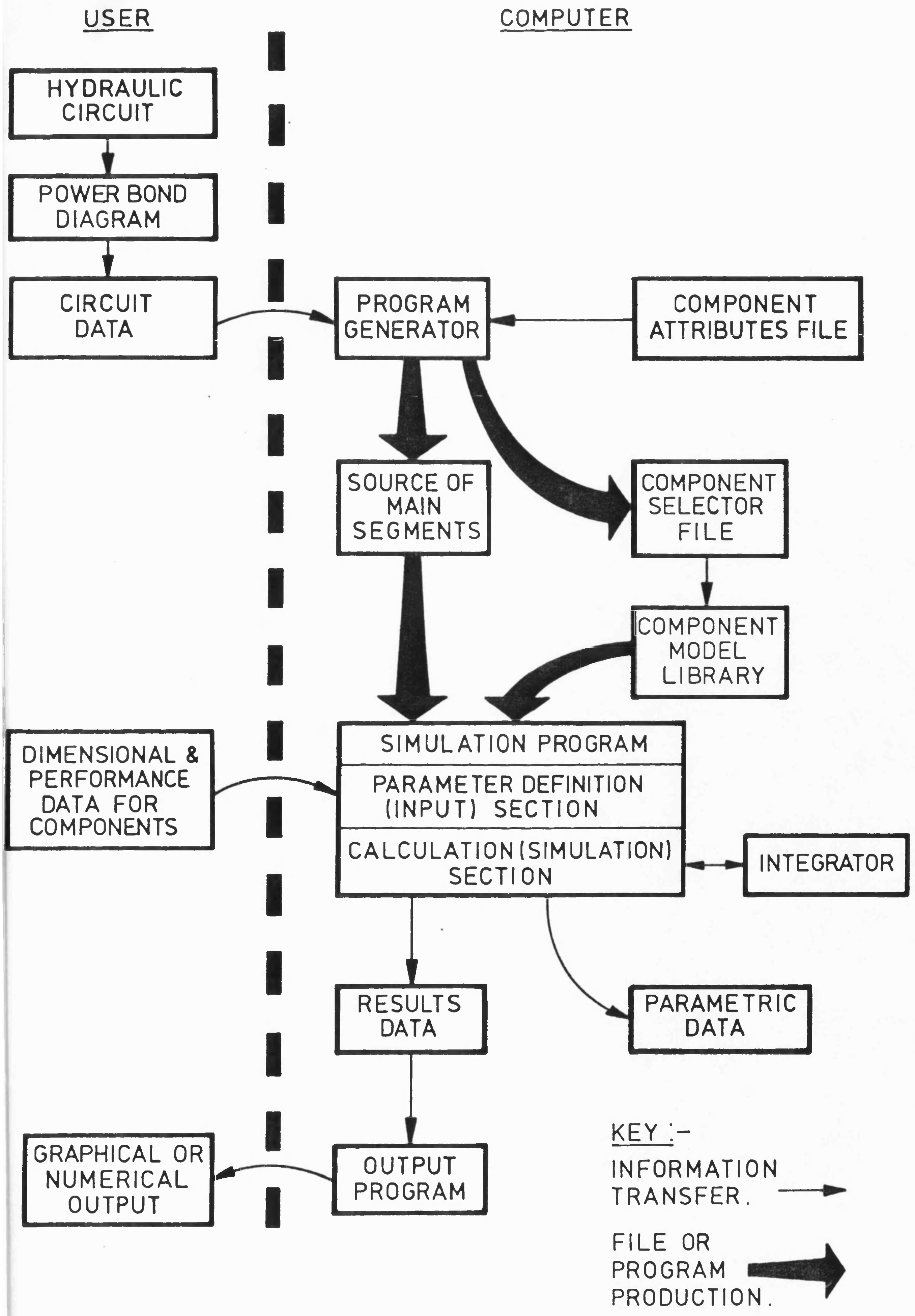


**FIGURE 5.1 THE STRUCTURE OF HASP**

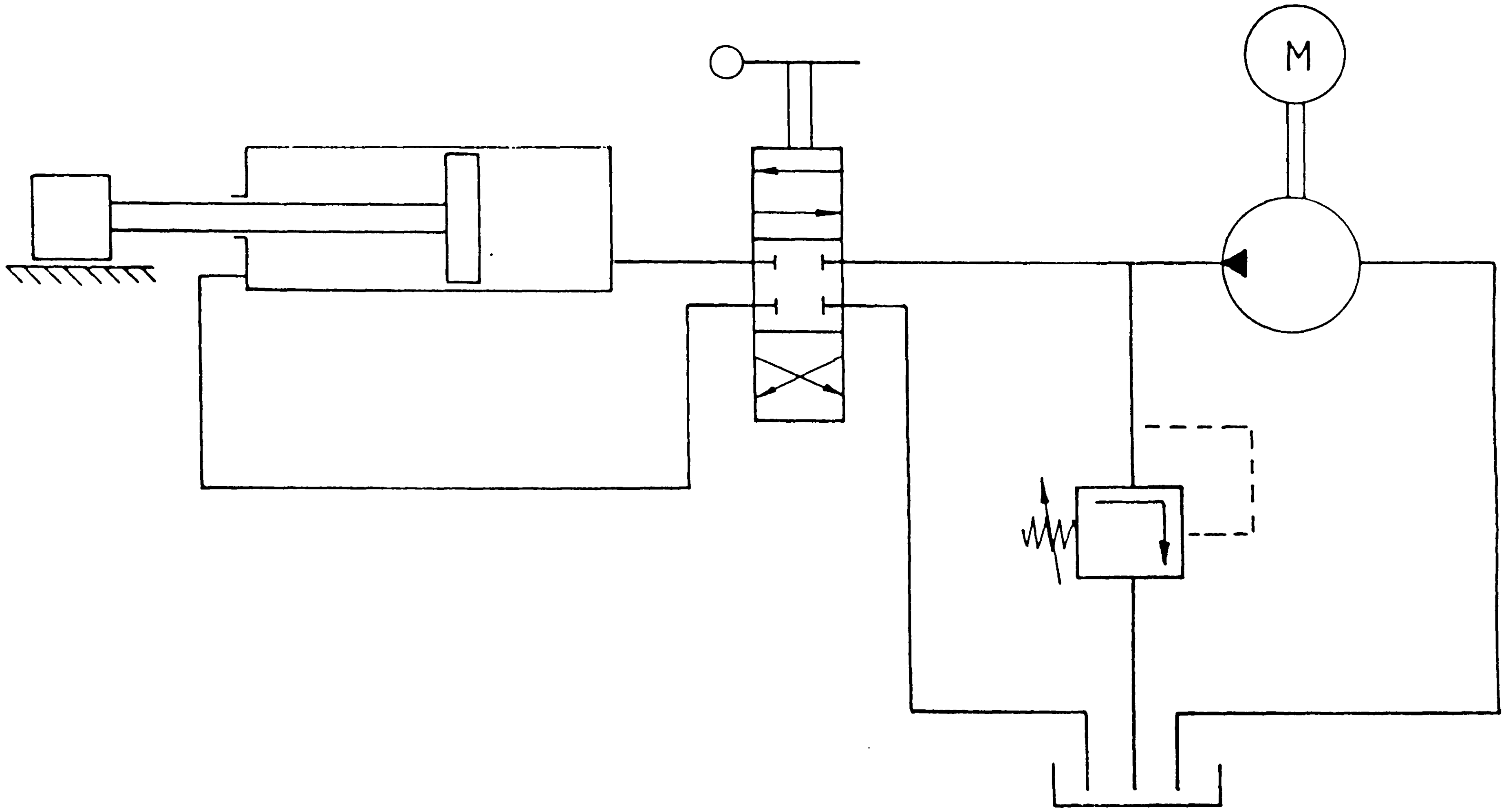


FIGURE 5.2 HYDRAULIC ACTUATOR CIRCUIT

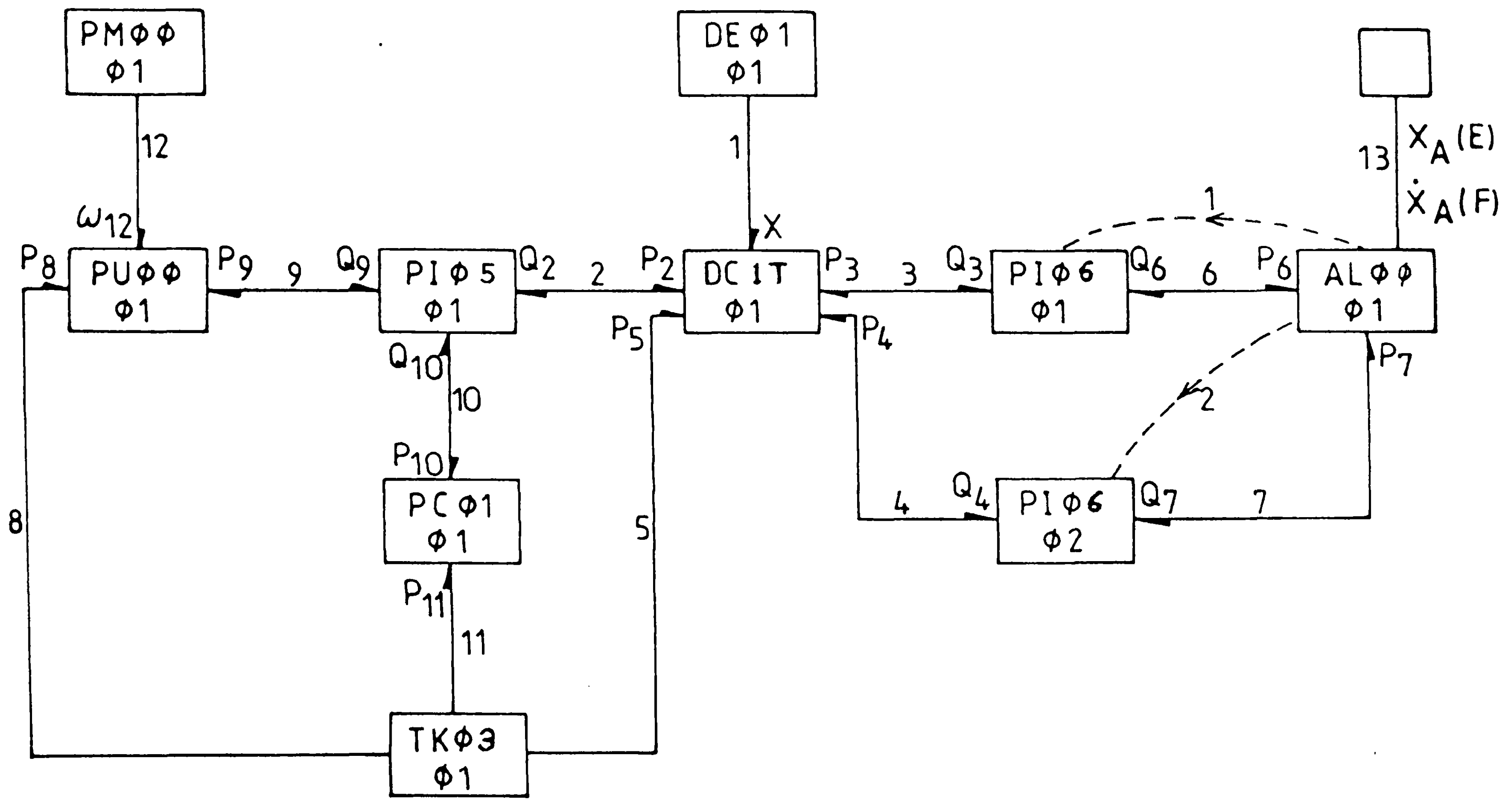


FIGURE 5.3 COMPUTER BLOCK DIAGRAM OF ACTUATOR CIRCUIT

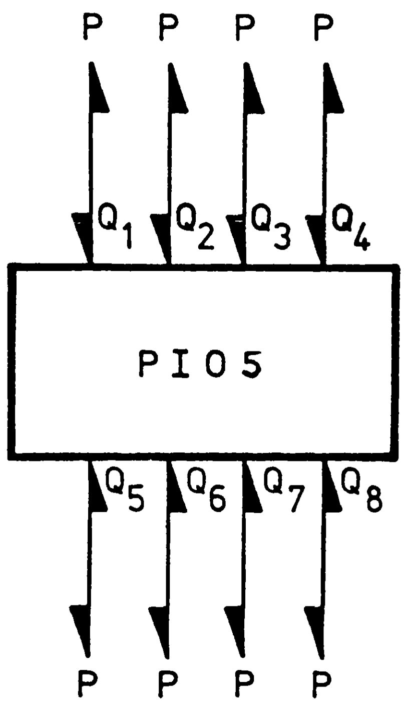


**FIGURE 5.4 LINK DIAGRAM FOR PI05, A FRICTIONLESS PIPE MODEL**

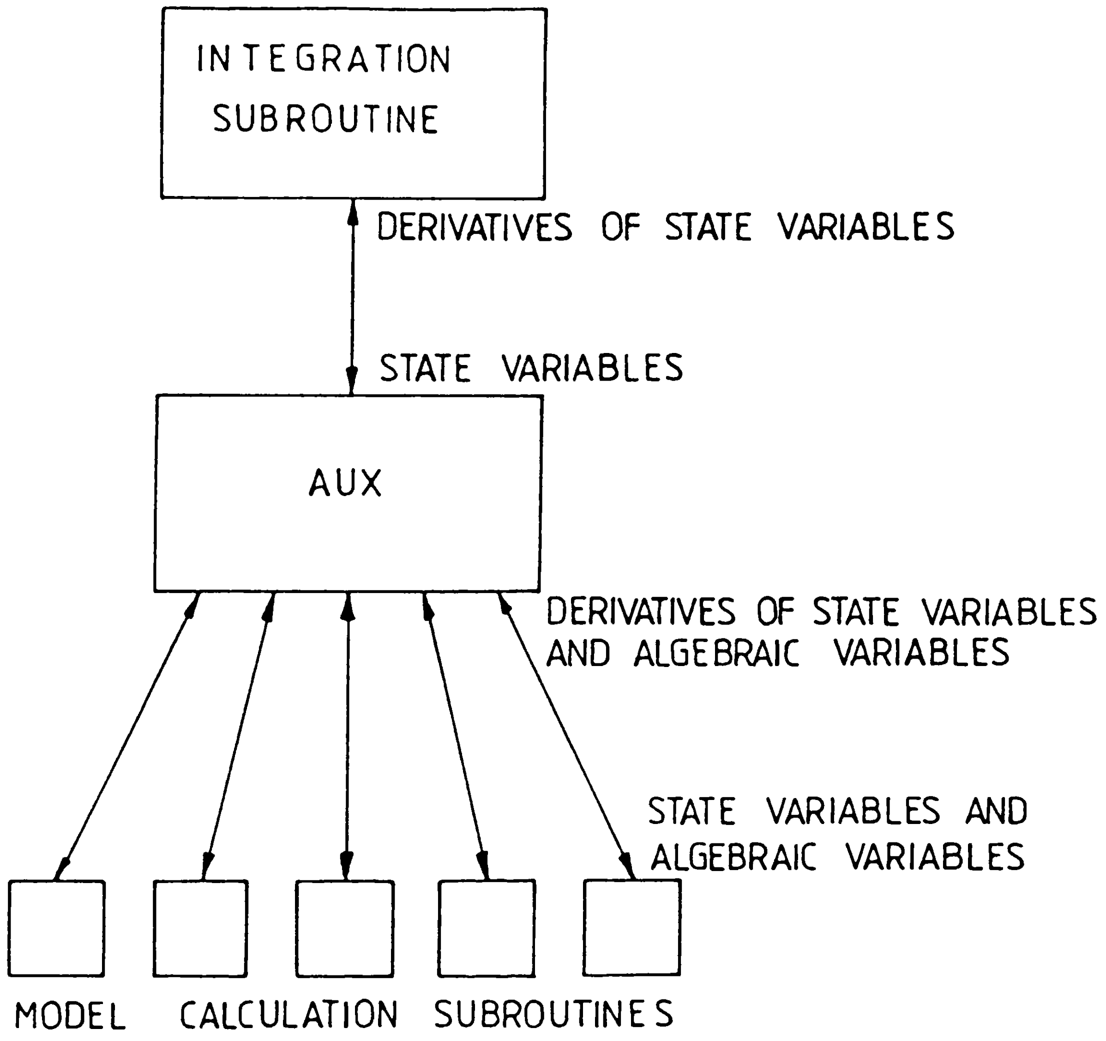

INTEGRATION SUBROUTINE
DERIVATIVES OF STATE VARIABLES
STATE VARIABLES
AUX
DERIVATIVES OF STATE VARIABLES AND ALGEBRAIC VARIABLES
STATE VARIABLES AND ALGEBRAIC VARIABLES
MODEL CALCULATION SUBROUTINES

FIGURE 5.5 THE INTEGRATION PROCESS INSIDE OF HASP

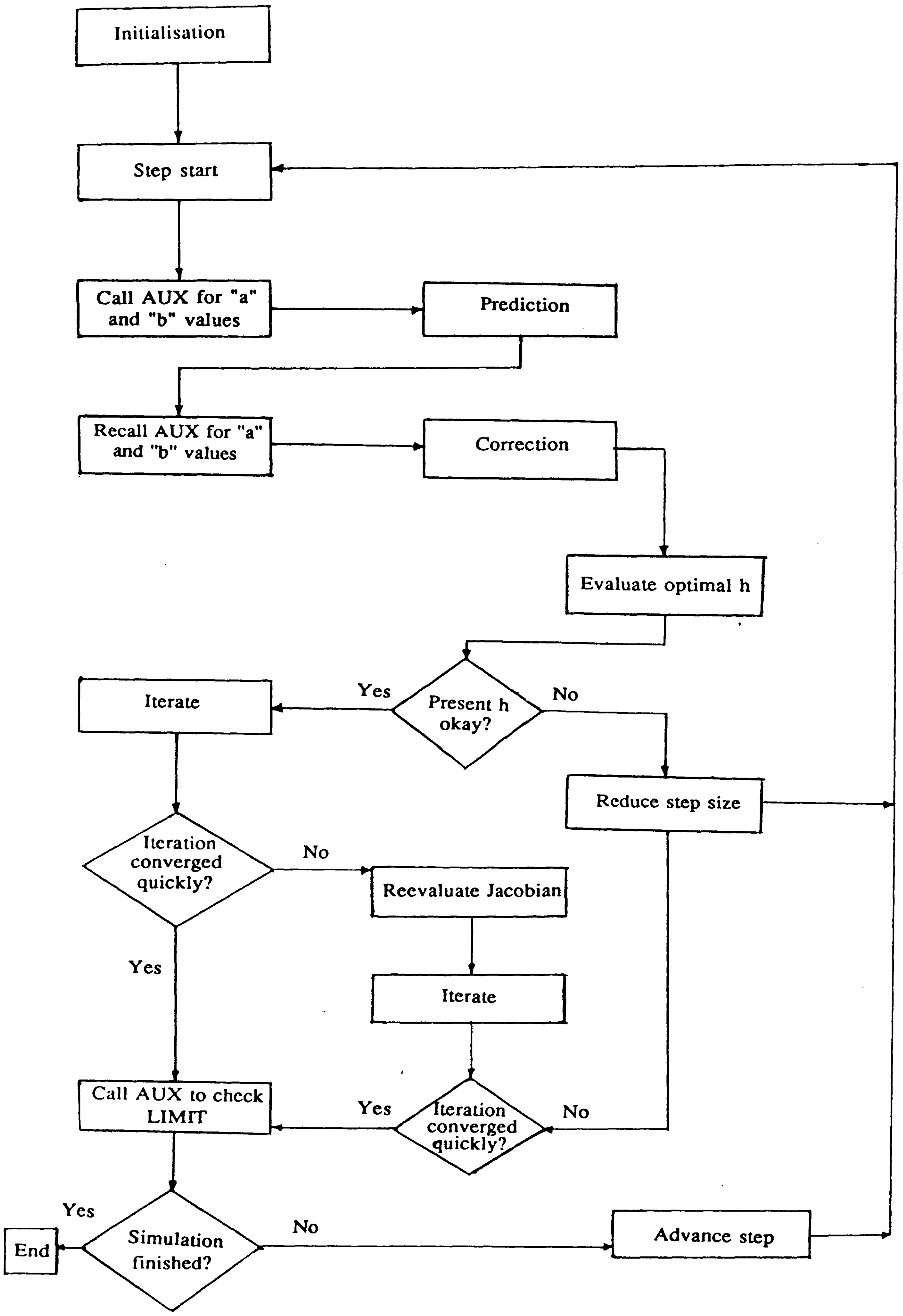


FIGURE 5.6 THE ALGORITHM ADOPTED IN IMPLEMENTING THE NEW INTEGRATION METHOD

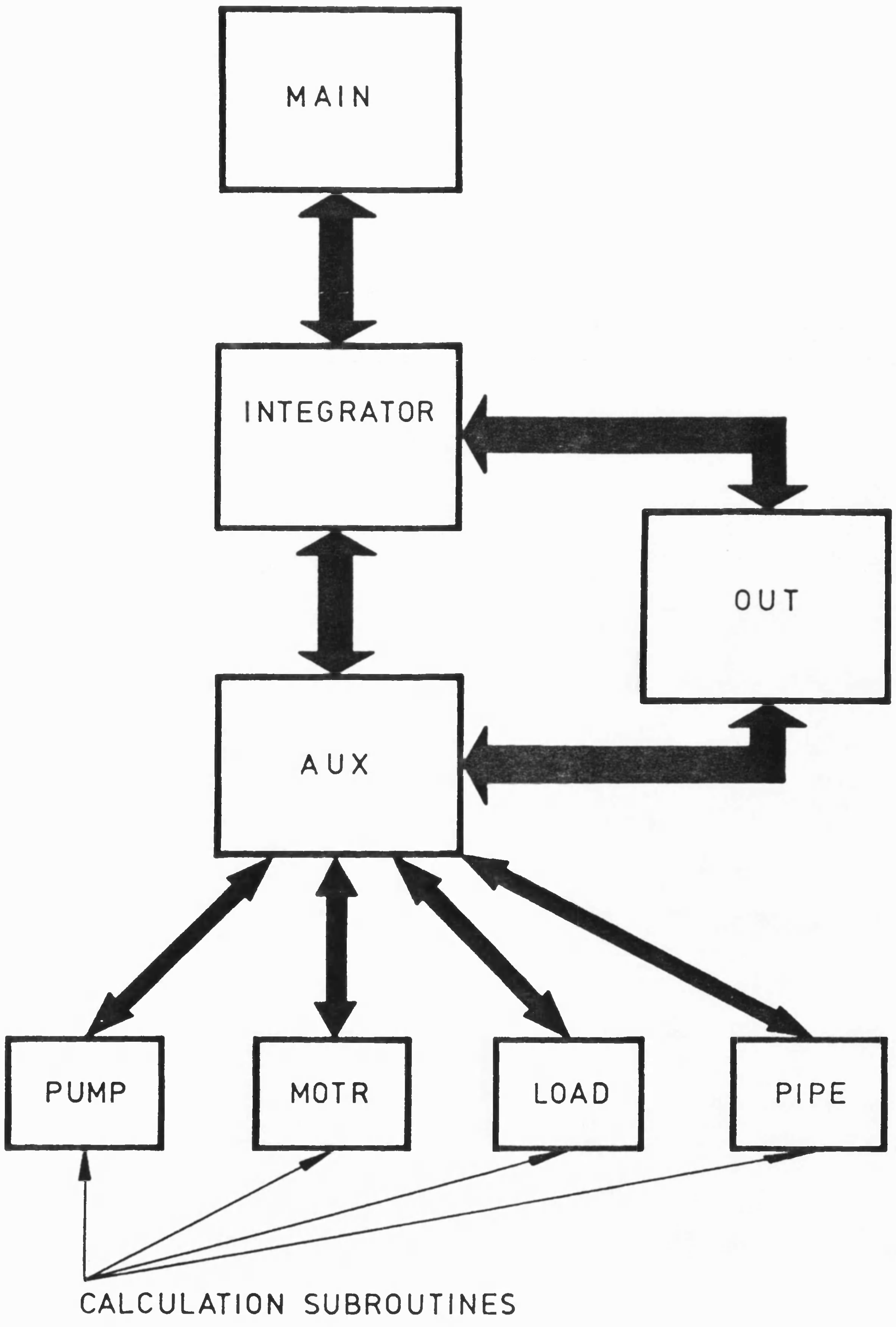


FIGURE 5.7 A TYPICAL HASP SIMULATION PROGRAM, HIGHLIGHTING THE MAIN FEATURES

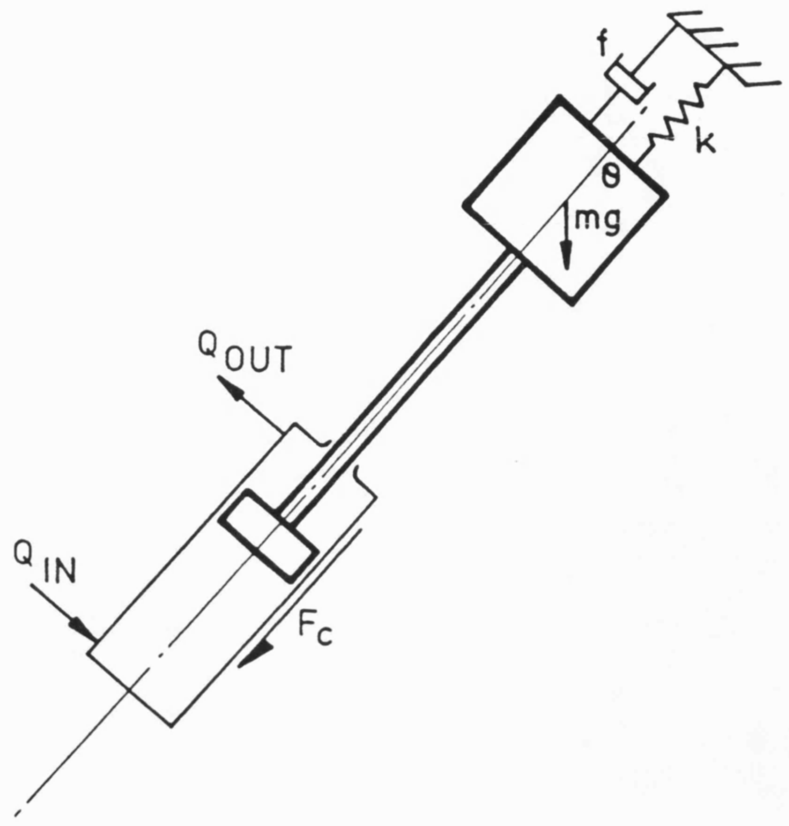


FIGURE 5.8 LINEAR ACTUATOR WITH LOAD

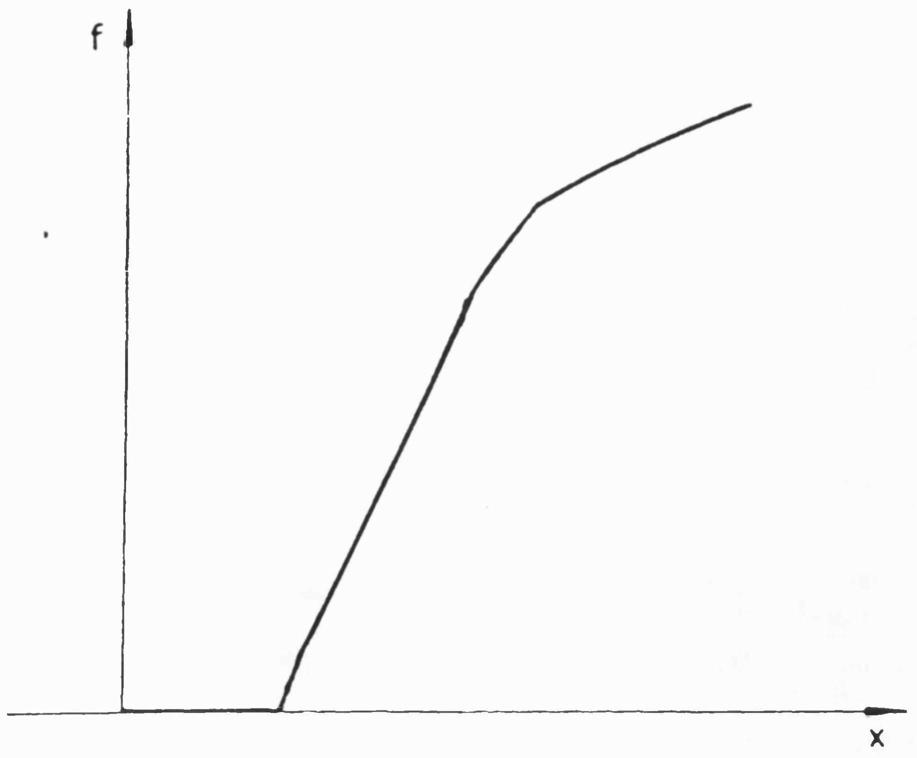


FIGURE 5.9 OPERATING CHARACTERISTIC OF A TYPICAL COMPONENT MODEL TO DEMONSTRATE DISCONTINUITIES

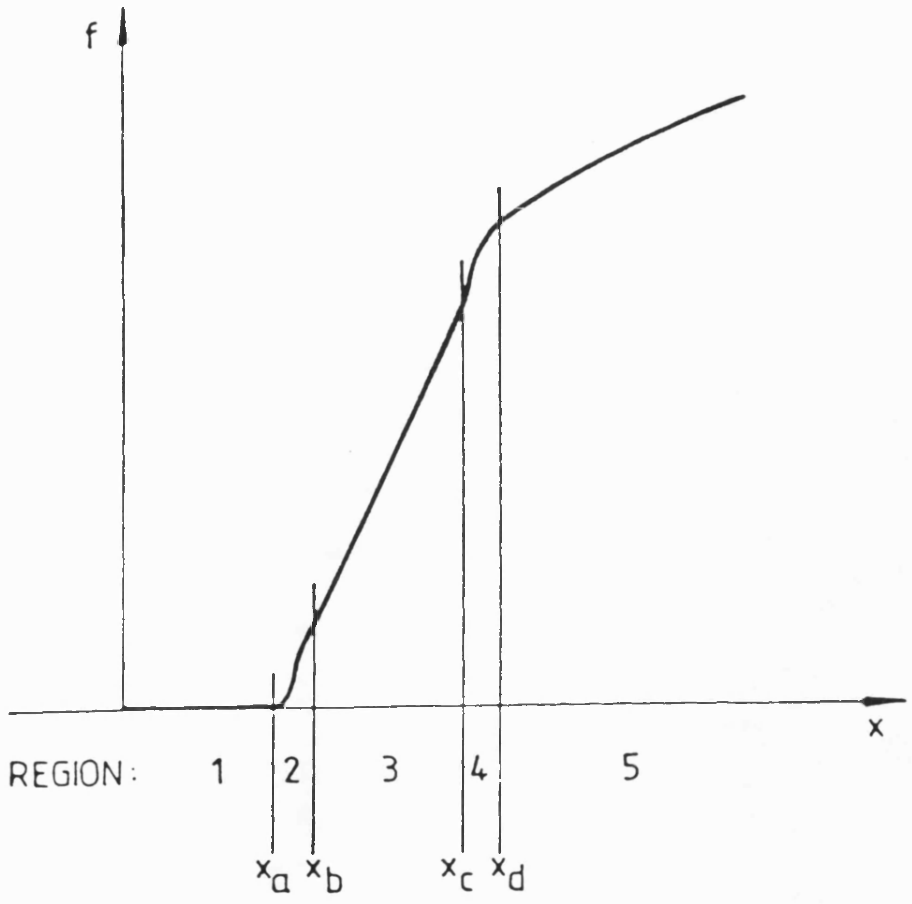


FIGURE 5.10 ACTION TAKEN BY HASP TO ENSURE GEARKC HAS NO DIFFICULTIES IN SOLVING THIS PROBLEM

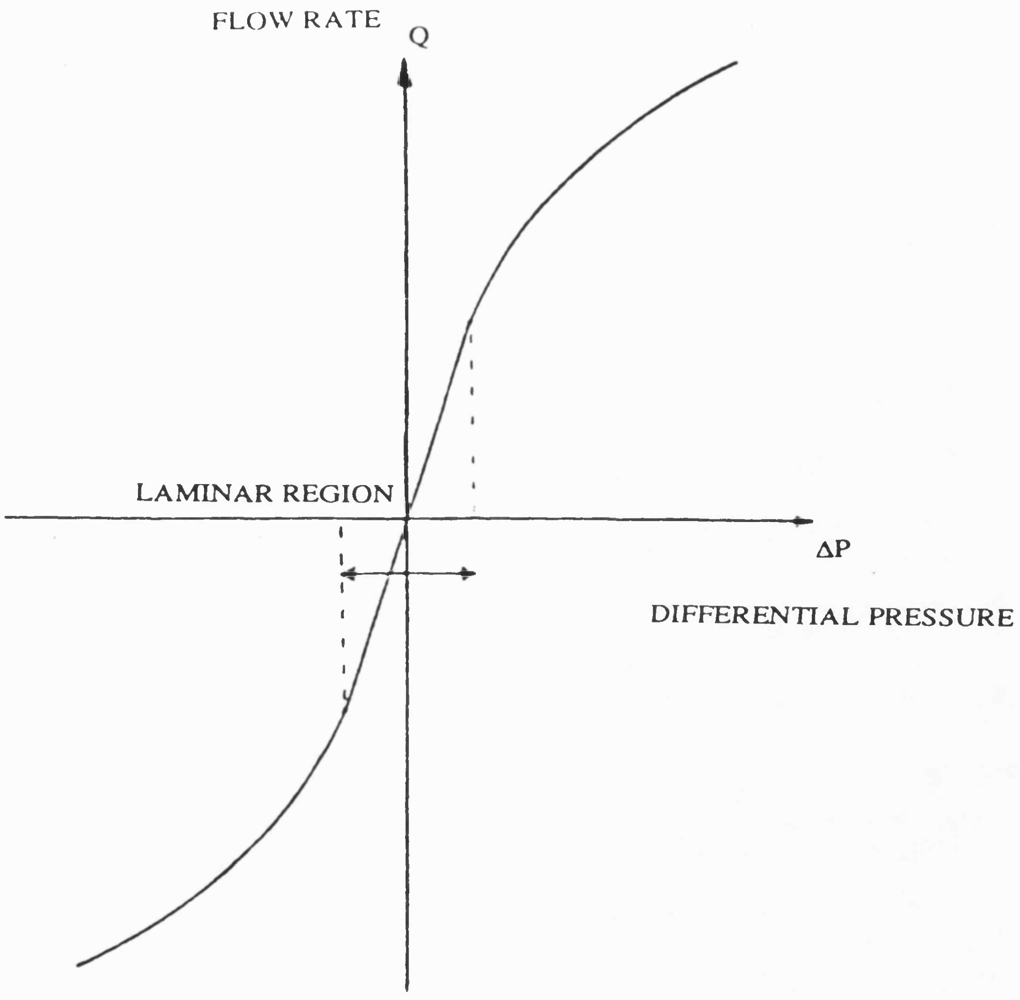


FIGURE 5.11 FLOW RATE V DIFFERENTIAL PRESSURE FOR AN ORIFICE MODEL

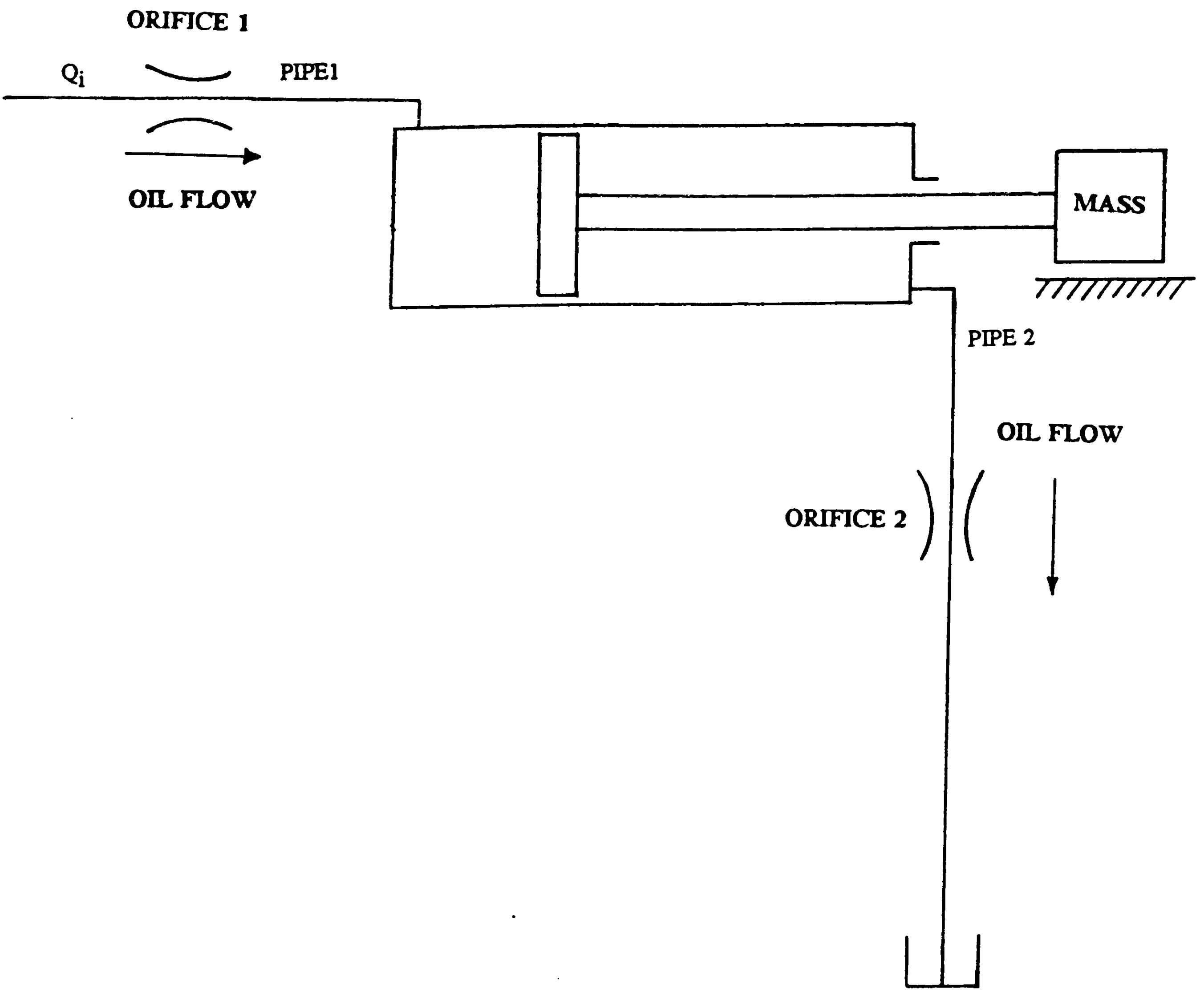


FIGURE 5.12 CIRCUIT DIAGRAM FOR SECOND PROBLEM

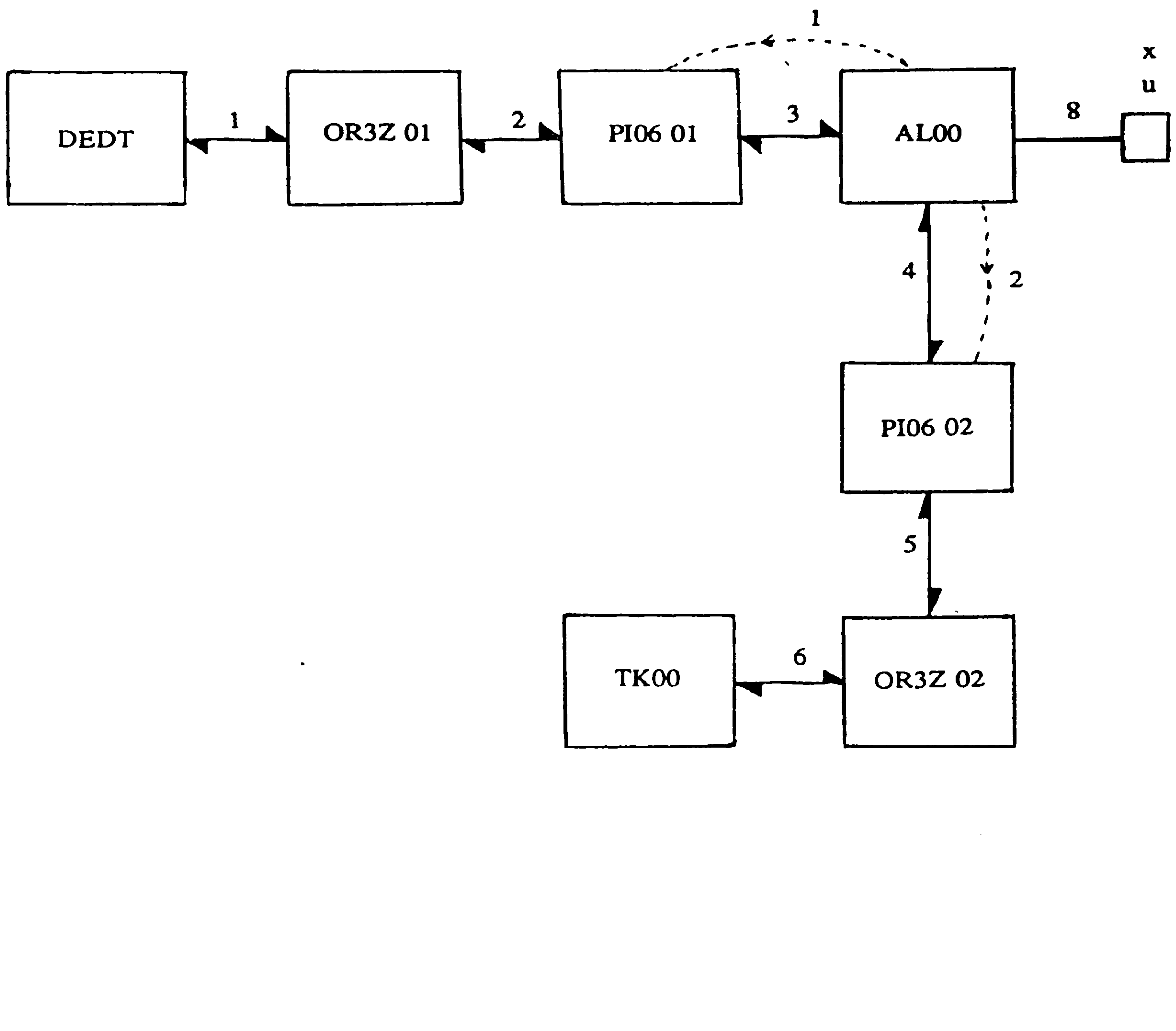


FIGURE 5.13 POWER BOND DIAGRAM FOR SECOND PROBLEM

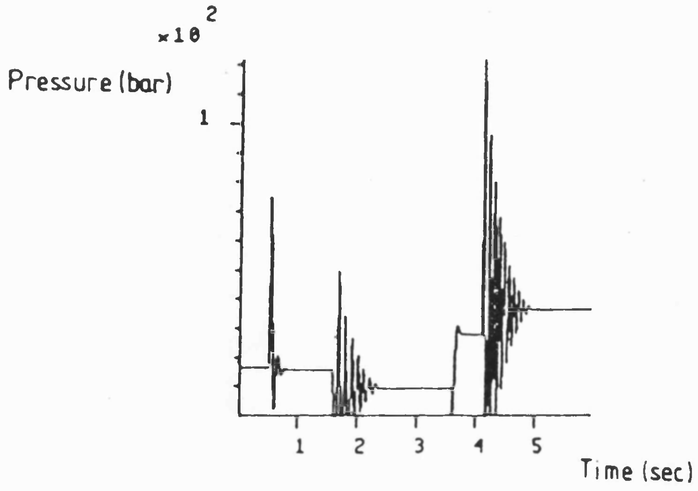


**FIGURE 5.14 SIMULATED PISTON PRESSURE FOR EXAMPLE 1 FOUND USING NEW METHOD**

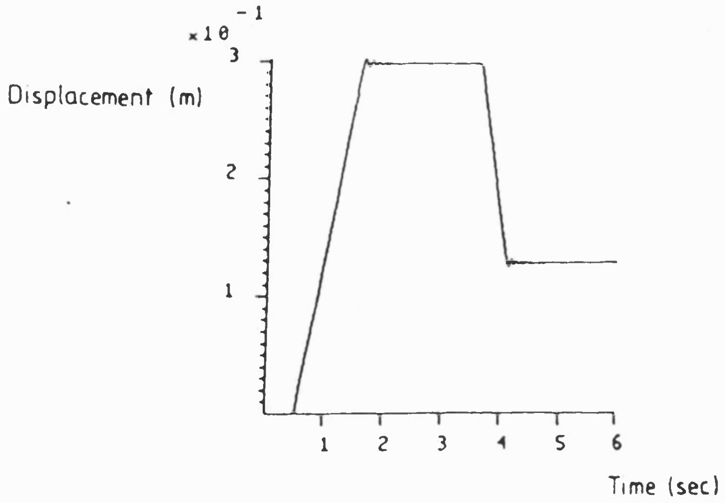


**FIGURE 5.15 SIMULATED ACTUATOR DISPLACEMENT FOR EXAMPLE 1 FOUND USING NEW METHOD**

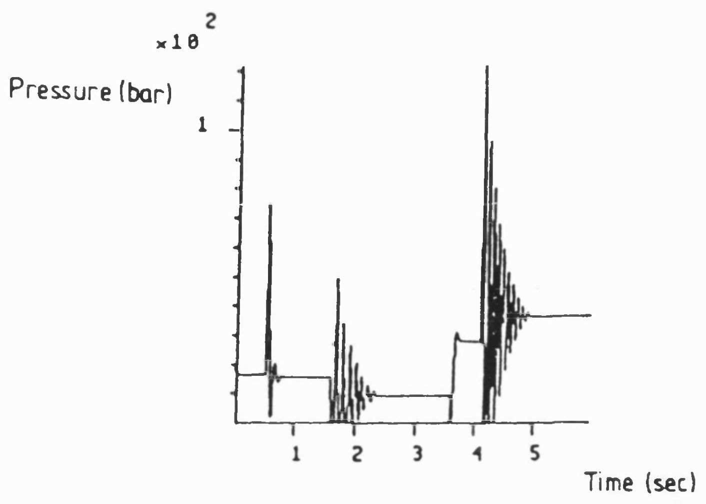


FIGURE 5.16 SIMULATED PISTON PRESSURE FOR EXAMPLE 1 FOUND USING GEAR'S METHOD

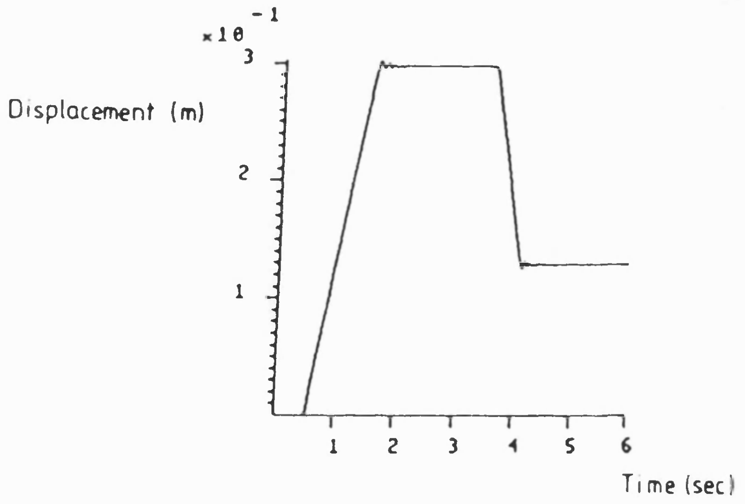


FIGURE 5.17 SIMULATED ACTUATOR DISPLACEMENT FOR EXAMPLE 1 FOUND USING GEAR'S METHOD

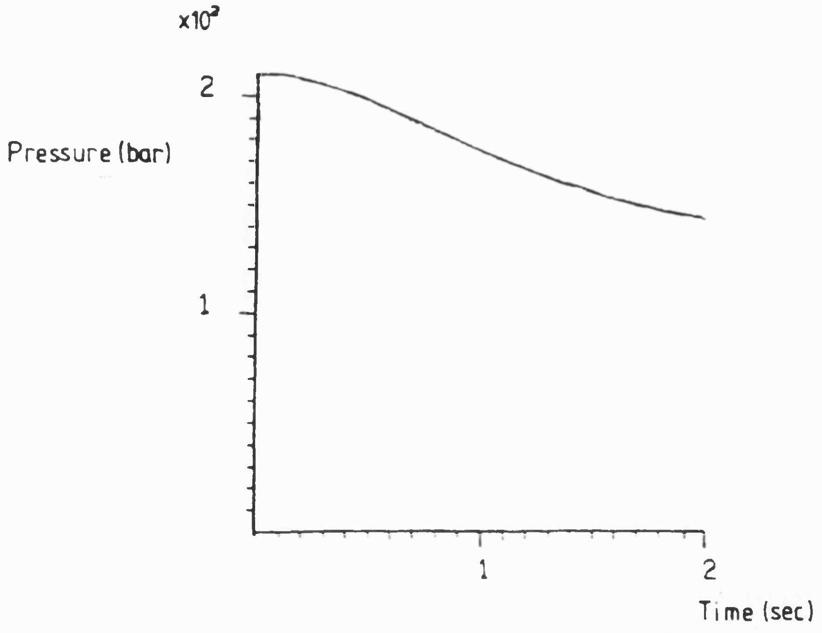


FIGURE 5.18 SIMULATED PISTON PRESSURE FOR EXAMPLE 2 FOUND USING NEW METHOD

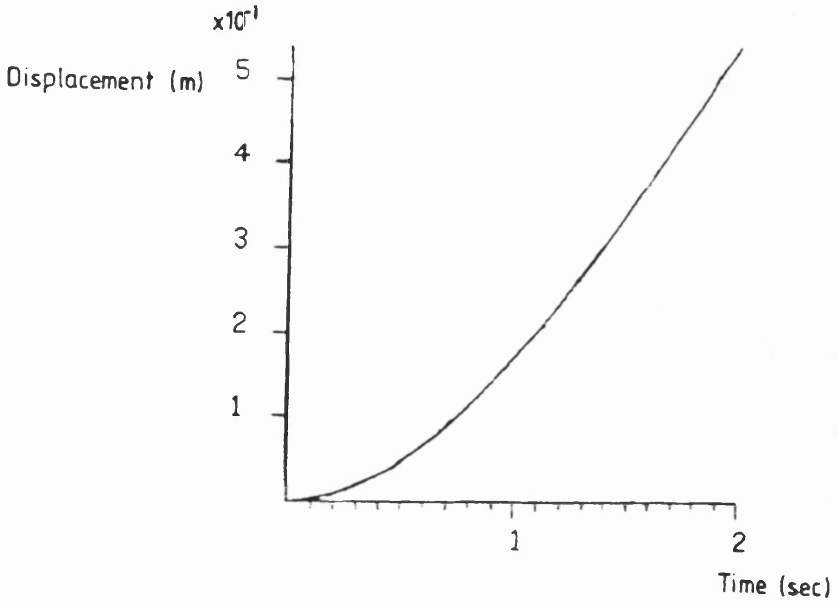


FIGURE 5.19 SIMULATED ACTUATOR DISPLACEMENT FOR EXAMPLE 2 FOUND USING NEW METHOD

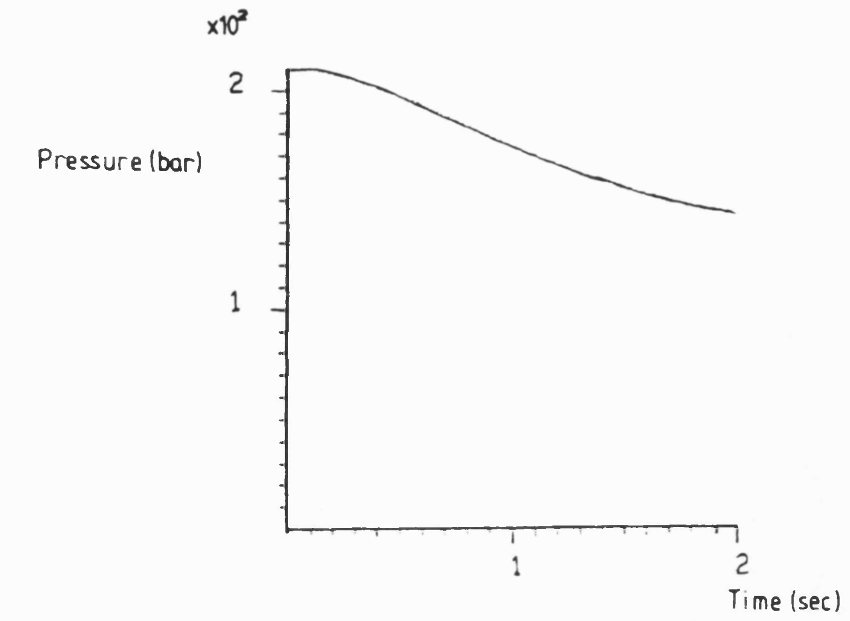


FIGURE 5.20 SIMULATED PISTON PRESSURE FOR EXAMPLE 2 FOUND USING GEAR'S METHOD

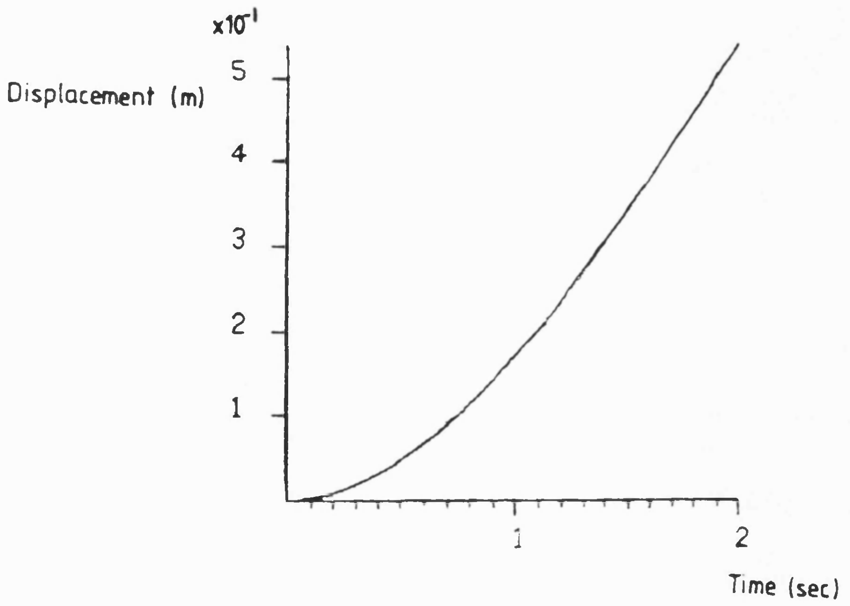


FIGURE 5.21 SIMULATED ACTUATOR DISPLACEMENT FOR EXAMPLE 2 FOUND USING GEAR'S METHOD

# CHAPTER 6

**DETAILED CONTENTS** **Page**

# CHAPTER 6

# DETEST - A FORTRAN PACKAGE FOR ASSESSING INITIAL VALUE METHODS

## Introduction

**6.1** The purpose of this chapter is to describe, and apply to the new method, a package designed to aid in the assessment of Initial Value methods for stiff systems of ordinary differential equations. This package consists of a collection of FORTRAN routines, combined with a canonical set of test problems, and was written by W.H. Enright and J.D. Pryce [17]. This collection of routines is intended to contribute to the determination of the problem domain over which a method is suitable, and to identify where possible weaknesses exist in the implementation. As such, these routines, combined with the test problems, can prove helpful in the development as well as the testing stage of a numerical method. Also, the package provides a wide range of problems to supplement the Fluid Power problems that have already been considered.

**6.2** It is assumed that the Initial Value method to be assessed can be applied to problems of the form:

$$y' = f(t,y) \qquad y(t_0) = y_0 \qquad \text{on the interval } [t_0, t_f] \tag{6.1}$$

All the test problems in the package are autonomous, where the right hand side of equation (6.1) depends only on y, i.e. $y' = f(y)$.

**6.3** Before describing the details of the package, the properties that a subroutine for solving stiff differential equations should possess are discussed, and these properties motivate the statistics generated by the package. Then, an outline of how to use the package, and the options that are available to the user, is given. This outline is accompanied by a description of the statistics that are generated, and the error conditions that can arise. Next, the package design is described and an overview of the individual routines that comprise the package is given. Finally, the implementation of the package on the VAX 750 computer within the School of Engineering is discussed, and then the new method is tested with the package.

## O.D.E. Solvers

**6.4** Although many codes have been developed for either non-stiff or stiff systems of O.D.E.'s in recent years, and these differ in the way that they are implemented or used, there are some standard options that are provided by virtually all general purpose integration methods. One such option is that a method be able to handle a request to integrate between the initial point $t_0$, and the endpoint, $t_f$, within a presribed tolerance, TOL, which is set by the user. In addition, it is expected that this can be interpreted to mean that the method was designed with the aim of keeping the magnitude of the global error proportional to TOL. This was explained more fully in chapter 4.

**6.5** The method to be assessed is implemented in a subroutine called SOLVER. The interface between SOLVER and the testing package is achieved by constructing a driver subroutine called METHOD, which is written by the user of the package. METHOD declares the workspace, initialises the parameters required by SOLVER, signals the appropriate options, invokes SOLVER, and then returns to the MAIN body of code. The calling sequence for METHOD is:

CALL METHOD (N, X, Y, XEND, TOL, HMAX, HSTART )

An example of the routine METHOD is given in Appendix D. The package assumes that the driver, METHOD, is written by the user so that, if the integration is successful, METHOD will return with the value of the independent variable, X, equal to XEND, the final integration time, and the dependent variable values, Y, updated to the approximate solutions at XEND. N is the number of differential equations being solved. If METHOD has not been successful, i.e. SOLVER has failed for some reason, it is assumed that the return to the calling program is made with X less than XEND. Consequently, for all of the test problems, the initial value of X is less than XEND. The parameters HMAX and HSTART are the maximum permitted step size and the initial starting step size, although these parameters may be ignored, or overwritten, by the method being assessed.

**6.6** The subroutine of the testing package that evaluates the differential equation is called FCN, and is invoked by reference to FCN(X, Y, YP). This subroutine evaluates the differential equation at the point X, Y, and then returns the vector of derivatives in the vector YP. Similarly, the calling sequence of the subroutine PDERV which evaluates the Jacobian matrix of partial derivatives is:

CALL PDERV(X, Y, DY)

Figure 6.1 shows the overall structure of the Enright package; each of the routines in the diagram is explained in detail later in the chapter.

**6.7** The testing program assesses a method by monitoring the method's peformance in solving a designated set of test problems. The standard statistics generated by the package will allow the evaluation of an individual method. However, direct comparisons of methods using the standard statistics is difficult, since different methods will be satifying the accuracy requirement differently. That is, the 'constant of proportionality' relating the requested tolerance and the achieved accuracy, will be method, as well as problem, dependent. As a result, if comparisons are to be made, the efficiency statistics must first be 'normalised'. The testing package will, as an option, produce tables of normalised efficiency statistics.

### Statistics Generated And Options Available

**6.8** Each run of the testing package allows a user to test his or her method on a specified subset of the available problems using a prescribed set of tolerances. Besides specifying the problems and the tolerances, the user must also specify: the name of the method, which is used in the output titles; whether the problem is to be solved in its scaled or unscaled form; the level of statistics required, and whether normalised efficiency statistics are desired.

**6.9** The complete set of problems available to a user include the five classes of test problems introduced by Enright, Hull and Lindberg [76], as well as two additional groups, one consisting of chemical kinetics problems, and the other of discontinuous problems. These problems are specified in Appendix D. It is also possible to add problems to the available set, and this is explained by Enright and Pryce [17]. The package requires the user to specify either the scaled or unscaled version of each problem. The 'unscaled' version refers to the problem in its natural scaling as it was first encountered, while the 'scaled' version refers to the original problem scaled by a diagonal scaling matrix chosen so that the maximum magnitude of each component of the scaled problem is one, over the range of integration.

**6.10** There are three levels of detail that are available, and the user selects the level he wishes by setting the option flag OPTION. The basic statistics available, corresponding to OPTION = 1, includes six measures of efficiency: TIME, OVHD, FCN CALLS, JAC CALLS, MAT FACT and NO OF STEPS; and two measures of reliability: END PNT GLB ERR and SMOOTHNESS measures. The standard table of statistics generated for each problem is illustrated in Figure 6.2. The definition of the efficiency measures are:

| | | |
|---|---|---|
| TIME | - | the total processor time in seconds required to solve the problem |
| OVHD | - | the total time in seconds excluding the time used in evaluating the derivatives, evaluating the Jacobian, and performing matrix factorisations |
| FCN CALLS | - | the total number of derivative evaluations required to solve the problem |
| JAC CALLS | - | the number of Jacobian evaluations, i.e. calls to PDERV, required to solve the problem |
| MAT FACT | - | the number of equivalent matrix factorisations, i.e. L-U decompositions, required to solve the problem |
| NO OF STEPS | - | the total number of steps required to solve the problem |

The definition of the reliability measures are:

| | | |
|---|---|---|
| END PNT GLB ERR | - | the global error at the endpoint, XEND, measured in the max-norm, and expressed in units of TOL |
| SMOOTHNESS | - | a measure of how well the method was able to keep the magnitude of the global error at the end point proportional to the tolerance |

More detailed levels of assessment, corresponding to OPTION = 2 and OPTION = 3, are also available to the user, as well as the choice of normalised or standard efficiency statistics, and these are fully explained by Enright and Pryce [17]. However, in testing the new method, only the efficiency and reliability measures already mentioned have been used, apart from a further reliability measure, MAX GLB ERR, which records the maximum global error observed throughout the integration, measured in units of the tolerance, TOL. The statistics required to test the new method correspond to OPTION = 2, and the standard table of statistics generated for each problem when using this value of OPTION is illustrated in Figure 6.3. No normalised statistics have been requested.

**6.11** There are two types of error conditions that can arise whilst using the testing package. The first is a failure of the method being assessed to successfully reach the endpoint, XEND. When this occurs, partial statistics are output, reflecting the costs before the failure, but the timing statistics and endpoint error statistics are omitted. In addition to the line of partial results, a message is also printed indicating that the method has failed to reach the endpoint. If the method was unable to take any steps, no partial statistics are available, and only the message 'METHOD FAILED TO START' is output. Summary statistics are only given over integrations that were successfully completed by the method being tested.

**6.12** There is a second type of failure that can occur when detailed results of each step are being monitored, such as when OPTION = 2. In this case, the testing package must obtain the 'true' global solution on each step, and possibly the 'true' local solution as well. These 'true' solutions are accurately determined using a reliable method, namely the Addison-Enright second derivative method, operating with an accuracy requirement more stringent than the TOL requested by the user. If this method is unable to obtain the solution, such as may happen when a very stringent tolerance has been specified, then the appropriate statistics assessing the maximum global error or local errors will not be complete. However, if the method being tested is still successful, the efficiency and endpoint accuracy results will be available and are reported. When this situation arises, a message is also output warning that a failure of the true solution has occurred.

**The Constituent Programs In The DETEST Package**

**6.13** The package is made up of several subroutines which use labelled COMMON statements for the communication of results. Again the reader is referred to Figure 6.1, which illustrates the overall working structure of the package. The main supervisory subroutine, STDTST, is called by the main program, which is written by the user. This subroutine supervises the output of the headings and the statistics, and processes all the problems selected by the user, ensuring that each problem is solved at the prescribed set of tolerances. The subroutine CONTRL is the routine that controls the integration of one problem at a specified tolerance. It does this by setting up the initial conditions for the problem, invoking the method being tested, and organising the collection of statistics. The subroutine STATS is called by the method being assessed at the completion of each successful step. Its purpose is to perform the required detailed analysis of the results of each step. The subroutines of the package which specify the information related to the individual test problems are:

|  |  |
|---|---|
| IVALU - | subroutine which specifies the initial conditions for each problem |

| | | |
|---|---|---|
| FCN | - | subroutine which evaluates the derivative |
| PDERV | - | subroutine which evaluates the Jacocian matrix of partial derivatives |
| EVALU | - | subroutine which specifies the true solution of each problem evaluated at the endpoint |

The subroutines IVALU, FCN, PDERV and EVALU are organised as case statements. A flag ID, which is passed through COMMON, identifies the problem being solved, and determines the appropriate block of code to be executed.

**6.14** The user has to write a main program which must determine: which problems are required to be solved by the method being tested; a list of the tolerances, TOL, with which to solve these problems; the value of OPTION for the statistical information required; whether normalised statistics are needed and which title to print with the output. An example of this driving program is given in Appendix D. STDTST is called by the main routine using the calling statement:

```
CALL STDTST(TITLE, OPTION, NORMEF, TOL, IDLIST, FLAG)
```

where the variables are defined by:

| | | |
|---|---|---|
| TITLE | - | integer variable containing the title to be printed over the statistics |
| OPTION | - | integer variable containing the value corresponding to the level of statistics required |
| NORMEF | - | integer variable that determines whether normalised or standard results are produced |
| TOL | - | real array containing up to ten tolerances on each program run |
| IDLIST | - | integer array which holds a list of the groups of problems to solved. Each problem is specified by a numeric code. If the problem code is given a negative sign, the system is integrated in unscaled form; if a positive sign, in scaled form |

| | | |
|---|---|---|
| FLAG | - | an error flag which determines where an error has arisen, and why, depending on the value that it returns |

**6.15** The subroutine CONTRL sets up the initial values and other problem dependent parameters for the test problems being integrated, and initiates one or more integrations of the problem. First, an appropriate starting stepsize is determined by invoking METHOD with a rough starting step size, and observing the choice of step size over the first two successful steps. This preliminary integration is aborted prematurely, and the appropriate starting stepsize, HSTART, is used in subsequent invocations. METHOD is then invoked again, and the appropriate statistics for the values of the variable OPTION gathered.

**6.16** Other subroutines of the package include: COEFF, STEP, NEWSTP, DDCOMP and SOLVE, which together with TRUE comprise the SECDER method described in detail by Addison [77], and used in the package to generate the 'true' local and global solutions, when so required by the routine STATS; PARCHK, which is invoked by STDTST to check that the user has supplied meaningful and consistent parameter values; LSQFIT and EFSTAT which perform the linear least squares fit and determine the normalised efficiency statistics respectively.

## Testing The New Method

**6.17** Many difficulties were encountered when transporting the DETEST package onto the VAX 750 computer in the School of Engineering. The major problems arose with the amount of storage space and memory required when interactively running the package. Unfortunately, this severely limited the number of problems that could be used to test the new method. The timing clock that accompanied the package also proved to be unsuitable, so a new routine had to be written to time the integration processes, based on the C.P.U. timer discussed in chapter 3. Finally, the word size on the VAX system only allows numbers, x, in the range $1 \times 10^{-38} < x < 1 \times 10^{38}$, and the writers of DETEST have assumed that the machine on which it is run is able to store numbers which have much larger exponents.

**6.18** Consequently, although the aims of the package are clear and constructive in testing a new integration method, in practice the package proved to very troublesome to implement. However, it was possible to test the new method on a limited set of test problems. To use the package, further modifications had to be made, since the general purpose predictor-corrector form of the new method being tested evaluates its own Jacobians using a perturbation technique, and also does not require function evaluations, but coefficient values, as explained in chapter 3. Consequently, it was necessary to write separate function routines for each problem to give the new method the coefficient values at each time step, and to modify the counters in the package that record the number of Jacobian matrices that are evaluated, and the number of function calls that are made.

## Results

**6.19** Figures 6.4, 6.5 and 6.6 illustrate the results given by the package, when using the new method in comparison with Enright's and Addison's method SECDER, for some of the test problems given in Appendix D. Figures 6.7, 6.8 and 6.9 illustrate the results given for SECDER in solving these same problems, which have all been solved in unscaled form. Problem A1 is a linear first order differential equation with constant coefficients, and the new method provides the exact solution to this problem, since it is the solution to an equation of this form from which the original numerical scheme was derived in chapter 3. The problem was solved on the interval [0, 20] and the statistics corresponding to OPTION = 2 found for the new method are shown in Figure 6.4. As is expected, the method shows a marked improvement over SECDER in solving this particular problem. Figure 6.5 shows the statistics found for problem A2, and again, this problem is linear. It can be seen that the statistics are much different to those found for the new method with problem A1. The number of function calls refers to the number of coefficient calls that were made during the integration of the problem, which was on the interval [0,120]. The smoothness fit of LOG10(ERROR) VS LOG10(TOL) indicates that the accuracy of the method is low, since the exponents of the global error formulae are much less than one, unlike the exponents for the same smoothness curves with SECDER, which are shown in Figure 6.8 for this problem, and prove to be close to one. Similarly, the mantissa values of the smoothness curves for the two methods demonstrate the difference in accuracy. Figure 6.6 shows the statistics found for the new method when applied to problem A3, and for this problem, which is solved on the interval [0, 20], the exponents of the smoothness curve for the new method are larger in magnitude. The corresponding statistics found for SECDER are shown in Figure 6.9. The reason that the number of MAT FACTs is zero in Figures 6.4, 6.5 and 6.6 is because a Newton-Jacobi iteration scheme has been employed with the version of the new method tested by the Enright package, and this does not require matrix inversions or L-U decompositions, since only the diagonal elements of the Jacobian are considered. This was explained in chapter 2.

## Conclusions

**6.20** Unfortunately, the package did not prove to be as useful for testing the new method as had been originally hoped. The intention had been to use the package to compare various methods, such as Gear and Runge-Kutta, with the new method, over a range of test problems, without the testing being problem dependent. However, the difficulties encountered whilst implementing the package stopped any worthwhile comparisons being made. For most of the problems on which the method could be tested, it was found that the low order of the method was a handicap, and often, prohibitively small time steps resulted during the integration process which led to problems with the method evaluating the 'true' solution, and no constructive results were forthcoming. The only conclusion that can be drawn is that it would be advantageous to increase the accuracy of the new method, if it is to compare more favourably with SECDER, over the range of test problems on which it has been tested. Also, it was felt that the set of problems used by the package could profitably be extended by the inclusion of the typical problems arising in hydraulics, which lead to systems that are both stiff and discontinuous.

**6.21** From the results found in this and the previous chapter, to continue with the new method, and enable its use as a competitive integrator, it appears that the order of accuracy of the method must be increased, since first order accuracy is not good enough for a general purpose integration routine, only for a routine used for solving specific problems. In the next chapter, the possibility of extending the order of accuracy of the method is investigated.

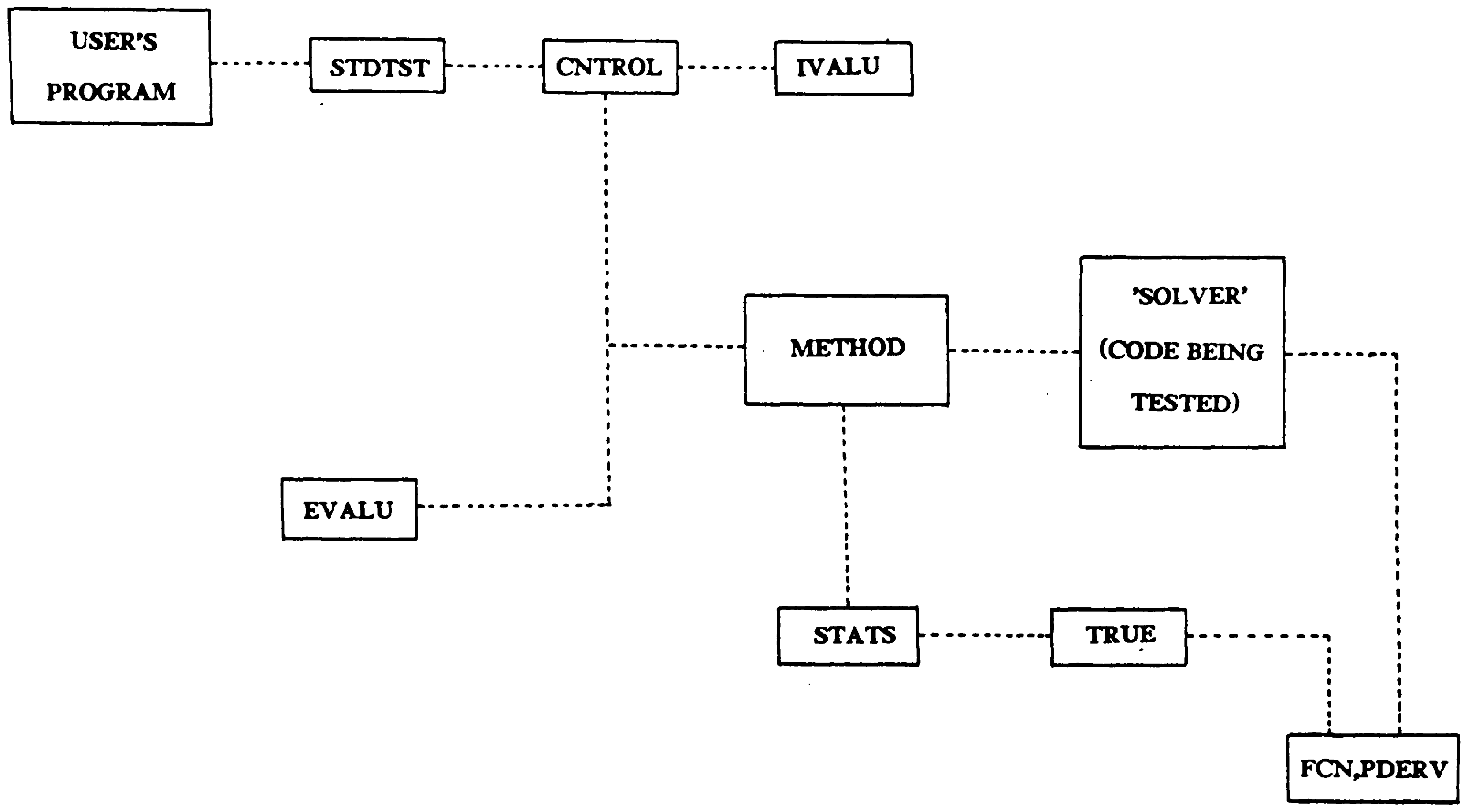


FIGURE 6.1 THE OVERALL WORKING STRUCTURE OF THE DETEST PACKAGE

```
STIFF DETEST PACKAGE    OPTION = 1, NORMEF = 0

GROUP 1             SECDER ADDISON-ENRIGHT SECOND DERIVATIVE METHOD

 E3  (SCALED)


    LOG10    TIME    OVHD     FCN      JAC     MAT    NO OF   END PNT

     TOL                      CALLS    CALLS   FACT   STEPS   GLB ERR

    -2.00    0.027   0.026     91       32      24      22      0.05

    -4.00    0.057   0.054    178       68      39      50      0.06

    -6.00    0.160   0.153    507      192     111     133      0.52

    -8.00    0.127   0.121    384      122      42     110      0.49


        SMOOTHNESS FIT OF LOG10(ERROR) VS LOG10(TOL)

        ENDPOINT GLOBAL ERROR

           = 1.70E-02*(TOL**0.802) APPROX
```

FIGURE 6.2 STANDARD STATISTICS FROM THE TESTING PACKAGE

```
STIFF DETEST PACKAGE    OPTION = 2, NORMEF = 0

GROUP 1              SECDER ADDISON-ENRIGHT SECOND DERIVATIVE METHOD

 E3  (SCALED)


LOG10   TIME    OVHD    FCN     JAC     MAT    NO OF   END PNT   MAXIMUM

 TOL                    CALLS   CALLS   FACT   STEPS   GLB ERR   GLB ERR

 -2.00  0.027   0.026    91      32      24     22      0.05      0.16

 -4.00  0.057   0.054   178      68      39     50      0.06      0.28

 -6.00  0.160   0.153   507     192     111    133      0.52      0.82

 -8.00  0.127   0.121   384     122      42    110      0.49      1.33


        SMOOTHNESS FIT OF LOG10(ERROR) VS LOG10(TOL)

        ENDPOINT GLOBAL ERROR

           = 1.70E-02*(TOL**0.802) APPROX

        MAXIMUM GLOBAL ERROR

           = 7.43E-02*(TOL**0.840) APPROX
```

**FIGURE 6.3 STATISTICS CORRESPONDING TO OPTION = 2**

```
STIFF DETEST PACKAGE    OPTION = 2, NORMEF = 0

GROUP 1                 FIRST ORDER METHOD

 A1  (UNSCALED)


LOG10    TIME     OVHD     FCN     JAC     MAT     NO OF   END PNT    MAXIMUM

 TOL    (MINS)   (MINS)   CALLS   CALLS    FACT    STEPS   GLB ERR    GLB ERR

 -2.00   0.002    0.002     3       0       0        1      0.00       0.00

 -4.00   0.002    0.002     3       0       0        1      0.00       0.00

 -6.00   0.002    0.002     3       0       0        1      0.00       0.00


          SMOOTHNESS FIT OF LOG10(ERROR) VS LOG10(TOL)

          ENDPOINT GLOBAL ERROR
             = 1.000E-02*(TOL**1.000) APPROX

          MAXIMUM GLOBAL ERROR
             = 1.000E-02*(TOL**1.000) APPROX
```

**FIGURE 6.4  STATISTICS FOR THE NEW METHOD WHEN APPLIED TO PROBLEM A1**

```
STIFF DETEST PACKAGE    OPTION = 2, NORMEF = 0

GROUP 1                 FIRST ORDER METHOD

 A2  (UNSCALED)


LOG10    TIME     OVHD     FCN      JAC      MAT    NO OF   END PNT   MAXIMUM

 TOL    (MINS)   (MINS)   CALLS    CALLS    FACT   STEPS   GLB ERR   GLB ERR

-2.00    2.421    2.102     84       54      0       62      0.14      0.21

-4.00    3.682    3.154    190       96      0       94      0.43      0.49

-6.00    5.790    4.987    286      108      0      126      0.65      0.74


        SMOOTHNESS FIT OF LOG10(ERROR) VS LOG10(TOL)

        ENDPOINT GLOBAL ERROR
           = 4.765-02*(TOL**0.572) APPROX

        MAXIMUM GLOBAL ERROR
           = 5.872E-02*(TOL**0.463) APPROX
```

**FIGURE 6.5 STATISTICS FOR THE NEW METHOD WHEN APPLIED TO PROBLEM A2**

```
STIFF DETEST PACKAGE    OPTION = 2, NORMEF = 0

GROUP 1              FIRST ORDER METHOD

 A3  (UNSCALED)


LOG10    TIME     OVHD     FCN      JAC     MAT    NO OF    END PNT    MAXIMUM

 TOL    (MINS)   (MINS)   CALLS    CALLS    FACT   STEPS    GLB ERR    GLB ERR

-2.00   1.972    1.561    212       97      0      234      0.19       0.26

-4.00   2.683    2.136    341      156      0      316      0.45       0.53

-6.00   6.547    6.214    589      240      0      462      0.72       0.77


          SMOOTHNESS FIT OF LOG10(ERROR) VS LOG10(TOL)

          ENDPOINT GLOBAL ERROR
             = 4.432E-02*(TOL**0.773) APPROX

          MAXIMUM GLOBAL ERROR
             = 5.217E-02*(TOL**0.654) APPROX
```

**FIGURE 6.6 STATISTICS FOR THE NEW METHOD WHEN APPLIED TO PROBLEM A3**

STIFF DETEST PACKAGE    OPTION = 2, NORMEF = 0

GROUP 1    SECDER ADDISON-ENRIGHT SECOND DERIVATIVE METHOD

A1 (UNSCALED)

| LOG10 | TIME | OVHD | FCN | JAC | MAT | NO OF | END PNT | MAXIMUM |
|---|---|---|---|---|---|---|---|---|
| TOL | (MINS) | (MINS) | CALLS | CALLS | FACT | STEPS | GLB ERR | GLB ERR |
| -2.00 | 0.361 | 0.335 | 70 | 28 | 19 | 24 | 0.00 | 0.15 |
| -4.00 | 0.641 | 0.606 | 110 | 45 | 22 | 43 | 0.00 | 0.31 |
| -6.00 | 1.130 | 1.081 | 180 | 75 | 26 | 74 | 0.01 | 0.49 |

SMOOTHNESS FIT OF LOG10(ERROR) VS LOG10(TOL)

ENDPOINT GLOBAL ERROR

= 4.142-02*(TOL**0.985) APPROX

MAXIMUM GLOBAL ERROR

= 8.361E-02*(TOL**0.868) APPROX

**FIGURE 6.7 STATISTICS FOR THE SECDER METHOD WHEN APPLIED TO PROBLEM A1**

STIFF DETEST PACKAGE    OPTION = 2, NORMEF = 0

GROUP 1    SECDER ADDISON-ENRIGHT SECOND DERIVATIVE METHOD

A2 (UNSCALED)

| LOG10 | TIME | OVHD | FCN | JAC | MAT | NO OF | END PNT | MAXIMUM |
|---|---|---|---|---|---|---|---|---|
| TOL | (MINS) | (MINS) | CALLS | CALLS | FACT | STEPS | GLB ERR | GLB ERR |
| -2.00 | 1.638 | 1.404 | 81 | 32 | 25 | 28 | 0.00 | 0.05 |
| -4.00 | 3.474 | 3.035 | 193 | 74 | 38 | 66 | 0.00 | 0.40 |
| -6.00 | 4.720 | 4.201 | 279 | 107 | 40 | 97 | 0.0 | 0.42 |

SMOOTHNESS FIT OF LOG10(ERROR) VS LOG10(TOL)

ENDPOINT GLOBAL ERROR

= 6.615E-02*(TOL**0.946) APPROX

MAXIMUM GLOBAL ERROR

= 4.941E-02*(TOL**0.857) APPROX

**FIGURE 6.8 STATISTICS FOR THE SECDER METHOD WHEN APPLIED TO PROBLEM A2**

```
STIFF DETEST PACKAGE    OPTION = 2, NORMEF = 0

GROUP 1              SECDER ADDISON-ENRIGHT SECOND DERIVATIVE METHOD

 A3  (UNSCALED)


LOG10    TIME     OVHD     FCN     JAC     MAT    NO OF   END PNT   MAXIMUM

 TOL    (MINS)   (MINS)   CALLS   CALLS   FACT   STEPS   GLB ERR   GLB ERR

-2.00   0.576    0.513    104      39      30      34     0.01       0.20

-4.00   1.096    0.992    200      75      36      69     0.00       0.82

-6.00   1.801    1.656    302     110      40     109     0.00       0.52


         SMOOTHNESS FIT OF LOG10(ERROR) VS LOG10(TOL)

         ENDPOINT GLOBAL ERROR
            = 1.000E-02*(TOL**1.000) APPROX

         MAXIMUM GLOBAL ERROR
            = 0.222E-02*(TOL**0.931) APPROX
```

**FIGURE 6.9 STATISTICS FOR THE SECDER METHOD WHEN APPLIED TO PROBLEM A3**

# CHAPTER 7

**DETAILED CONTENTS** **Page**

# CHAPTER 7

# APPLICATION OF THE NEW METHOD FOR THE SOLUTION OF PARTIAL DIFFERENTIAL EQUATIONS

## Introduction

**7.1** This chapter considers a particular application of the new method. Since the explicit method was found to be stable for diagonally dominant systems, and the discretisation of parabolic partial differential equations (p.d.e.'s) often leads to such systems, it was decided to apply the method to solve some of these equations. This chapter looks at parabolic p.d.e.'s and explains their reduction to a set of ordinary differential equations, the way in which they are usually solved, and the very stiff systems that often result. Then the new method is applied to solve the differential equations, and compared with traditional methods, such as the Crank-Nicolson method. Finally, the possibility of extending the accuracy of the method is discussed, and two approaches that have been adopted are discussed.

## Reduction To A System Of Ordinary Differential Equations

**7.2** Here, a parabolic p.d.e. [68] is considered, and the way in which it can be reduced to a set of o.d.e.'s is explained. The equation considered is the heat equation, given by:

$$U_t = \kappa U_{xx}$$

This is one of the simplest parabolic p.d.e.'s, and is derived from the theory of heat conduction. Its solution gives, for example, the temperature U at a distance x units of length from one end of a thermally heated bar after t seconds of heat conduction. For such a problem, the temperature at the ends of a bar of length 1 (say) are often known for all time, i.e. the boundary conditions are known. It is also usual for the temperature distribution along the bar to be known at some particular instant. This instant is usually taken as zero time, and the temperature distribution is called the initial condition. The solution gives U for values of x between 0 and 1, and values of t from zero to infinity. To find the solution, finite difference schemes are normally applied directly to the parabolic equation, but it is also possible to reduce the equation to a system of o.d.e.'s and then apply standard o.d.e. solvers.

7.3 Consider the partial differential equation

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} \qquad 0 < x < X, \;\; t > 0$$

where U(x,t) satisfies the inital condition

$$U(x,0) = g(x) \quad 0 \le x \le X$$

and has known boundary values at $x = 0$ and $x = X$, for $t > 0$, i.e.

$$U(0,t) = \alpha(t) \quad t > 0$$

$$U(X,t) = \beta(t) \quad t > 0$$

In this chapter, $\Delta t$ is denoted as k, and $\Delta x$ is denoted as h. Also, $r = k/h^2$.

If the right hand side of the p.d.e. is replaced at (x,t) by

$$\frac{\partial^2 U}{\partial x^2} = \frac{1}{h^2} \{ U(x-h, t) - 2U(x, t) + U(x+h, t) \} + O(h^2) \tag{7.1}$$

where the interval $0 \le x \le X$ is subdivided into N equal intervals by the grid lines $x_i = ih$, $i = 0, 1, \ldots, N$ with $Nh = X$; and writing equation (7.1) at every mesh point $x_i = ih$, $i = 1, 2, \ldots, N-1$ along time level t, the values $V_i(t)$ approximating $U_i(t)$ will be the exact solution values of the system of (N-1) ordinary differential equations given by:

$$\frac{dV_1(t)}{dt} = \frac{1}{h^2} ( V_0 - 2V_1 + V_2 )$$

$$\frac{dV_2(t)}{dt} = \frac{1}{h^2} ( V_1 - 2V_2 + V_3 )$$

$$\begin{array}{cc} . & . \\ . & . \\ . & . \end{array}$$

$$\frac{dV_{N-1}}{dt} = \frac{1}{h^2} ( V_{N-2} - 2V_{N-1} + V_N )$$

where $V_0$ and $V_N$ are known boundary values for all t. These equations can be written in matrix form as:

$$\frac{d}{dt}\begin{bmatrix} V_1 \\ V_2 \\ . \\ . \\ V_{N-1} \end{bmatrix} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & . & . & . & 0 \\ 1 & -2 & 1 & . & . & . & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & 1 & -2 \end{bmatrix}\begin{bmatrix} V_1 \\ V_2 \\ . \\ . \\ V_{N-1} \end{bmatrix} + \frac{1}{h^2}\begin{bmatrix} V_0 \\ 0 \\ . \\ 0 \\ V_N \end{bmatrix}$$

i.e as

$$\frac{d\underline{V}(t)}{dt} = A\,\underline{V}(t) + \underline{b} \qquad (7.2)$$

where $\underline{V}(t) = [V_1, V_2, ...., V_{N-1}]^T$, $\underline{b}$ is a column vector of zeroes and known boundary values, and matrix A, of order (N-1), is given by:

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & . & . & . & 0 \\ 1 & -2 & 1 & . & . & . & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & 1 & -2 \end{bmatrix} \qquad (7.3)$$

## System Stiffness

7.4 The system being solved is now examined, to demonstrate the high stiffness ratios that can result from reducing a p.d.e. to a set of o.d.e.'s. The stiffness ratio of the eigenvalues of the system matrix A, for large N, is given by [68]:

$$\sin^2\left(\frac{(N-1)\Pi}{2N}\right) / \sin^2\left(\frac{\Pi}{2N}\right) \approx \frac{4N^2}{\Pi^2} \qquad (7.4)$$

If for example N = 100, then the stiffness ratio of the system is given by 4.052 x $10^3$, and it can be seen that a stiff system can occur when an accurate solution to the p.d.e. in the x-direction is required. Consequently, to ensure the stability of the results, care must be taken when choosing which numerical scheme to apply.

## Applying The New Method

7.5 Before applying the new method to the system of equations given by equation (7.2), it is worthwhile examining this system and studying the way in which it is normally solved. Analagous to the true solution of the scalar version of equation (7.2), found by separation of variables, the solution of the system of equations given in equation (7.2) will be of the same form, with the exponential of scalars replaced by the exponentials of matrices. Consequently, the numerical methods that are normally used to solve equation (7.2) will attempt to approximate an exponentiated matrix, and the way in which this is done is important in determining the behaviour of the method. The new method was based on the true solution of the scalar form of equation (7.2), but when applied to systems of equations the method can also only attempt to approximate the exponential of a matrix. Most methods are formed by considering the Pade approximations to $e^{\theta}$, with $\theta$ real, which will be explained in detail.

**Forming A Solution For $\underline{V}(t)$**

**7.6** This section shows the form of the true solution for equation (7.2), under the conditions that are considered in this chapter. The formula presented will give the true solution at each time step. The solution of equation (7.2) satisfying the initial condition

$\underline{V}(0) = [g_1, g_2, ..., g_{N-1}]^T = \underline{g}$, and in the particular case where $\underline{b}$ is constant is shown in the next section to be

$$\underline{V}(t) = -A^{-1}\underline{b} + \{\exp(tA)\}(\underline{g} + A^{-1}\underline{b}) \qquad (7.5)$$

hence

$$\underline{V}(t+k) = -A^{-1}\underline{b} + \{\exp(t+k)A\}(\underline{g} + A^{-1}\underline{b})$$

$$= -A^{-1}\underline{b} + \{\exp(kA)\}\{\exp(tA)\}(\underline{g} + A^{-1}\underline{b})$$

where, as has already been defined, $k = \Delta t$

This leads to, using equation (7.5),

$$\underline{V}(t+k) = -A^{-1}\underline{b} + \{\exp(kA)\}(\underline{V}(t) + A^{-1}\underline{b}) \qquad (7.6)$$

and if furthermore, all the boundary values are zero, then

$$\underline{V}(t+k) = \{\exp(kA)\}\underline{V}(t) \qquad (7.7)$$

**The Solution Of $d/dt\underline{V}(t) = A\underline{V}(t) + \underline{b}(t)$**

**7.7** This section derives the true solution for equation (7.2), under certain initial and boundary conditions. The exponential of the real (nxn) matrix P is defined by

$$\exp P = e^P = I_n + P + \frac{P^2}{2!} + \frac{P^3}{3!} + ... = \sum_{m=0}^{\infty} \frac{P^m}{m!} \qquad (7.8)$$

where $P^0 = I_n$ is the unit matrix of order n

Since

$$e^P e^{-P} = e^{-P} e^P = e^0$$

and by equation (7.8)

$$e^0 = I_n$$

then

$$e^P e^{-P} = I_n \qquad (7.9)$$

Premultiplying both sides of equation (7.9) by the inverse $(e^P)^{-1}$ of $e^P$, shows that

$$e^{-P} = (e^P)^{-1}$$

Putting P =At into equation (7.8), where the matrix A is independent of t, and differentiating with respect to t, it follows that

$$d/dt(e^{At}) = Ae^{At} = e^{At}A \qquad (7.10)$$

Now considering $\underline{V}(t) = e^{At}\underline{g}$, where $\underline{g}$ is independent of time. This clearly satisfies the initial condition $\underline{V}(0) = \underline{g}$. Differentiation w.r.t. t gives that

$$\frac{d\underline{V}}{dt} = Ae^{At}\underline{g} = A\underline{V}$$

Hence, the solution of

$$\frac{d\underline{V}}{dt} = A\underline{V}$$

which satisfies $\underline{V}(0) = \underline{g}$ is

$$\underline{V}(t) = e^{At}\,\underline{g} \qquad (7.11)$$

Similarly, the vector function

$$\underline{V}(t) = -A^{-1}\underline{b} + e^{At}(\,\underline{g} + A^{-1}\underline{b}) \qquad (7.12)$$

which satisfies the initial condition $\underline{V}(0) = \underline{g}$, is the solution of

$$\frac{d\underline{V}}{dt} = A\underline{V} + \underline{b}$$

provided that vector $\underline{b}$ and matrix A are independent of t.

**Standard Finite Difference Approximations**

**7.8** This section examines the present numerical techniques that are employed in the solution of equation (7.2), by forming an estimate to equation (7.7). In order to derive a set of finite difference equations from equation (7.7), it is necessary to approximate the exponential of kA by a finite algebraic function of kA. Since exp(kA), by definition, is

$$I + kA + \frac{1}{2}k^2A^2 + \frac{1}{6}k^3A^3 + \dots$$

then one immediate approximation is I + kA, with a leading error term of order $k^2$. The vector of values $\underline{u} = [u_1, u_2, \dots, u_{N-1}]^T$ approximating V in equation (7.7) will then be the solution of the finite difference equations

$$\underline{u}(t+k) = (I + kA)\underline{u}(t) \qquad (7.13)$$

If $t = t_j = jk$ and $r = k/h^2$, these equations are, for zero boundary values

$$\begin{bmatrix} u_{1,j+1} \\ u_{2,j+1} \\ \cdot \\ \cdot \\ u_{N-1,j+1} \end{bmatrix} = \begin{bmatrix} 1-2r & r & 0 & . & . & . & 0 \\ r & 1-2r & r & . & . & . & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & . & . & r & 1-2r \end{bmatrix} \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ \cdot \\ \cdot \\ u_{N-1,j} \end{bmatrix}$$

and the approximation given above is Euler's method, applied to a system of equations. This method is also known as the simple explicit method when used in a p.d.e. context.

**7.9 Pade approximations.** This section defines Pade approximants to $e^{\theta}$, and demonstrates how most numerical methods, when applied to the scalar test equation

$$y' = \lambda y \quad y(0) = 1$$

will attempt to approximate the solution $e^{\lambda t}$ by forming a Pade approximant to the exponential. Assume that $e^{\theta}$ is to be approximated by

$$\frac{(1+p_1\theta)}{(1+q_1\theta)}$$

where $p_1$ and $q_1$ are constants. The Pade approximant is the rational function whose Maclaurin's series expansion agrees with that of $e^\theta$ to as many terms as possible. The determination of $p_1$ and $q_1$ requires two equations, which will come from the coefficients of $\theta$ and $\theta^2$, so the leading error term will be of the order $\theta^3$. Hence

$$e^\theta = \frac{1+p_1\theta}{1+q_1\theta} + c_3\theta^3 + c_4\theta^4 + \ldots$$

Therefore,

$$(1+q_1\theta)(1+\theta+1/2\theta^2+1/6\theta^3 + \ldots) = 1 + p_1\theta + (1+q_1\theta)(c_3\theta^3+c_4\theta^4+\ldots)$$

and thus

$$(1 + q_1 - p_1)\theta + (1/2 + q_1)\theta^2 + (1/6 - 1/2q_1 - c_3)\theta^3 + O(h^4) + \ldots = 0$$

This equation is satisfied uniquely in terms of order three by

$$p_1 = \frac{1}{2},\ q_1 = -\frac{1}{2},\ c_3 = -\frac{1}{12}$$

The rational approximation

$$\frac{(1+\frac{1}{2}\theta)}{(1-\frac{1}{2}\theta)}$$

is called the (1,1) Pade approximant of order 2 to $e^\theta$, and has a leading error term of order 3. In general it is possible to approximate $e^\theta$ by

$$e^\theta = \frac{1+p_1\theta+p_2\theta^2+\ldots+p_S\theta^S}{1+q_1\theta+q_2\theta^2+\ldots+q_T\theta^T} + c_{S+T+1}\theta^{S+T+1} + O(\theta^{S+T+2}) \qquad (7.14)$$

where $c_{S+T+1}$ is a constant.

The rational function given by

$$R_{S,T} = \frac{1 + p_1\theta + p_2\theta^2 + \ldots + p_S\theta^S}{1 + q_1\theta + q_2\theta^2 + \ldots + q_T\theta^T} = \frac{P_S(\theta)}{Q_T(\theta)} \qquad (7.15)$$

is called the (S,T) Pade approximant of order (S+T) to $e^{\theta}$. Table 7.1 gives eight of the Pade approximants to $e^{\theta}$, and their leading error terms.

**7.10 The Crank-Nicolson method.** The Pade approximants can be used to find many different numerical methods. For example, the (1,0) Pade approximant will replace equation (7.7), which is:

$$\underline{Y}(t+k) = \{\exp(kA)\}\underline{Y}(t)$$

by

$$\underline{u}(t+k) = (I + kA)\underline{u}(t)$$

which is the explicit method given by equation (7.13)

Similarly, the (1,1) Pade approximant replaces equation (7.7) by

$$\underline{u}(t+k) = (I - 1/2kA)^{-1}(I + 1/2kA)\underline{u}(t) \qquad (7.16)$$

For numerical calculations, this needs to be written as

$$(I - 1/2kA)\underline{u}(t+k) = (I + 1/2kA)\underline{u}(t)$$

and this gives the Crank-Nicolson scheme, which is

$$-ru_{i-1,j+1} + 2(1+r)u_{i,j+1} - ru_{i+1,j+1} = ru_{i-1,j} + 2(1-r)u_{i,j} + ru_{i+1,j} \qquad (7.17)$$

$$i = 1, \ldots, N-1$$

This method is second order accurate in t, since its error term via a Pade approximant is of order $k^3$. It has better stability properties than the method given by equation (7.13) [68]. However, it can produce unwanted finite oscillations near points of discontinuity, and this is demonstrated later in the chapter. The fully implicit method, given by:

$$\underline{u}(t+k) = (I - kA)^{-1}\underline{u}(t)$$

is only first order accurate, but does have improved stability properties over the Crank-Nicolson method, and does not demonstrate the same problems near points of discontinuity.

## The New Method

**7.11** The new method does not form a Pade approximant to $e^{\lambda t}$ when applied to the scalar equation $y' = \lambda y$, but actually forms the exponential $e^{\lambda t}$, i.e. the estimate to the solution it gives is

$$y_{n+1} = y_n e^{\lambda h} \tag{7.18}$$

However, when solving systems of differential equations, the method does not form a Pade approximant to $e^{Ah}$, but will form a different approximation. The approximation it makes is first order accurate in t, but is different from the one given by equation (7.13). This idea is discussed further when trying to improve the accuracy of the method, where an attempt is made to combine solutions formed at the half time step and full time step by the new method, in order to form an approximation which is of higher order than one.

**7.12 Applying the new explicit method.** Stability is guaranteed with the new explicit method if the system matrix of the problem to be solved is strictly diagonally dominant. In this section the stability matrix of the explicit method for the system given by equation (7.2) is examined to demonstrate that the eigenvalues of this matrix all have magnitude less than, or equal to one. Besides the system examined here, there are several other parabolic p.d.e.'s which will lead to diagonally dominant system matrices, if they are reduced to a set of ordinary differential equations. One example is the equation given by:

$$U_t = \alpha U_{xx} - \beta U \qquad \alpha > 0, \beta > 0$$

which will lead to a strictly diagonally dominant system matrix.

Applying the new method to each equation of the system given by (7.2) will lead to the set of equations

$$u_{1n+1} = \frac{u_{0n} + u_{2n}}{-2}(e^{-2r} - 1) + u_{1n}e^{-2r}$$

$$u_{2n+1} = \frac{u_{1n} + u_{3n}}{-2}(e^{-2r} - 1) + u_{2n}e^{-2r}$$

$$\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}$$

$$u_{N-1n+1} = \frac{u_{N-2n} + u_{Nn}}{-2}(e^{-2r} - 1) + u_{N-1n}e^{-2r}$$

taking the boundary conditions as

$$u_0 = 0 = u_N$$

will lead to the matrix equation

$$\begin{bmatrix} u_{1n+1} \\ u_{2n+1} \\ \cdot \\ \cdot \\ u_{N-1n+1} \end{bmatrix} = \begin{bmatrix} \gamma & \delta & 0 & . & . & . & 0 \\ \delta & \gamma & \delta & . & . & . & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & 0 & \delta & \gamma & \delta \\ 0 & 0 & 0 & . & . & \delta & \gamma \end{bmatrix} \begin{bmatrix} u_{1n} \\ u_{2n} \\ \cdot \\ \cdot \\ u_{N-1n} \end{bmatrix} \qquad (7.19)$$

where

$$\gamma = e^{-2r}$$

$$\delta = (1-e^{-2r})/2$$

**7.13 Stability of the new method.** Referring to section 4.20, the stability matrix of the explicit method for this problem is given by:

$$\begin{bmatrix} e^{-2r} & \frac{1}{2}(1-e^{-2r}) & 0 & . & . & . & 0 \\ \frac{1}{2}(1-e^{-2r}) & e^{-2r} & \frac{1}{2}(1-e^{-2r}) & . & . & . & 0 \\ . & . & e^{-2r} & . & . & . & . \\ . & . & . & . & \frac{1}{2}(1-e^{-2r}) & e^{-2r} & \frac{1}{2}(1-e^{-2r}) \\ 0 & 0 & 0 & . & 0 & \frac{1}{2}(1-e^{-2r}) & e^{-2r} \end{bmatrix} \qquad (7.20)$$

which is of the form:

$$\begin{bmatrix} a & b & 0 & . & . & . & 0 \\ c & a & b & . & . & . & 0 \\ 0 & c & a & b & . & . & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & c & a & b \\ 0 & 0 & 0 & . & . & c & a \end{bmatrix} \qquad (7.21)$$

and, since the N-1 eigenvalues of the tridiagonal (N-1 x N-1) matrix given by equation (7.21) are:

$$\lambda_s = a + 2b(c/b)^{0.5}\cos(s\Pi/N) \quad s = 1, 2, \ldots, N-1$$

then the eigenvalues of the (N-1 x N-1) matrix in equation (7.20) are given by:

$$\lambda_{M_e} = e^{-2r} + (1 - e^{-2r})\cos(m\Pi/N) \quad m = 1, 2, \ldots, N-1 \qquad (7.22)$$

r will always be positive, so the largest value that $e^{-2r}$ can take is less then 1. Substituting in some real numbers, then, since $r = k/h^2$, taking k = 0.01 and h = 0.01 will give r = 100. The eigenvalues for the stability matrix are then given by:

$$\lambda_{M_e} = 1.3839 \times 10^{-87} + (1 - 1.3839 \times 10^{-87})\cos(m\Pi/N) \quad m= 1,\ldots,N-1$$

Cos(x) lies between -1 and 1, and hence the modulus of $\lambda_{M_e} \leq 1$ for all m.

**7.14 Results.** Figure 7.1 shows the analytical solution at t = 0.25 seconds to the problem

$$U_t = U_{xx} \quad 0 < x < 1, \ t > 0$$

satisfying

$$U(0,t) = U(1,t) = 0 \quad t > 0$$

$$U(x,0) = 1 \quad 0 \leq x \leq 1, \ t = 0$$

Also shown are the results given by the Crank-Nicolson method, and the unwanted oscillations near the points of discontinuity for this method are demonstrated. For the Crank-Nicolson method, then $h = 0.025$ and $k = 0.025$, giving $r = 40$. The results given by the new method in solving the same problem are shown in Figures 7.2 and 7.3. The results shown in Figure 7.3 have been found with $h = 0.05$ and different vlaues of k. The explicit method given in equation (7.13) was also used to solve this problem, but the results proved to be graphically unpresentable if $r \geq 1/2$, since instability occurs. For very small values of k, needed to ensure that $r \leq 1/2$, then the results were close to the true solution.

## Discussion Of Results And Improving The Accuracy Of The Method

**7.15** Although a small time step is needed to ensure high accuracy using the first order method, instability is not demonstrated for any value of k, and this is an interesting observation. For this method, the time step is not limited by stability considerations, but by accuracy, and this is a worthwhile point to notice. The inaccuracy of the method has led to efforts aimed at improving the order of the method, and this is the basis for the work presented in the remainder of the chapter. Before solving the systems of o.d.e.'s arising in this chapter, the poor accuracy of the new method, although it was known, was acceptable since the results given by new method, for most hydraulic circuits studied in this thesis, were good in comparison to other low order, highly stable numerical methods. However, for the problems that it has been used to solve in this chapter, an increase in the accuracy of the method would certainly prove to be advantageous, and consequently, the rest of this chapter explains two approaches that have been taken to try and increase the order of the method. First, an extrapolation approach is adopted by considering the method when applied to the problem given by equation (7.2), and secondly, the construction of the method from first principles, as demonstrated in chapter 3, is reconsidered to see if there is a possibility of improving the accuracy in that way.

## Extrapolation Approach

**7.16** The true solution of equation (7.2), under the conditions considered in this chapter, is given by equation (7.7), which is:

$$\underline{V}(t+k) = \{\exp(kA)\}\underline{V}(t)$$

If the exponential is approximated by the new method, then the solution vector found by the method will be:

$$\underline{u}(t+k) = M_e\underline{u}(t) \qquad (7.23)$$

where $M_e$ is the matrix given in equation (7.20)

Over a time interval of 2k, this would be

$$\underline{u}^{(1)}(t+2k) = M_f\underline{u}(t) \qquad (7.24)$$

where $M_f$ is of the same form as $M_e$, with k replaced by 2k

Alternatively, the application of equation (7.23) twice, each over a time interval of k, leads to the equation

$$\underline{u}^{(2)}(t+2k) = M_eM_e\underline{u}(t),$$

i.e

$\underline{u}^{(2)}(t+2k) = (M_e)^2\underline{u}(t)$ (7.25)

7.17 The Maclaurin expansion of exp(2kA) in

$\underline{V}(t+2k) = \{\exp(2kA)\}\underline{V}(t)$

gives

$\underline{V}(t+2k) = \{I + 2kA + 2k^2A^2\}\underline{V}(t) + O(k^3)$ (7.26)

and to form a more accurate solution, then it may be possible to combine the solutions given by equations (7.24) and (7.25) to create a formula that is accurate to terms of order $k^2$. Unfortunately, there is no easy way of applying this idea, and so the term

$\{I + 2kA + 2k^2A^2\}$

must be evaluated in full. This term, along with $M_f$ and $(M_e)^2$, is:

$$I + 2kA + 2k^2A^2 = \begin{bmatrix} d_1 & d_2 & d_3 & 0 & . & . & 0 \\ d_2 & d_4 & d_2 & d_3 & . & . & . \\ d_3 & d_2 & d_4 & d_2 & d_3 & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & d_3 & d_2 & d_4 & d_2 \\ 0 & . & . & . & d_3 & d_2 & d_1 \end{bmatrix} \qquad (7.27)$$

where $d_1 = 1 - 4r + 10r^2$

$d_2 = 2r - 8r^2$

$d_3 = 2r^2$

$d_4 = 1 - 4r + 12r^2$

$$(M_e)^2 = \begin{bmatrix} e_1 & e_2 & e_3 & \cdot & \cdot & \cdot & 0 \\ e_2 & e_1 & e_2 & e_3 & \cdot & \cdot & \cdot \\ e_3 & e_2 & e_1 & e_2 & e_3 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & e_3 & e_2 & e_1 & e_2 \\ 0 & \cdot & \cdot & \cdot & e_3 & e_2 & e_1 \end{bmatrix} \tag{7.28}$$

where $e_1 = e^{-4r} + 1/4(e^{-4r} - 2r + 1)$

$e_2 = -e^{-2r}(e^{-2r} - 1)$

$e_3 = 1/4(e^{-4r} - 2r + 1)$

$$M_f = \begin{bmatrix} f_1 & f_2 & 0 & \cdot & \cdot & \cdot & 0 \\ f_2 & f_1 & f_1 & \cdot & \cdot & \cdot & \cdot \\ 0 & f_2 & f_1 & f_2 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & f_2 & f_1 & f_2 \\ 0 & \cdot & \cdot & \cdot & \cdot & f_2 & f_1 \end{bmatrix} \tag{7.29}$$

where $f_1 = e^{-4r}$

$f_2 = 1/2(1 - e^{-4r})$

Expanding the exponential terms in $M_f$ and $(M_e)^2$ up to $r^2$ will give the same matrices as those in equations (7.28) and (7.29), with the $e_i$'s and the $f_i$'s now being given by:

$f_1 = 1 - 4r + 8r^2$ $\qquad$ $e_1 = 1 - 4r + 7r^2$

$f_2 = 2r - 4r^2$ $\qquad$ $e_2 = 2r - 6r^2$

$e_3 = -r^2$

**7.18** To form a more accurate method, the matrices $M_f$ and $(M_e)^2$ must be combined to form the matrix, or an estimate to the matrix, given in equation (7.27). The combination that has been used is given by $3M_f - 2(M_e)^2$, and this is termed $M_g$. $M_g$ is of the same form as the matrix given in equation (7.27), with the $d_i$'s now being given by

$$d_1 = 1 - 4r + 10r^2$$

$$d_2 = 2r$$

$$d_3 = 2r^2$$

$$d_4 = 1 - 4r + 12r^2$$

The elements of $M_g$ are identical to those in the matrix given by equation (7.27), apart from the off-diagonal elements given by:

$$M_{g_{i,i+1}} \quad i = 1, N\text{-}1$$

$$M_{g_{i+1,i}} \quad i = 1, N\text{-}1$$

which are each missing a $-8r^2$ term in comparison with the $d_i$'s given for equation (7.27). This may prove to be important when applying the method that the matrix $M_g$ corresponds to, which is:

$$3\underline{u}^{(1)}(t+2k) - 2\underline{u}^{(2)}(t+2k) \qquad (7.30)$$

**7.19 Applying the method of extrapolation.** The method given by equation (7.30) was applied to the problem defined in section 7.14, and the results at $t = 0.25$ are shown in Figure 7.4, for $h = 0.025$, and different values of k. The accuracy of the extrapolated method is an improvement on the accuracy of the explicit method alone, but still is not as accurate as would be hoped. This could be due to the missing terms of the elements of $M_g$. Consequently, a new approach was adopted in an attempt to improve the accuracy of the method, and this is explained in the next section.

**Alternative Approach To Improve The Accuracy Of The Method**

**7.20** The approach studied here returns to the original formulation of the method. The true solution of the linear first order differential equation

$$y' = a(t)y + b(t) \qquad y(t_0) = \alpha$$

as given in equation (3.2) is:

$$y(t) = e^{\int_{t_0}^{t} a(z)dz}\left[y(t_0) + \int_{t_0}^{t} b(s)\, e^{-\int_{t_0}^{s} a(x)dx} ds\right]$$

The new method was originally formulated by considering both a and b as constant functions. This section investigates the possibility of obtaining a higher order method by approximating the integrals in equation (3.2) by a more accurate process than has been previously adopted.

**7.21** On the interval $[t_n,t_{n+1}]$, then a half-point is introduced, and so the interval becomes $[t_n,t_{n+1/2}]$, $[t_{n+1/2},t_{n+1}]$. The explicit method is then employed to obtain a solution at $t_{n+1/2}$ and $t_{n+1}$. Equation (3.2) is now employed to produce new values at $t_{n+1/2}$ and $t_{n+1}$, with the integrals being evaluated using the trapezoidal rule. Simpson's rule, derived in section 2.34, is now used to estimate the integrals in equation (3.2), and so a third and final approximation will be made to the solution $y(t_{n+1})$.

**7.22** Using the trapezoidal rule to find the solution at $t_{n+1/2}$ requires equation (3.2) to be written as:

$$y(t_{n+1/2}) = e^{\int_{t_n}^{t_{n+1/2}} a(z)dz}\left[y(t_n) + \int_{t_n}^{t_{n+1/2}} b(s)\, e^{-\int_{t_n}^{s} a(x)dx} ds\right] \qquad (7.31)$$

with

$$e^{\int_{t_n}^{t_{n+1/2}} a(z)dz} = e^{\frac{\Delta t}{4}[a(t_n) + a(t_{n+1/2})]} \tag{7.32}$$

and

$$\int_{t_n}^{t_{n+1/2}} b(s)\, e^{-\int_{t_n}^{s} a(x)dx} = \frac{\Delta t}{4}\left[\, b(t_n)\, e^{-\int_{t_n}^{t_n} a(x)dx} + b(t_{n+1/2}) e^{-\int_{t_n}^{t_{n+1/2}} a(x)dx} \right]$$

$$= \frac{\Delta t}{4}\,[b(t_n) + b(t_{n+1/2})\{\, e^{-\frac{\Delta t}{4}(a(t_n) + a(t_{n+1/2}))} \}\,] \tag{7.33}$$

Suppressing suffices, then equations (7.32) and (7.33) may be written as:

$$e^{\int_{t_n}^{t_{n+1/2}} a(z)dz} = e^{\frac{\Delta t}{4}[a_n + a_{n+1/2}]} \tag{7.34}$$

and

$$\int_{t_n}^{t_{n+1/2}} b(s)\, e^{-\int_{t_n}^{s} a(x)dx} = \frac{\Delta t}{4}\,[\, b_n + b_{n+1/2}( e^{-\frac{\Delta t}{4}\{a_n + a_{n+1/2}\}} )\,] \tag{7.35}$$

Substituting equations (7.34) and (7.35) into equation (7.31), and again suppressing suffices will give:

$$y_{n+1/2} = e^{\frac{\Delta t}{4}[a_n + a_{n+1/2}]}\{\, y_n + \frac{\Delta t}{4}[\, b_n + b_{n+1/2}( e^{-\frac{\Delta t}{4}(a_n + a_{n+1/2})} )\,]\,\} \tag{7.36}$$

which will reduce to:

$$y_{n+1/2} = e^{\frac{\Delta t}{4}[a_n + a_{n+1/2}]}\{\, y_n + \frac{\Delta t}{4} b_n \,\} + \frac{\Delta t}{4} b_{n+1/2} \tag{7.37}$$

Using a similar process to find the solution at $t_{n+1}$ leads to the scheme

$$y_{n+1} = e^{\frac{\Delta t}{4}[a_n + 2a_{n+1/2} + a_{n+1}]} \{ y_n + \frac{\Delta t}{4} b_n \} + \frac{\Delta t}{2} b_{n+1/2}\, e^{\frac{\Delta t}{4}[a_{n+1/2} + a_{n+1}]} + \frac{\Delta t}{4} b_{n+1} \qquad (7.38)$$

**7.23** When Simpson's rule is used to find a further approximation to $y(t_{n+1})$, then the resultant numerical scheme will be:

$$y_{n+1} = e^{\frac{\Delta t}{6}[a_n + 4a_{n+1/2} + a_{n+1}]} \{ y_n + \frac{\Delta t}{6} b_n \} + \frac{2\Delta t}{3} b_{n+1/2} e^{-\frac{\Delta t}{12}[ a_n - 5a_{n+1/2} - 2a_{n+1}]} + \frac{\Delta t}{6} b_{n+1}$$

and so the three numerical schemes that will be used in addition to the explicit method have been formed, and hence it is possible to find

$y^{(1)}_{n+1/2}, y^{(1)}_{n+1}, y^{(2)}_{n+1/2}, y^{(2)}_{n+1}$ and the final solution at $t_{n+1}$, $y^{(3)}_{n+1}$

where the superscripts denote the successive estimations to the solution.

**7.24 Results.** Figure 7.5 shows the results obtained when applying the technique explained above to the problem defined in section 7.13. Again, the results at t = 0.25 are given, for h = 0.025, and different values of k. Unfortunately, the results do not prove to be as accurate as had been hoped. When the method described in section 7.21 was analysed, where the integrals were approximated using the trapezium rule, then the local error was found to include a term involving $Ur^3$, which is likely to be relatively large with comparison to the solution U(x,t). The local error expression is given by:

$$\text{L.E.} = \frac{k}{8} U_{tt} - \frac{r^3}{12} U - \frac{r^3}{8} h^2 U_{xx} \qquad (7.39)$$

and it can be seen that the second term is undesirable. Consequently, this technique for improving the accuracy of the method has not been pursued, since it appears to introduce parasitic terms into the local error, which will stop the local error from being reduced in magnitude. This was not the required result.

## Conclusions

**7.25** The work that has been described in this chapter did not prove to be as rewarding as had been hoped, although one interesting aspect of the work still provides a point of discussion. This aspect is the stability of the method, and when applied to solve the problem in section 7.13, the explicit method did not demonstrate instability for any value of r, although a small time step was required in order to form a sufficiently accurate estimate to the true solution. For traditional explicit methods, when applied to stiff systems, the step size employed during the integration is determined by stability considerations. In the case of the new method, the step size is determined by the local error, i.e the accuracy of the method. This is an interesting and notable finding.

| (S,T) | $R_{S,T}(\theta)$ | Principal Error Term |
|---|---|---|
| (1,0) | $1 + \theta$ | $\frac{1}{2}\theta^2$ |
| (2,0) | $1 + \theta + \frac{1}{2}\theta^2$ | $\frac{1}{6}\theta^3$ |
| (0,1) | $\frac{1}{1 - \theta}$ | $-\frac{1}{2}\theta^2$ |
| (1,1) | $\frac{1 + \frac{1}{2}\theta}{1 - \frac{1}{2}\theta}$ | $-\frac{1}{12}\theta^3$ |
| (2,1) | $\frac{1 + \frac{2}{3}\theta + \frac{1}{6}\theta^2}{1 - \frac{1}{3}\theta}$ | $-\frac{1}{72}\theta^4$ |
| (0,2) | $\frac{1}{1 - \theta + \frac{1}{2}\theta^2}$ | $\frac{1}{6}\theta^3$ |
| (1,2) | $\frac{1 + \frac{1}{3}\theta}{1 - \frac{2}{3} + \frac{1}{6}\theta^2}$ | $\frac{1}{72}\theta^4$ |
| (2,2) | $\frac{1 + \frac{1}{2}\theta + \frac{1}{12}\theta^2}{1 - \frac{1}{2}\theta + \frac{1}{12}\theta^2}$ | $\frac{1}{720}\theta^5$ |

**TABLE 7.1 PADE APPROXIMANTS TO $e^\theta$ , $\theta$ real**

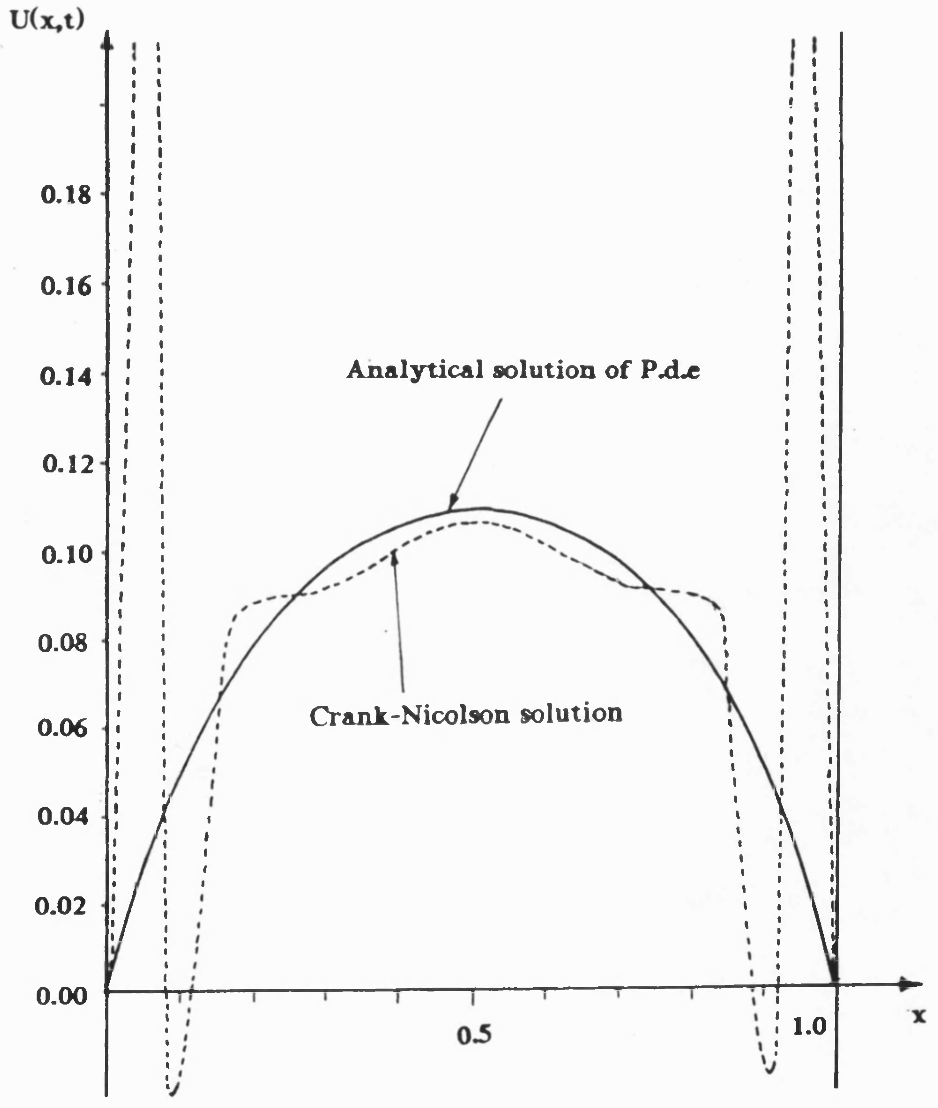


FIGURE 7.1 SOLUTION OF $U_t = U_{xx}$ AT t = 0.25 secs

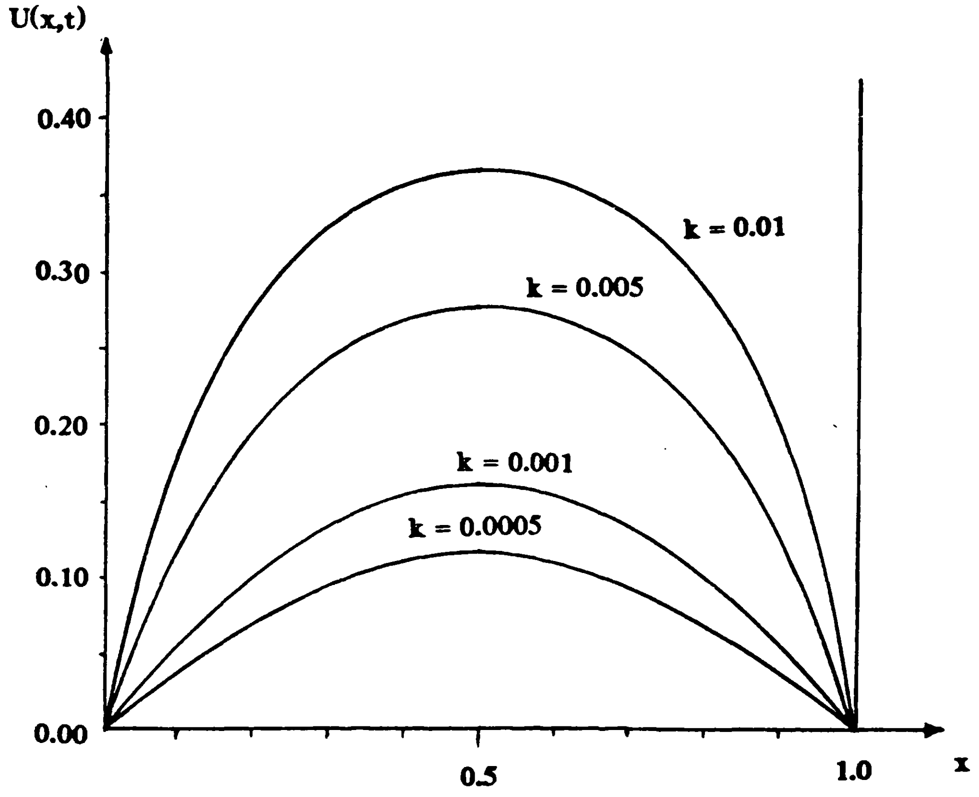


FIGURE 7.2 SOLUTION OF $U_t = U_{xx}$ AT t = 0.25 secs USING THE NEW EXPLICIT METHOD WITH $\delta x = 0.025$

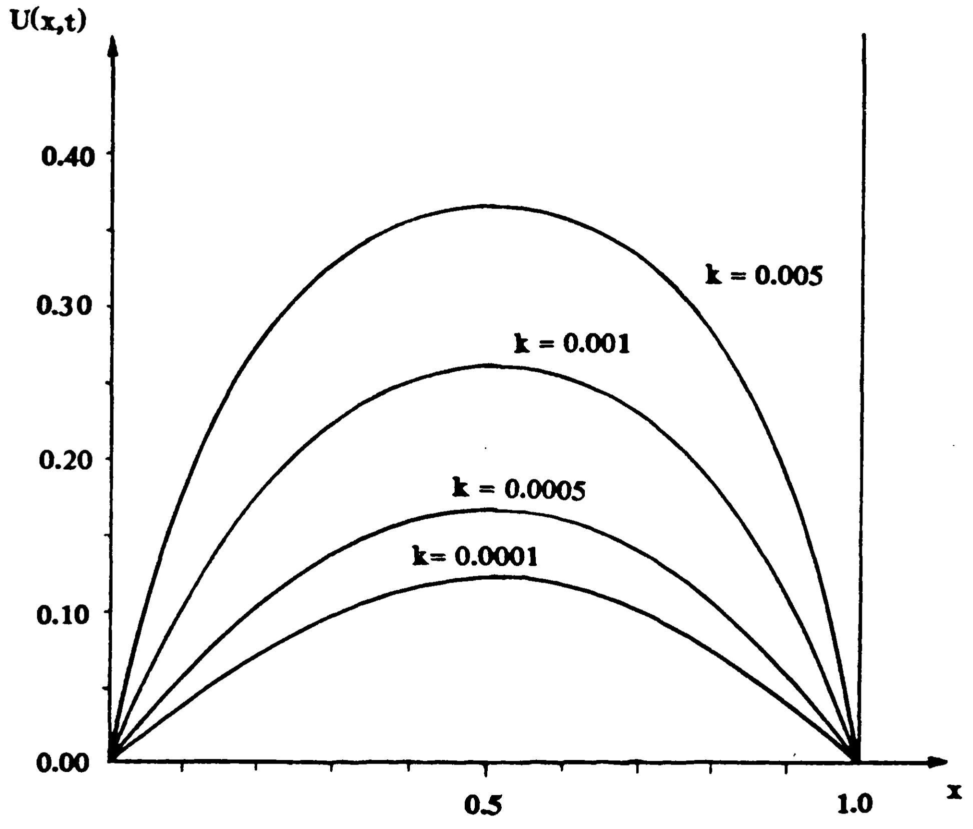


FIGURE 7.3 SOLUTION OF $U_t = U_{xx}$ AT t = 0.25 secs USING THE NEW EXPLICIT METHOD WITH $\delta x = 0.05$

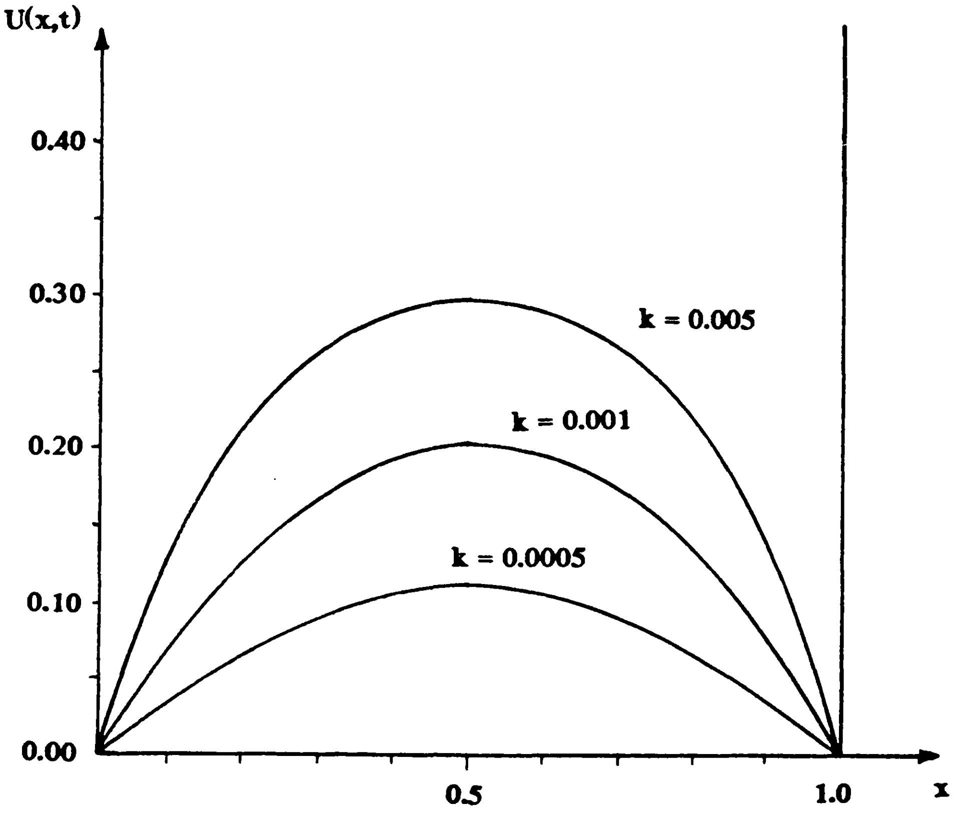


FIGURE 7.4 SOLUTION OF $U_t = U_{xx}$ AT t = 0.25 secs USING THE METHOD OF EXTRAPOLATION WITH $\delta x = 0.025$

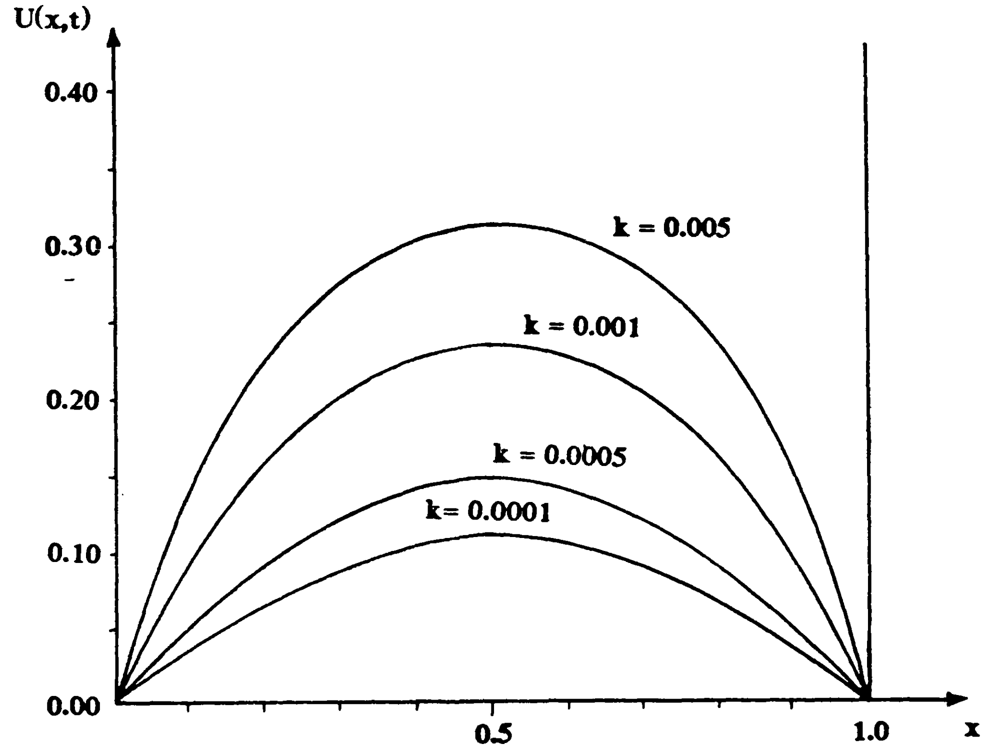


FIGURE 7.5 SOLUTION OF $U_t = U_{xx}$ AT t = 0.25 secs USING THE SECOND APPROACH WITH $\delta x = 0.025$

# CHAPTER 8

**DETAILED CONTENTS** **Page**

# CHAPTER 8

# RUNGE-KUTTA AND SWITCHING METHODS

## Introduction

**8.1** This chapter looks at Runge-Kutta methods, in particular implicit Runge-Kutta methods, and studies their potential as an alternative set of single-step methods for use as an integrator in the HASP package. First, Runge-Kutta methods are defined, and some of the many different formulae that arise, each with their particular application, are discussed. Both explicit and implicit Runge-Kutta methods are described, and then a particular class of implicit Runge-Kutta methods is discussed, namely diagonally implicit, and the merits of this class when applied to hydraulic simulation explained. The implementation of Runge-Kutta methods, both explicit and implicit, in HASP is discussed, and the potential benefits explored. Finally, the idea of switching methods [21] is investigated. This idea, which is at the forefront of research with integration methods for stiff systems, is presented in the form of strategic decision-making criteria that operate throughout the integration process; enabling the numerical method employed to be replaced by another method when the pre-set criteria so determine.

**8.2** Runge-Kutta methods aim to produce equivalent accuracy to a Taylor's series method, without the need to compute high order derivatives. Instead, a combination of f(t,y) values taken at several points in the interval $[t_n,t_{n+1}]$ is used. The methods obtain high order accuracy by sacrificing linearity, but they do retain the valuable one step nature of single-step methods. The methods were proposed by Runge [38] and subsequently developed by Kutta [39] and Heun [40]. Although the methods retain the advantages of single-step methods, the loss of linearity leads to more difficult error analysis than in the case of linear multi-step methods.

## Derivation Of Explicit Runge-Kutta Methods

**8.3** The general explicit R-stage Runge-Kutta method is defined by:

$$y_{n+1} - y_n = h\phi(t_n, y_n, h) \qquad (8.1)$$

where

$$\phi(t,y,h) = \sum_{r=1}^{R} c_r k_r \qquad (8.2)$$

$$k_1 = f(t,y)$$

$$\cdot \quad \cdot$$

$$k_r = f(t+ha_r,\ y+h\sum_{s=1}^{r-1} b_{rs}k_s),\quad r = 2, 3, \ldots, R \qquad (8.3)$$

$$a_r = \sum_{s=1}^{r-1} b_{rs},\quad r = 2, 3, \ldots, R \qquad (8.4)$$

and $y_{n+1}$ and $y_n$ are the respective numerical approximations to $y(t_{n+1})$ and $y(t_n)$. An R-stage Runge-Kutta method involves R function evaluations at each step. Each of the functions $k_r(t,y,h)$, $r = 1, 2, \ldots, R$, may be interpreted as an approximation to the derivative $y'(t)$, and the function $\phi(t,y,h)$ as a weighted mean of these approximations.

**8.4** The true solution will satisfy

$$y(t_{n+1}) = y(t_n) + h\phi(t_n, y(t_n), h) + E_{n+1} \qquad (8.5)$$

The idea is to find the a's, c's and b's, so that the Taylor's series expansion of equation (8.5) in ascending powers of h agrees with the Taylor expansion

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + (h^2y''(t_n))/2 + \ldots \qquad (8.6)$$

to as many terms as possible for a given value of R.

**8.5** The Taylor's series expansion of equation (8.5) will involve the use of a Taylor's series expansion in two dimensions, i.e.

$$f(t+\alpha,y+\beta) = e^{\alpha\partial/\partial x + \beta\partial/\partial y} f(t,y)$$

Introducing the notation

$$f = f(t,y),\ f_t = \partial f(t,y)/\partial t,\ \ f_{tt} = \partial^2 f(t,y)/\partial t^2,\ \ f_{ty} = \partial^2 f(t,y)/\partial t\partial y,\ \text{etc}$$

and noting that

$$y''(t,y) = d/dtf(t,y) = \partial f/\partial t + \partial f/\partial y dy/dt = f_t + f_y f$$

then the general formula for $R = 2$ will now be developed.

**8.6 Derivation of two-stage Runge-Kutta methods.** This section derives the formula for two-stage Runge-Kutta methods and demonstrates the existence of an infinite number of two-stage, second order Runge-Kutta methods. The general formula for a two-stage Runge-Kutta method is given by:

$$y_{n+1} = y_n + h(c_1k_1 + c_2k_2) \qquad (8.7)$$

where

$$k_1 = f(t_n,y_n)$$

$$k_2 = f(t_n+ha_2,y_n+b_{21}hk_1)$$

The true solution will satisfy

$$y(t_{n+1}) = y(t_n) + h(c_1K_1 + c_2K_2) + E_{n+1} \qquad (8.8)$$

where

$$K_1 = f(t_n,y(t_n))$$

$$K_2 = f(t_n+ha_2,y(t_n)+b_{21}hK_1)$$

Expanding $K_2$, using a Taylor's series expansion of two variables, will give:

$$K_2 = f + (a_2hf_t + b_{21}hK_1f_y) + \frac{1}{2!}(a_2^2h^2f_{tt} + 2a_2hb_{21}hK_1f_{ty} + b_{21}^2h^2K_1^2f_{yy}) + O(h^3) \qquad (8.9)$$

where f and its derivatives are evaluated at $(t_n, y(t_n))$.

Substituting equation (8.9) into equation (8.8) will give:

$$y(t_{n+1}) = y(t_n) + (c_1+c_2)hf + h^2c_2a_2f_t + h^2c_2b_{21}ff_y + O(h^3) \qquad (8.10)$$

This expansion is then compared with the Taylor's series expansion about $y(t_{n+1})$, given by:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + (h^2y''(t_n))/2! + (h^3y'''(t_n))/3! + O(h^4)$$

$$= y(t_n) + hf + h^2(f_t+ff_y)/2 + h^3(f_{tt}+2ff_{ty}+f^2f_{yy}+f_tf_y+ff_y{}^2)/6 + O(h^4) \qquad (8.11)$$

The comparison of equation (8.11) with equation (8.10) shows that an agreement of terms in $h^0$, $h^1$, $h^2$ can be obtained with

$$c_1+c_2 = 1, \;\; c_2a_2 = 1/2, \;\; c_2b_{21} = 1/2 \qquad (8.12)$$

There are three equations in four unknowns, and hence there is a free parameter. Except in trivial cases this parameter cannot be used to match terms in $h^3$, and there is therefore an infinite number of two-stage, second order, Runge-Kutta methods. For R-stage explicit methods, with $R > 1$, there is always at least one free parameter. Table 8.1 shows the maximum possible order of accuracy obtained with R-stage methods for $R \leq 8$.

## Stability Of Runge-Kutta Methods

**8.7** To study the stability properties of Runge-Kutta methods, the test differential equation, $y' = \lambda y \;\; y(0) = 1$, first mentioned in section 2.45, can be used. When the two-stage Runge-Kutta method given by equation (8.7) is applied to the test o.d.e., then $k_1$ and $k_2$ become:

$$k_1 = \lambda y_n$$

$$k_2 = \lambda(y_n + b_{21}hk_1)$$

and hence equation (8.7) can be rewritten as:

$$y_{n+1} = y_n + h(c_1\lambda y_n + c_2(\lambda(y_n + b_{21}h\lambda y_n)))$$

i.e.

$$y_{n+1} = y_n + h(c_1 + c_2)\lambda y_n + h^2c_2\lambda^2b_{21}y_n \qquad (8.13)$$

Using the values given in equation (8.12) will reduce equation (8.13) to:

$$y_{n+1} = y_n + h\lambda y_n + (h^2\lambda^2y_n)/2 \qquad (8.14)$$

Hence, for stability, the requirement is that

$$\left|1 + h\lambda + \frac{h^2\lambda^2}{2}\right| < 1$$

i.e.

$$-2 < h\lambda < 0 \quad \text{(for } \lambda \text{ real)} \qquad (8.15)$$

**8.8** Figure 8.1 shows the stability regions for explicit Runge-Kutta methods of order 1, 2, 3 and 4. Unlike multi-step methods, the stability region increases for methods as the order of accuracy increases. Although the stability region for the methods of order four is larger than that for order one, a restriction on hλ is still made, and this handicaps the methods when they are used to solve very stiff systems. However, the computational efficiency involved in using explicit Runge-Kutta methods rather than implicit multi-step methods often compensates for this restriction in steplength.

**Runge-Kutta Merson**

**8.9** Because of the difficulty involved in forming theoretical error estimates for Runge-Kutta methods, it is also hard to construct reliable step control strategies. Often, the time step is controlled using an extrapolation process, explained in section 4.12. This however, is an inefficient means of monitoring the local error since each step must be repeated once, if not several times. Merson [78] proposed a means of estimating the local error for a five-stage, fourth order method that does not require the re-computation of the solution at any point during the integration. He showed that if $f(t,y) = Lt + My + N$, with L, M, N constant, i.e. $f(t,y)$ is linear, with constant coefficients, then an error estimate can be formed at each time step using different weightings of the $k_i$ coefficients, $i = 1, 2, 3, 4, 5$. Merson's method is given by:

$$y_{n+1} = y_n + h(k_1 + 4k_4 + k_5)/6 \tag{8.16}$$

where

$$k_1 = f(t_n,y_n)$$

$$k_2 = f(t_n+h/3,y_n+hk_1/3)$$

$$k_3 = f(t_n+h/3,y_n+hk_1/6+hk_2/6)$$

$$k_4 = f(t_n+h/2,y_n+hk_1/8+2hk_3/8)$$

$$k_5 = f(t_n+h,y_n+hk_1/2-3hk_3/2+2hk_4)$$

and

$$z_{n+1} = y_n + h(k_1 - 3k_3 + 4k_4)/2 \tag{8.17}$$

The local error for the two schemes, for a linear problem of the form quoted above, is given by using the following relations:

$$z_{n+1} = y(t_{n+1}) - \frac{h^5 y^{(5)}(t_n)}{120} + O(h^6) \tag{8.18}$$

$$y_{n+1} = y(t_{n+1}) - \frac{h^5 y^{(5)}(t_n)}{720} + O(h^6) \qquad (8.19)$$

i.e. the local error of equation (8.16) is given by

$$E_{n+1} \approx \frac{h^5 y^{(5)}(t_n)}{720} = \frac{y_{n+1} - z_{n+1}}{5}$$

Hence, at each step, $\frac{|y_{n+1} - z_{n+1}|}{5}$ is monitored. If it is too large, then h is reduced; if it is much smaller than the pre-defined tolerance, then h is increased.

**8.10** Care must be taken when applying the linearity assumption in order to use Merson's method, since experiments have shown that the local error estimate found when using this assumption often grossly overestimates, and occasionally underestimates $E_{n+1}$ [74]. The author has assisted with the implementation of the method in HASP, a full description of which is given by Wang [74]. Good results have been obtained with the circuits that do not give rise to constant high ratios of mathematical stiffness. An alternative form of step control that has also been implemented in the package, in use with other Runge-Kutta methods, has been developed by Fehlberg [79] [80].

**Runge-Kutta Fehlberg**

**8.11** Fehlberg's method of step control is to use 6 k's to produce a fifth order method, and a subset of these k's to produce a fourth order method, the fifth order method being for error purposes. For the five-stage Runge-Kutta method given by:

$$y_{n+1} = y_n + h\sum_{i=1}^{5} c_i k_i \qquad (8.20)$$

where the coefficients are chosen to give a fourth order method; then if a six-stage, fifth order method of the form

$$z_{n+1} = y_n + h\sum_{i=1}^{6} c_i^* k_i \tag{8.21}$$

is also used to form a solution, $\varepsilon = \left| z_{n+1} - y_{n+1} \right|$ can be taken as an estimate to the local error in forming $y_{n+1}$ using equation (8.20). This is because, if the true solution satisfies

$$y(t_{n+1}) = y(t_n) + \sum_{i=1}^{5} c_i k_i + E_{n+1} \tag{8.22}$$

then, using the localising assumption of exact values at $t_n$, and subtracting equation (8.20) from (8.22) and equation (8.21) from an equation similar to (8.22) to give:

$$E_{n+1} = y(t_{n+1}) - y_{n+1} \quad E_{n+1} = O(h^5) \tag{8.23}$$

and

$$E^*_{n+1} = y(t_{n+1}) - z_{n+1} \quad E^*_{n+1} = O(h^6) \tag{8.24}$$

the subtraction of equation (8.24) from (8.23) will lead to:

$$z_{n+1} - y_{n+1} = E_{n+1} - E^*_{n+1} \tag{8.25}$$

and so, the principal local error term for the fourth order method is given by:

$$z_{n+1} - y_{n+1} \tag{8.26}$$

**Implicit Runge-Kutta Methods**

**8.12** The chapter now goes on to discuss implicit Runge-Kutta methods. For non-stiff problems, and for oscillatory problems [88] explicit Runge-Kutta methods are generally suitable. However, the stability regions of the explicit methods indicate that it is not advantageous to use them when the problem being solved is likely to be stiff over a large proportion of the integration interval. Implicit Runge-Kutta methods are particularly applicable to stiff systems since it is possible to obtain A-stable implicit Runge-Kutta formulae of the type discussed by Butcher [18] [81] of arbitrarily high order [82], whereas the order of an A-stable linear multi-step method is limited to 2 [83]. Consequently, high order accuracy may be obtained, as well as the stability properties that will ensure that for an inherently stable system, the method is stable for any step size. Implicit Runge-Kutta methods can be written in the form [20]:

$$y_{n+1} = y_n + h\sum_{i=1}^{R} c_i f(t_n + a_i h, y_{n,i}) \tag{8.27}$$

where

$$y_{n,i} = y_n + h\sum_{j=1}^{R} b_{ij} f(t_n + a_j h, y_{n,j}) \tag{8.28}$$

**8.13** The formulae given by equations (8.27) and (8.28) can be characterised by displaying their coefficients as a Butcher matrix of the form

$$\begin{array}{cccccc}
b_{11} & b_{12} & \cdots & b_{1R} & a_1 \\
b_{21} & b_{22} & \cdots & b_{2R} & a_2 \\
\cdot & \cdot & \cdots & \cdot & \cdot \\
\cdot & \cdot & \cdots & \cdot & \cdot \\
\cdot & \cdot & \cdots & \cdot & \cdot \\
b_{R1} & b_{R2} & \cdots & b_{RR} & a_R \\
c_1 & c_2 & \cdots & c_R &
\end{array} \tag{8.29}$$

It can be seen that equation (8.27) will lead to an explicit method if $b_{ij} = 0$ for $i \leq j$. Equation (8.27) is said to give a semi-implicit method if $b_{ij} = 0$ for $i < j$, and it will give an implicit method if $b_{ij} \neq 0$ for either of these cases. The computational effort involved in using a semi-implicit Runge-Kutta method is in general considerably less than that which is required by a fully implicit Runge-Kutta formula [20]. A particularly efficient class of semi-implicit Runge-Kutta formulae was first suggested by Norsett [19] who considered the case where the $b_{ii}$ are all equal and non-zero. These formulae were further studied by Crouziex [84] and by Alexander [20]. They were termed Diagonally Implicit Runge-Kutta Formulae (DIRK) by Alexander. Before defining classes of semi-implicit Runge-Kutta methods, which is the purpose of this chapter, an additional stability requirement often needed by numerical methods for very stiff systems is defined.

## S-Stability

**8.14** A-stability, where the stability region of a method includes the whole of the left-hand $\lambda h$ plane, is not the whole answer to the problem for stiff equations. In their work with large systems of stiff, non-linear equations, Prothero and Robinson [85] found that the A-stability of a method is no guarantee that it will give stable solutions, and that the accuracy of the solutions obtained often appears to be unrelated to the order of the method used. The problem lies in the choice of the test equation, and the standard test problem, first discussed in section 2.45, is not suitable for drawing decisive conclusions with some types of problem. The analysis by Prothero and Robinson has led to the introduction of a new stability concept.

**Definition [85]** A Runge-Kutta method is S-stable, if for any bounded function $g:[0,T] \rightarrow \Re$ having a bounded derivative, and any positive constant $\lambda_0$, there is a positive constant $h_0$ such that the numerical solution $(y_n)$ to the equation

$$y' = g'(t) + \lambda(y - g(t)) \tag{8.30}$$

satisfies

$$\left| \frac{y_{n+1} - g(t_{n+1})}{y_n - g(t_n)} \right| < 1 \tag{8.31}$$

provided $y_n \neq g(t_n)$, for all $0 < h < h_0$, and all complex $\lambda$, with $\mathrm{Re}(-\lambda) \geq \lambda_0$.

A Runge-Kutta method is strongly S-stable if

$$\frac{y_{n+1} - g(t_{n+1})}{y_n - g(t_n)} \rightarrow 0$$

as $\mathrm{Re}(-\lambda) \rightarrow \infty$ for all h such that $[t_n,t_{n+1}]$ is a subset of $[0,T]$.

S-stability $\Rightarrow$ A-stability, since if g is taken to be zero, equation (8.30) will reduce to the standard test equation, and equation (8.31) reduces to the normal conditions for A-stability. The converse does not hold.

## Diagonally Implicit Runge-Kutta Methods

**8.15** To integrate a system of m differential equations, an implicit method with a full matrix given by equation (8.29) requires the solution of mR simultaneous, non-linear equations at each time step. One way to overcome this problem is to use a lower triangular matrix $(b_{ij})$ in equation (8.28). The formulae given by equation (8.28) may then be solved in R successive stages, with only an m-dimensional system to be solved at each stage. As has been already discussed already, such methods are termed semi-implicit. In solving equation (8.28) successively by a Newton-type iteration scheme, which is suitable for stiff systems due to there being no stipulation on a combination of the Lipschitz constants of the system, and the time step, in order to ensure convergence, then a linear system is solved at each stage with a Jacobian matrix of the form $I - hb_{ii}\, \partial \underline{f}/\partial \underline{y}$ involved. If all the $b_{ii}$ are equal, then the LU-factorisation of the single matrix can be stored and used repeatedly. Crouziex [84] has determined all the two-stage, third order and three-stage fourth order semi-implicit Runge-Kutta methods, and from his work the following theorems have been extracted by Alexander [20].

**Theorem 1 [20]** There is exactly one A-stable DIRK formula each of two-stage, third order and three-stage fourth order. These are given by:

Two-stage, third order

$$\begin{array}{lll} 1/2 + 1/(2\sqrt{3}) & 0 & 1/2 + 1/(2\sqrt{3}) \\ -1/\sqrt{3} & 1/2 + 1/(2\sqrt{3}) & 1/2 - 1/(2\sqrt{3}) \\ 1/2 & 1/2 & \end{array} \qquad (8.32)$$

Three-stage fourth order

$$\begin{array}{llll} (1+\alpha)/2 & 0 & 0 & (1+\alpha)/2 \\ -\alpha/2 & (1+\alpha)/2 & 0 & 1/2 \\ (1+\alpha) & -(1+2\alpha) & (1+\alpha)/2 & (1-\alpha)/2 \\ 1/(6\alpha)^2 & 1-1/3\alpha^2 & 1/(6\alpha)^2 & \end{array} \qquad (8.33)$$

where $\alpha = 2\cos\{(\pi/18)/\sqrt{3}\}$

**Theorem 2** There is no DIRK four-stage formula with fifth order accuracy.

**8.16** The two methods given by equations (8.32) and (8.33) are also S-stable, the proof for this is given by Prothero and Robinson [85]. Alexander also presents two strongly S-stable DIRK formula of order two in two stages and one strongly S-stable DIRK formula of order three in three stages. These are:

$$\begin{array}{ccc} \alpha & 0 & \alpha \\ 1-\alpha & \alpha & 1 \\ 1-\alpha & \alpha & \end{array} \qquad \text{where } \alpha = 1 \pm (\sqrt{3})/2 \qquad (8.34)$$

and

$$\begin{array}{cccc} \alpha & 0 & 0 & \alpha \\ a_2-\alpha & \alpha & 0 & a_2 \\ c_1 & c_2 & \alpha & 1 \\ c_1 & c_2 & \alpha & \end{array} \qquad \begin{array}{c} \text{where } \alpha \text{ is the root of} \\ x^3 - 3x^2 + 3x/2 - 1/6 = 0 \\ \text{lying in the interval } (1/6, 1/2) \end{array} \qquad (8.35)$$

$a_2 = (1 + \alpha)/2$

$c_1 = -(6\alpha^2 - 16\alpha + 1)/4$

$c_2 = (6\alpha^2 - 20\alpha + 5)/4$

**8.17** Although Alexander developed the formulae for several diagonally implicit Runge-Kutta methods, he did not develop an efficient time step control to be used with the methods. The technique he employed was Richardson's extrapolation. However, Alexander did achieve good results when he applied the DIRK formulae given in equations (8.32) - (8.35) to several of the test problems devised by Enright and given in Appendix D. He compared the results he found using the DIRK formulae to those given by the Hindmarsh-Gear method [86]. Some of the results that he found are given in tables 8.2, 8.3 and 8.4. It can be seen that, even when using a poor time step control, the DIRK methods display considerable advantage over an adapted version of Gear's method for the test problems selected.

**8.18 Diagonally implicit Runge-Kutta formulae with error estimates.** Cash [87] developed a class of embedded diagonally implicit strongly S-stable Runge-Kutta methods based on those formed by Alexander with an additional facility, a ready-formed estimate of the local error at each step that entails virtually no extra computational cost. These methods have been implemented in the HASP package, and a description of these methods and their application to hydraulic systems is now given

**8.19** Cash's idea was to use the approach first proposed by Fehlberg, which has been discussed in section 8.11. He found embedded methods and from these could form an approximation to the local error at each step. For the coefficient values given in equation (8.35), leading to a three-stage third order method, there is an embedded two-stage, second order formula given by the coefficient values:

$$\begin{array}{lll} \alpha & \alpha & \alpha \\ a_2-\alpha & 0 & a_2 \\ c_1 & c_2 & \end{array} \qquad (8.36)$$

where $\alpha$ and $a_2$ are given in equation (8.35)

$$c_1 = (a_2 - 1/2)/(a_2 - \alpha)$$

$$c_2 = (\alpha - 1/2)/(\alpha - a_2)$$

The important point to note about these embedded formulae is that virtually no extra work is required to compute the second order solution once the third order solution has been computed. This is because the quantities

$$f(t_n+a_ih, y_{n,i}) \quad i = 1, 2$$

will already have been computed.

**8.20 Fourth order strongly S-stable formula** It is a five-stage, fourth order method that has been implemented in HASP. The coefficient values for this method are given by:

$$\begin{array}{cccccc}
\alpha & 0 & 0 & 0 & 0 & a_1 \\
b_{21} & \alpha & 0 & 0 & 0 & a_2 \\
b_{31} & b_{32} & \alpha & 0 & 0 & a_3 \\
b_{41} & b_{42} & b_{43} & \alpha & 0 & a_4 \\
c_1 & c_2 & c_3 & c_4 & \alpha & 1 \\
c_1 & c_2 & c_3 & c_4 & \alpha &
\end{array} \qquad (8.37)$$

where

|  |  |
|---|---|
| $\alpha = 0.4358665215$ | $b_{21} = -1.1358665215$ |
| $b_{31} = 1.085433307$ | $b_{32} = -0.721299828$ |
| $b_{41} = 0.416349502$ | $b_{42} = 0.190984004$ |
| $b_{43} = -0.118643265$ | $a_1 = \alpha$ |
| $a_2 = -0.7$ | $a_3 = 0.8$ |
| $a_4 = 0.924556762$ | $c_1 = 0.896869653$ |
| $c_2 = 0.018272527$ | $c_3 = -0.0845900311$ |
| $c_4 = -0.266418671$ |  |

8.21 The coefficient values for the embedded, third order formula which is used for the purposes of error estimation are:

$$\begin{array}{ccccc} \alpha & 0 & 0 & 0 & a_1 \\ b_{21} & \alpha & 0 & 0 & a_2 \\ b_{31} & b_{32} & \alpha & 0 & a_3 \\ b_{41} & b_{42} & b_{43} & \alpha & a_4 \\ c_1 & c_2 & c_3 & c_4 & \end{array} \tag{8.38}$$

where $\alpha$, the $a_i$'s and the b coefficients are given above, and

$c_1 =$ 0.776691933
$c_2 =$ 0.029747279
$c_3 =$ -0.026744024
$c_4 =$ 0.220304812

**8.22 Estimating the local truncation error.** A proceedure is now considered for estimating the local truncation error of the formula given by equation (8.37), and for controlling the steplength used by the method during the integration. Supposing that the finally accepted approximation $y_n$ to $y(t_n)$ has been computed, and the requirement is to compute an approximate solution at $t_{n+1} = t_n + h$. Assuming that $y_n$ is exact, and denoting the solution obtained at $t_{n+1}$ using the embedded formula in equation (8.38) by

$$\overset{1}{y}_{n+1}$$

and that obtained at at $t_{n+1}$ using the formula given in equation (8.37) by

$$\overset{2}{y}_{n+1}$$

then an estimate $E^*$ to the local truncation error of the asymptotically less accurate $\overset{1}{y}_{n+1}$ is:

$$E^* = \overset{2}{y}_{n+1} - \overset{1}{y}_{n+1} \tag{8.39}$$

If $|E^*|$ is less than a prescribed tolerance, then the solution $y^2_{n+1}$ is carried forward as the approximation to the true solution.

**8.23 Step-control.** If a local error tolerance, TOL, is imposed at each step, and h and h' are the current steplength and the next steplength to be chosen respectively, then if E is set as

$$E = | y^2_{n+1} - y^1_{n+1} | \tag{8.40}$$

the steplength is controlled in the following way, although the algorithm discussed in chapter 4 can also be employed.

i) If $E > TOL$, $h' = h/2$ and start again from $t_n$
ii) If $TOL/\mu < E < TOL$, $h' = h$
iii) If $E < TOL/\mu$, $h' = 2h$

Here the factor $\mu$ is introduced, to make sure that the steplength is not doubled too often when it is not safe to do so. The choice of $\mu$ is somewhat arbitrary, but practical experience based on third and fourth order formulae have shown that for the cases where the order of the method, p, is 3 or 4, the choice $\mu = 2^p + 2^{p+1}$ is adequate [87].

**Implementation of DIRK methods in HASP**

**8.24** Since the method has been formed to solve o.d.e.'s of the form

$$\underline{y}' = \underline{f}(t,\underline{y})$$

the implementation inside of HASP is a straightforward procedure, since the package has been designed for a classical integration method, namely Gear's method. The calling process from the main program MAIN to the Runge-Kutta method, which has been writen in FORTRAN as a subroutine called DIRK, is:

```
CALL DIRK(T, TEND, Y, N, AUX, TOL, ITEST, TAB, OUT, IFAIL)
```

where the arguments are the same as those explained in some detail in chapter 5. The function values needed by DIRK are again returned by the subroutine AUX when the numerical integrator requires them. This method has been used to solve the fifth order linear actuator circuit which is described in secion 5.6. At present the method has only been used experimentally, although the results it has obtained compare favourably with the results given by Gear's method and the new method. Many tests remain if DIRK methods are to provide alternative integrators to be used within HASP. Some difficulties were encountered when using the DIRK methods, and the major one was choosing the starting values for $y_{n,i}$ given in equation (8.28). The method of choice is not simple, since Newton's iteration technique requires an accurate starting value, and several approaches have been adopted. Originally, a linear extrapolation approach was used, but this did not prove to be very successful. The method used at present, which is not ideal, is to take as the starting values the final values calculated at the last time step. Although this process works satisfactorily, an improved method for determining the starting values of the $y_{n,i}$ coefficients would decrease the C.P.U. time needed by the method, since a large proportion of the time spent in the integration routine is taken up with the evaluation of these coefficients. The DIRK method described here provides a realistic alternative method for solving the stiff systems of ordinary differential equations that arise in Fluid Power simulation because of its superior stability properties and its high order of accuracy. The work that is necessary before this method, or one similar, can be employed as a general purpose integration method by HASP, would provide an interesting topic of work for the future.

## Automatic Selection Of Methods For Solving Stiff And Non-Stiff Systems Of O.D.E.'s

**8.25** The work undertaken with the HASP simulation package has identified particular problem areas. The main problem is mathematical stiffness, but this generally proves to present difficulties only throughout certain parts of a simulation, i.e. only a small proportion of the total simulation time actually gives difficulty to the numerical integrator employed by the package. Recent work by Petzold [21], Robertson [22] and Richards, Wade and Everett [88] has explored the possibility of implementing a scheme, when solving o.d.e.'s, that automatically determines whether a problem can be solved more efficiently using a class of methods suited for non-stiff problems or a class of methods suited for stiff problems. This chapter now goes on to investigate the work undertaken with switching methods, which has been mostly involved with multi-step methods, and explains the possible application to the HASP package. The background behind the decision-making techniques is given, along with a suggestion for a possible extension of the idea to encorporate different Runge-Kutta formulae as the "stiff methods" and "non-stiff methods".

**8.26** Petzold [21] has devised a technique that uses the information available at the end of each step during the integration for making the decision between employing one of two different types of method. If a problem changes character in the interval of integration, the solver automatically switches to the class of methods which is likely to be the most efficient for that part of the problem. The results she found, using a modified version of Enright's package, indicate that many problems can be solved more efficiently using this scheme than a single class of methods, and that the overhead of choosing the most efficient method is relatively small, in comparison to the complete run time for the problems solved.

**8.27 Motivation for switching methods.** A scheme that switches methods would be useful in several different situations. The user of an o.d.e. solver may not know whether the problem to be solved is stiff, or the solver may be called by another code, such as with HASP, where the character of the problem is not known in advance. Using a "switching" technique, then the most effective family of methods is chosen automatically. Moreover, many stiff problems in Fluid Power simulation are often non-stiff after the initial phase, or transient. Integrating through the time period after the transient with a method designed for stiff problems is very expensive, whereas methods designed for non-stiff problems are much better suited for this purpose. As the problem becomes non-stiff, the code can eventually switch to "non-stiff methods". In general, a problem may be stiff in some intervals and non-stiff in others. A "switching" scheme selects the methods that are most efficient for each interval.

**8.28** Shampine [89] [90] [91] originally undertook the work involved with detecting stiffness. He also outlined a scheme [92] for automatically altering the solution algorithm based on the stiffness of the problem for codes implementing A-stable formulae. However, the work undertaken recently has not restricted the automatic selection of a method to be necessarily A-stable, and this provides much more freedom when choosing the relevant sets of methods.

**8.29 Basic strategy for choosing methods.** As the integration proceeds, the objective is to choose the family of methods which will solve a given problem most efficiently. This decision was made by Petzold by comparing the method that is currently being used to the method that would be used if the code switched to the other family of methods. LSODE, an updated version of the GEAR package, written by Hindmarsh [93], has been modified to switch automatically between Adam's methods and backward differentiation formulae depending on the nature of the problem being solved. This chapter presents the idea of using Runge-Kutta formulae for both the non-stiff and the stiff methods. To compare the methods, the step size that each method could use on the next step must be considered, and also the cost per step of each method. Since one step of a "non-stiff method" is typically much cheaper than one step of a "stiff method", it is favourable to use a "non-stiff method" as long as the step sizes it uses are not very much smaller than the step sizes that would be used by a "stiff method".

**8.30 Controlling the step size for each method.** For the "non-stiff methods" several considerations will affect the step size. First, a step size must be chosen to ensure that the formula is accurate over the next step, i.e. so that the norm of the local truncation error is less than some constant $\varepsilon$. If the code uses functional iteration to solve the corrector equation, rather than Newton's method, then the step size must be small enough so that the iteration will converge rapidly. Finally, the step size must be small enough to ensure that the method is stable. For the "stiff method", the step size is chosen so that the formula is accurate over the next step. Letting N be the "non-stiff method", and S the "stiff method", then if N and S are both linear multi-step methods of order p, for example, the requirement can be made that the principal part of the local truncation error is less than $\varepsilon$. Suppose that method N is currently being used with step size $h_C$, that N is stable for the problem with this step size, and that $||\ E_N\ ||$ is the estimate for the local truncation errror. Then, applying the technique described in chapter 4, $h_N$ and $h_S$, the step sizes that the non-stiff and stiff methods could use on the next step, will satisfy

$$h_N \le \left(\frac{\varepsilon}{||E_N||}\right)^{\frac{1}{p+1}} h_C \tag{8.41}$$

$$h_S \le \left(\frac{\varepsilon(C_N/C_S)}{||E_N||}\right)^{\frac{1}{p+1}} h_C \tag{8.42}$$

where $C_N$ and $C_S$ are constants depending on the methods N and S.

**8.31** The step size of method N may also be affected by considerations such as stability and convergence of the functional iterations. Hence, what effects, if any, these conditions will have, must be found. To accomplish this, an estimation for $||\, \partial \underline{f}/\partial \underline{y}\, ||$, or an estimate of the spectral radius $\rho(\partial \underline{f}/\partial \underline{y})$ is required. When a "stiff method" is used, a Jacobian matrix is available and $||\, \partial \underline{f}/\partial \underline{y}\, ||$ can be computed directly. The norm can be computed cheaply, relative to the cost of matrix factorisation, whenever a new Jacobian matrix is formed. Thus, the norm which is available at any given time may correspond to a time several steps back, but this is not likely to be too severely in error, since "stiff methods" generally re-evaluate the Jacobian whenever it changes significantly. Shampine [91] gives a way of cheaply obtaining a lower bound for $||\, \partial \underline{f}/\partial \underline{y}\, ||$ during the corrector iteration when using the "non-stiff method". The basic idea is that if the iteration is written as:

$$\underline{y}^{(s+1)} = h\gamma \underline{f}(\underline{y}^{(s)}) + \underline{\varphi} \tag{8.43}$$

then

$$\frac{||\, \underline{y}^{(s+1)} - \underline{y}^{(s)}\, ||}{||\, \underline{y}^{(s)} - \underline{y}^{(s-1)}\, ||} \le h\gamma ||\, \partial \underline{f}/\partial \underline{y}\, || \tag{8.44}$$

The maximum of the ratios

$$(1/h\gamma)\frac{\| \underline{y}^{(s+1)} - \underline{y}^{(s)} \|}{\| \underline{y}^{(s)} - \underline{y}^{(s-1)} \|} \tag{8.45}$$

obtained over the current step, is a lower bound for $\| \partial \underline{f}/\partial \underline{y} \|$. These bounds are quite good, but can fluctuate when the dominant eigenvalues of $\partial \underline{f}/\partial \underline{y}$ are complex.

8.32 Supposing a lower bound $\kappa$ for $\| \partial \underline{f}/\partial \underline{y} \|$ has been generated or $\| \partial \underline{f}/\partial \underline{y} \|$ has been computed directly. Referring to chapter 2, then $h_N$ must be small enough so that the functional iteration will converge at a sufficiently rapid rate r, with $h_N$ satisfying $h_N\gamma \| \partial \underline{f}/\partial \underline{y} \| \leq 1/2$; so the requirement is that

$$h_N \leq r/(2\gamma \| \partial \underline{f}/\partial \underline{y} \|) \tag{8.46}$$

**8.33 Stability constraints.** Stability also constrains the step size for the non-stiff method. If $r_q$ is the radius of the largest half disc contained in the stability region of method N, then the condition

$$h_N \rho(\partial \underline{f}/\partial \underline{y}) \leq r_q \tag{8.47}$$

must be satisfied or the computation can become unstable. Hence the requirement is that $h_N$ satisfies

$$h_N \leq (r_q w_f)/\kappa \tag{8.48}$$

where $\kappa$ is the lower bound for $\| \partial \underline{f}/\partial \underline{y} \|$, and the factor $w_f$ is included to be reasonably sure that $h_N$ will lead to stable computation since $\kappa$ is only a lower bound.

**8.34 Decision-making criterion for determining which method to use.** Once the estimates for $h_N$ given by equation (8.41), (8.46) and (8.48) have been made, then the step size that the non-stiff method, N, could use on the next step, is the largest $h_N$ that satisfies these three conditions. The step size that the stiff method, S, could use is the largest $h_S$ that satisfies equation (8.42). Supposing that N is cheaper per step than S, so that it is still more economical to take as many as M steps with N for each step that have to be taken with S, then, if the method currently being used is N, the *shift* to method S would be made if

$$h_S \geq Mh_N \tag{8.49}$$

Since it is important to guard against changing families of methods too frequently, to avoid the possibility of unstable computation, it is necessary to stay with either method S or N longer than is really optimal. One way of doing this is to shift from method S to method N if

$$h_N \geq Mh_S \tag{8.50}$$

**8.35 Implementation considerations.** When practically implementing the ideas that have been described, it is important that the information upon which the code is basing its decision be reliable, and not misleading. The code should recognise situations where it is not possible to obtain reliable information, and take appropriate action. Generally, the problems that occur are involved with finding a reliable estimate for the local truncation error, and a reasonable lower bound for $\| \partial \underline{f} / \partial \underline{y} \|$. Petzold describes in detail the considerations that must be made when using linear multi-step methods for both the "stiff methods" and the "non-stiff methods". Here, a description is given of the way in which Runge-Kutta methods could be used, and the advantages that they would provide.

**Switching Methods With Runge-Kutta Formulae**

**8.36** The proposals made here are that, instead of using linear multi-step methods within both the classes of stiff and non-stiff methods, Runge-Kutta formulae are considered instead. Runge-Kutta methods have immediate advantages over multi-step methods in that integration can commence with a high order method, and the code is compact, and hence suitable for microcomputers, where the saving of storage space is important. Richards, Wade and Everett [88] suggested using explicit Runge-Kutta methods of orders 2, 3, 4 and 5 as the "non-stiff methods". For the "stiff methods" they employed backward Runge-Kutta methods of orders 1, 2, 3, 4 and 5, which were developed by Cash [94]. However, the work they presented was not very productive, in part because, although for the explicit Runge-Kutta methods it was possible to use a Fehlberg-type error estimate, with the implicit Runge-Kutta methods they employed a Richardson error estimation, which, as is discussed in chapter 4, is not an ideal method for the approximation of the local error of a method, because of the extra work involved.

**8.37** Instead of using the Runge-Kutta methods suggested above, the proposal here is that the Runge-Kutta method described in section 8.11 is employed as a non-stiff solver along with other explicit methods such as that described in section 8.9; and the Runge-Kutta method described in section 8.20 is employed as a stiff solver, along with the other formulae suggested by Cash [87] that have ready-formed error estimates. Since a reliable approximation to the local error for each of the methods in both these classes can easily be made, there exists immediately a set of highly accurate methods for use as non-stiff and stiff methods. The proposals made here present a challenging branch of future work, with which the author is already involved.

**8.38** Again, the decisions determining when to change from the non-stiff code to the stiff code and vice versa will be determined by consideration of the step size that each method would use on the next step. Since there exist reliable forms of controlling the local error, the optimal step size for each method could easily be evaluated using relations similar to those given in equations (8.41) and (8.42). Since DIRK methods can also be chosen that are strongly S-stable, there is no need to consider the stability region for this class of methods, when they are applied to inherently stable systems. However, the stability regions of the explicit methods are somewhat smaller, and hence a relation similar to that given in equation (8.48) must be used to determine an optimal h that will satisfy the stability requirements.

**8.39 Decision-making criterion for determining which Runge-Kutta method to use.** The largest h that the explicit method can use during the next step of the integration process can now be evaluated, and the largest step that the DIRK method requires will also be known. Now, it is necessary to determine, on average, how much cheaper per step the explicit method is to use than the implicit method. Then, equation (8.49) or (8.50) can be used to determine whether to shift from the non-stiff method to the stiff method, or vice versa. Since the explicit Runge-Kutta methods do not make use of an iteration scheme, the problems involved with iterating can be disregarded. When implementing the methods, it is essential to ensure that reliable information is given to the portions of code that decide what will happen next. One further consideration which may be useful when contemplating this is that, in general, when accuracy determines the length of time step, rather than stability, the problem is non-stiff; but, if stability determines the size of the time step rather than accuracy, the problem is stiff.

## Conclusions

**8.40** Having studied both Runge-Kutta methods, and the idea of "switching methods", then the merits of their implementation inside the HASP package is apparent. Explicit Runge-Kutta methods have already been successfully transplanted into the simulation package, and with some further work, implicit Runge-Kutta methods will soon be available as alternative integration methods for solving the stiff systems of differential equations arising in Fluid Power simulation. Hopefully, more work will be conducted with the package in the area of expert systems, in the sense that the subroutine solving the differential equations decides for itself which method is most suitable for an individual problem.

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Max. Order attainable | 1 | 2 | 3 | 4 | 4 | 5 | 6 | 6 |

TABLE 8.1 MAXIMUM POSSIBLE ORDER OF ACCURACY OBTAINED WITH R-STAGE RUNGE-KUTTA METHODS FOR R = 1 - 8

| | | Time | Max.Err | No. Steps | NFE | NJE |
|---|---|---|---|---|---|---|
| B1 | $\epsilon = 10^{-2}$ | | | | | |
| | DIRK(2,2) | 1.956 | 8.433E-2 | 67 | 435 | 28 |
| | DIRK(2,3) | 2.100 | 8.419E-2 | 59 | 391 | 29 |
| | DIRK(3,3) | 1.737 | 6.301E-2 | 47 | 454 | 24 |
| | GEAR | 6.013 | 4.525E-1 | 293 | 600 | 11 |
| | $\epsilon = 10^{-4}$ | | | | | |
| | DIRK(2,3) | 6.628 | 2.390E-3 | 217 | 1,364 | 37 |
| | DIRK(3,3) | 5.576 | 1.733E-3 | 163 | 1,521 | 33 |
| | DIRK(3,4) | 5.986 | 1.740E-3 | 169 | 1,586 | 35 |
| | GEAR | 7.588 | 5.884E-2 | 397 | 726 | 19 |
| | $\epsilon = 10^{-6}$ | | | | | |
| | DIRK(3,3) | 19.223 | 5.414E-5 | 542 | 4,956 | 41 |
| | DIRK(3,4) | 18.230 | 4.252E-5 | 489 | 4,496 | 45 |
| | GEAR | 13.410 | 7.598E-5 | 710 | 1,222 | 24 |

**TABLE 8.2 TEST RESULTS FOR DIRK METHODS FOUND BY ALEXANDER**

| | | Time | Max.Err | No. Steps | NFE | NJE |
|---|---|---|---|---|---|---|
| B5 | $\epsilon = 10^{-2}$ | | | | | |
| | DIRK(2,2) | 1.912 | 2.174E-2 | 52 | 342 | 15 |
| | DIRK(2,3) | 2.097 | 1.947E-2 | 47 | 313 | 15 |
| | DIRK(3,3) | 1.956 | 8.173E-3 | 39 | 376 | 14 |
| | GEAR | 44.951 | 5.502E-2 | 2,387 | 4,753 | 6 |
| | $\epsilon = 10^{-4}$ | | | | | |
| | DIRK(2,3) | 8.175 | 3.757E-4 | 191 | 1,211 | 28 |
| | DIRK(3,3) | 7.307 | 2.327E-4 | 148 | 1,393 | 27 |
| | DIRK(3,4) | 8.174 | 2.406E-4 | 151 | 1,429 | 28 |
| | GEAR | 48.029 | 4.191E-4 | 2,337 | 4,825 | 14 |
| | $\epsilon = 10^{-6}$ | | | | | |
| | DIRK(3,3) | 23.674 | 1.363E-5 | 479 | 4,408 | 31 |
| | DIRK(3,4) | 23.911 | 5.779E-6 | 457 | 4,219 | 32 |
| | GEAR | 48.648 | 8.540E-6 | 2,577 | 4,198 | 16 |

**TABLE 8.3 FURTHER TEST RESULTS FOR DIRK METHODS FOUND BY ALEXANDER**

| | | Time | Max.Err | No. Steps | NFE | NJE |
|---|---|---|---|---|---|---|
| C1 | $\epsilon = 10^{-2}$ | | | | | |
| | DIRK(2,2) | 0.656 | 2.394E-3 | 20 | 139 | 11 |
| | DIRK(2,3) | 0.631 | 1.679E-3 | 20 | 143 | 10 |
| | DIRK(3,3) | 0.751 | 3.143E-3 | 18 | 177 | 9 |
| | GEAR | 1.034 | 6.074E-3 | 57 | 101 | 13 |
| | $\epsilon = 10^{-4}$ | | | | | |
| | DIRK(2,3) | 1.941 | 3.257E-5 | 53 | 390 | 27 |
| | DIRK(3,3) | 1.653 | 8.344E-5 | 40 | 454 | 20 |
| | DIRK(3,4) | 1.716 | 6.783E-5 | 40 | 457 | 20 |
| | GEAR | 2.131 | 1.166E-4 | 112 | 186 | 20 |
| | $\epsilon = 10^{-6}$ | | | | | |
| | DIRK(3,3) | 4.802 | 4.073E-6 | 133 | 1,419 | 33 |
| | DIRK(3,4) | 4.233 | 1.266E-6 | 109 | 1,259 | 37 |
| | GEAR | 4.113 | 1.763E-6 | 206 | 289 | 27 |

**TABLE 8.4 FURTHER TEST RESULTS FOR DIRK METHODS FOUND BY ALEXANDER**

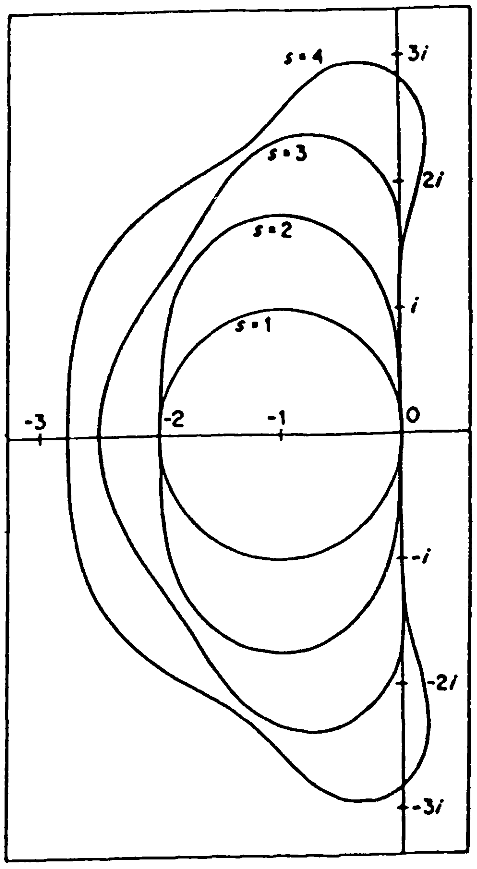


**FIGURE 8.1 STABILITY REGIONS FOR EXPLICIT RUNGE-KUTTA METHODS OF ORDER 1, 2, 3 AND 4**

# CHAPTER 9

**DETAILED CONTENTS** **Page**

# CHAPTER 9

# CONCLUSIONS

## Introduction

**9.1** A new integration method has been developed and tested in this thesis, and its application as a competitive numerical integrator has been investigated. Also, a study has been made of the mathematical problems arising in Fluid Power simulation. This chapter draws together the material presented throughout the thesis and suggests possible future work

## The Integrator

**9.2** The integrator plays a vital role within a simulation package, and the demand for efficiency increases as the systems being modelled become more sophisticated. The new method was originally studied in an attempt to improve the performance of the Hydraulic Automatic Simulation Package (HASP), when the hydraulic systems being simulated presented the problem of mathematical stiffness. The tests with the method on the particular problems described in chapter 3 led to favourable results in comparison with classical integration methods, and hence the method was studied in more detail.

## Analysis Of The Method

**9.3** After the preliminary investigations, the method was seriously contemplated; a time step control was devised and the stability properties of the method were thoroughly examined. The explicit form of the method proved to have desirable stability properties for diagonally dominant systems, and this provided the basis for the work presented in chapter 7. The implicit form of the method, which was studied, also demonstrated good stability properties, although it was not possible to form a general restraint on the integration time step in order to ensure the stability of the method. This was due to the difficulty in finding a relation between the eigenvalues of the system matrix, and the eigenvalues of the stability matrix formed for the method. However, the predictor-corrector pair, with a time step control, did provide encouraging results when applied to stiff and oscillatory problems, and so it was decided to implement the method inside the HASP package.

## Implementation In HASP

**9.4** The method was successfully implemented inside the HASP package, with the appropriate modifications made to the program generator to allow the use of the method. However, there is a problem in implementing the new method, in that the integrator requires coefficient values and not derivative values. This means that for each test circuit studied, the component models had to be changed to provide the necessary information to the new integration method. For the test circuits examined, these modifications have been made, so that the component models pass back coefficient values to the integrator. But to ensure the general use of the method as an alternative integrator with the package would require changes to be made to most of the models in the component model library, and this would provide a long and arduous task. Since the results with the test circuits examined did not prove to be as successful as had been expected, rather than modify an exhaustive set of models, a package designed specifically for testing stiff differential equation solvers was employed, which provided a large set of stiff problems. The tests carried out with this package provided the basis for the work presented in chapter 6.

**9.5** The method proved to be competitive in comparison with classical integration methods, when the problem to be solved was stiff or oscillatory. However, because of the low order of accuracy of the method, it was not capable of competing against variable order methods, when the original fast transient of a problem is dead, and the system is approaching, or has reached, steady-state. In this case, variable order methods can increase their order of accuracy, and hence the integration time step used, and still satisfy stability constraints. Hence the order of accuracy of the method had to be improved, and several approaches were adopted, two of which are discussed in chapter 7. However, so far, none of the attempts made have proved to be successful, and the accuracy of the method remains too low to allow its use as a general purpose integration method.

**9.6** The performance of the method is admirable for certain applications, and, if a way were found to improve the order of accuracy of the method, it could still provide a viable alternative as a complete integration method, both for stiff and non-stiff problems. However, its applications at present are limited, and as such the future of the method is restricted, particularly in playing a role as an integrator within the HASP package. Ways of extending the accuracy of the method could provide an interesting field of research for future work.

### Runge-Kutta And Switching Methods

**9.7** The study of the mathematical problems arising in Fluid Power simulation during the course of this project has persuaded the author to write chapter 8. It is felt that the direction forward for numerical integration methods, particularly with regard to integrators within packages such as HASP, is to employ decision-making criteria implemented within an algorithm deciding the integration method to be used. These criteria would ensure that the most suitable numerical method available is always used throughout the integration process. Since single-step methods are able to cope better with the commonly occurring mathematical difficulties, such as discontinuities, then in Fluid Power simulation the most suitable techniques with which to develop the idea of switching could be the Runge-Kutta methods. Recent work with these methods has provided numerical schemes with good stability properties and high accuracy. It is felt that this direction in future work, particularly with regard to the HASP package, is the way in which numerical integration should be headed.

# APPENDIX A

# CODE FOR THE INITIAL TESTING OF THE NEW METHOD

## Introduction

**A.1** The object of this appendix is to present the computer code implementing the new method. The commented code that is given is a code for the method operating with a fixed time step. Also included in the code are the program listings for the C.P.U. timing clock, and these are given in subroutines TIM and CONVER. The user of the code has the choice of whether to use the explicit form of the method, or the first implicit form of the method, which is combined with the explicit method to form a predictor-corrector pair. The functions that return the coefficient values for the method must be written individually for each problem, and those presented here are from the third order system discussed in section 3.35. This program also needs these functions to supply the Jacobian matrix, since a Jacobian matrix is formed specifically for each problem. In Appendix C, the code employs a perturbation technique to evaluate the Jacobian.

## Algorithm And Basic Strategy For The Program

**A.2** The flow chart representing the program action is given in Figure 3.1. The basic algorithm, in list form, is as follows:

i) The program starts
ii) The system data is input
iii) The program data is initialised
iv) The functions evaluating the coefficients are called
v) The subroutine implementing the explicit method is called
vi) If required, the subroutine implementing the implicit method is called

vii) If the implicit method is used, the functions evaluating the coefficients are recalled

viii) The solution at the present time step is formed

ix) If the time step is the last one then the program finishes, else the time step is advanced, and the program returns to iv)

```
C     THIS PROGRAM IMPLEMENTS THE NEW METHOD, WHICH CAN BE
C       APPLIED TO A VARIETY OF PROBLEMS. THE USER ENTERS THE TIME
C       STEP CHOSEN, THE FINAL TIME, WHETHER THE EXPLICIT METHOD
C       OR THE IMPLICIT METHOD IS REQUIRED AND WHETHER SCREEN
C       OUTPUT IS REQUIRED AS WELL AS GRAPHICAL OUTPUT
C*********************************************************************
C       VARIABLES IN ARGUMENT LIST
C       H      -   INTEGRATION STEP LENGTH. SET BY THE USER.
C       TEND   -   FINAL INTEGRATION TIME IN SECONDS
C       P1     -   ARRAY THAT HOLDS THE PISTON PRESSURE VALUES
C       P2     -   ARRAY THAT HOLDS THE ROD PRESSURE VALUES
C       U      -   ARRAY THAT HOLDS THE PISTON VELOCITY VALUES
C       TOL    -   TOLERANCE USED BY THE ITERATION SCHEME
C       KA,KB  -   ORIFICE FLOW COEFFICIENTS
C       B      -   BULK MODULUS OF THE HYDRAULIC OIL
C       AR1    -   PISTON-SIDE AREA OF ACTUATOR PISTON
C       AR2    -   ROD-SIDE AREA OF ACTUATOR PISTON
C       EF     -   VISCOUS FRICTION COEFFICIENT
C       M      -   MASS TO BE MOVED BY ACTUATOR
C       PS     -   SUPPLY PRESSURE
C       V1     -   INITIAL COMBINED VOLUME OF ACTUATOR AND PIPE 1
C       V2     -   INITIAL COMBINED VOLUME OF ACTUATOR AND PIPE 2
C       COUNT  -   INTEGER VARIABLE USED AS A COUNTER
C       XX     -   ARRAY USED TO STORE "A" COEFFICIENT VALUES
C       YY     -   ARRAY USED TO STORE "B" COEFFICIENT VALUES
C       ATIME, BTIME, TTIME AND TIM ARE INVOLVED IN THE C.P.U.
C       TIMING OF THE ROUTINE
C*********************************************************************
C       THE NEW METHOD APPLIED TO A NON-LINEAR, NON-OSCILLATORY
C       AND STIFF PROBLEM
```

```
C*********************************************************************
        IMPLICIT DOUBLE PRECISION (A-H, O-Z)
        DIMENSION P1(0:2000), P2(0:2000), U(0:2000), XX(3), YY(3)
        DOUBLE PRECISION KA, KB, M
        INTEGER FLAG1, FLAG2, FLAG3, COUNT
        INTEGER*4 ATIME, BTIME, TTIME, TIM
C       ENABLING GRAPHICS, USING A SPECIFIC GRAPHICS PACKAGE
1       CONTINUE
        OPEN (1, FILE=' CADRES.DAT ', ACCESS=' DIRECT ', STATUS =
      1 ' NEW ', RECL=2)
C       INITIALISING VARIABLES
        DATA U(0), P1(0), P2(0), AR1, AR2 / 0.0, 0.0, 0.0, 1.96D-3, 1.47D-3 /
        DATA B, EF, KA, KB / 1.8D9, 4.D3, 3.2D-12, 3.2D-12 /
        DATA M, PS, V1, V2 / 1.D3, 2.1D7, 1.D-3, 1.D-3 /
        COUNT = 0
        TTIME = 0
C       USER INPUT STAGE
        WRITE(6, 10)
10      FORMAT(3X, 'ENTER H AND FINAL T VALUE',/)
        READ(5,*)H,TEND
11      CONTINUE
        WRITE(6,12)
12      FORMAT(3X, 'ENTER 1 FOR EXPLICIT METHOD, 2 FOR IMPLICIT',/)
        READ(5,*)FLAG1
        IF((FLAG1 .NE. 1) .AND. (FLAG1 .NE. 2)) GOTO 11
13      CONTINUE
        WRITE(6,14)
14      FORMAT(3X, 'PRESS 1 FOR GRAPHICAL OUTPUT, 2 FOR SCREEN')
        READ(5,*)FLAG2
        IF((FLAG2 .NE. 1) .AND. (FLAG2 .NE. 2)) GOTO 13
C       INCREMENTING COUNT AS THE EVALUATION BEGINS AT THE
C       NEXT TIME STEP. AN ALTERNATIVE VERSION OF THE PROGRAM
C       ALLOWS THE USER TO EVALUATE THE EIGENVALUES OF THE
C       SYSTEM AT EACH TIME STEP IF THEY ARE REQUIRED
5       CONTINUE
        COUNT = COUNT + 1
C       CALLING THE EXPLICIT METHOD AFTER FORMING THE COEFFICIENT
C       VALUES BY INVOKING THE RELEVANT FUNCTIONS
        BTIME = TIM(ATIME)
        XX(1) = A1(B,V1,KA,PS,P1(COUNT-1))
        XX(2) = A2(B,V2,KB,P2(COUNT-1))
      XX(3) = A3(EF,M)
      YY(1) = B1(AR1,V1,B,PS,U(COUNT-1),KA,P1(COUNT-1))
      YY(2) = B2(B,AR2,V2,U(COUNT-1))
      YY(3) = B3(AR1,AR2,M,P1(COUNT-1),P2(COUNT-1))
      CALL PREDIC(P1,P2,U,XX,YY,H,COUNT)
C     IF FLAG1 = 2 THEN THE IMPLICIT METHOD IS NOW INVOKED,
C     OTHERWISE, THE VALUES CALCULATED BY THE EXPLICIT METHOD
C     ARE TAKEN AS THE SOLUTION
      IF (FLAG1 .EQ. 1) GOTO 15
      CALL CORREC(P1,P2,U,AR1,AR2,B,EF,V1,V2,PS,KA,KB,M,H,COUNT)
15    CONTINUE
      ATIME = TIM(ATIME)
      TTIME = TTIME + (ATIME- BTIME)
C     CHANGING VOLUMES IN THE ACTUATOR AS NECESSARY
      CHVOL1 = AR1*H*(U(COUNT) + U(COUNT-1))/2.
      CHVOL2 = AR2*H*(U(COUNT) + U(COUNT-1))/2.
      V1 = V1 + CHVOL1
      V2 = V2 - CHVOL2
C     THERE IS A CHECK HERE TO ENSURE THAT THE ACTUATOR
C     HAS NOT REACHED THE LIMIT OF ITS TRAVEL
      IF (V2 .LE. 0) THEN
        TEMP = FLOAT(COUNT)*H
        PRINT*,'ANNULUS VOLUME IS ZERO, RUN STOPPED AT T = ',TEMP
        GOTO 20
      ENDIF
C     CHECKING TO SEE IF THE INTEGRATION HAS FINISHED YET
      IF(COUNT .LT. INT(TEND/H)) GOTO 5
C
20    CONTINUE
C     WHEN THE SIMULATION HAS FINISHED, THE RESULTS ARE SENT
C     TO THE OUTPUT ROUTINE
      CALL CONVER(TTIME)
      CALL WRITER(P1,P2,U,H,COUNT,FLAG2)
      WRITE(6,21)
21    FORMAT(3X,' IF YOU REQUIRE A RE-RUN PRESS 1')
      READ(5,*)FLAG3
      IF (FLAG3 .EQ. 1) GOTO 1
      STOP
      END
```

```
C*******************************************************************
C     THIS IS THE SUBROUTINE WHICH EVALUATES THE SOLUTION GIVEN
C     BY THE EXPLICIT METHOD
      SUBROUTINE PREDIC(P1,P2,U,XX,YY,H,COUNT)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      DIMENSION P1(0:2000), P2(0:2000), U(0:2000), XX(3), YY(3)
      INTEGER COUNT
C     EVALUATING P1(n+1), P2(n+1), U(n+1).
      P1(COUNT) = (YY(1)/XX(1))*(DEXP(XX(1)*H) - 1.D0) + P1(COUNT-1)*
     1            DEXP(XX(1)*H)
      P2(COUNT) = (YY(2)/XX(2))*(DEXP(XX(2)*H) - 1.D0) + P2(COUNT-1)*
     1            DEXP(XX(2)*H)
      U(COUNT) = (YY(3)/XX(3))*(DEXP(XX(3)*H) - 1.D0) + U(COUNT-1)*
     1            DEXP(XX(3)*H)
      RETURN
      END
C*******************************************************************
C     THIS IS THE ROUTINE THAT EVALUATES THE SOLUTION USING THE
C     IMPLICIT METHOD. THIS ROUTINE EMPLOYS A NEWTON ITERATION
C     SCHEME, AND INCORPORATES AN N.A.G. INVERSION ROUTINE
C     F01AAF
      SUBROUTINE CORREC(P1,P2,U,AR1,AR2,B,EF,V1,V2,
     1                  PS,KA,KB,M,H,COUNT)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C     AA IS A 2-D ARRAY THAT WILL CONTAIN THE JACOBIAN MATRIX,
C     UNIT WILL CONTAIN TNE INVERSE OF THIS MATRIX AND WKSPCE
C     IS AN ARRAY NEEDED BY THE NAG ROUTINE. DUM IS AN ARRAY
C     THAT IS USED TO CHECK IF THE ITERATES ARE CONVERGING
      DIMENSION P1(0:2000), P2(0:2000), U(0:2000), AA(5,5), UNIT(5,5),
     1            WKSPCE(7), DUM(3)
      DOUBLE PRECISION KA, KB, M
C     IA, IFAIL, N AND IUNIT ARE ALL INTEGER VARIABLES THAT ARE
C     NEEDED BY THE NAG ROUTINE. IA STORES THE DIMENSION OF
C     AA, IUNIT STORES THE DIMENSION OF UNIT, AND IFAIL IS NEEDED
C     TO LET THE USER KNOW IF THE ROUTINE FAILS
      INTEGER COUNT, I, IA, IFAIL, IUNIT, N, INCREM
C     SETTING UP WORK SPACE AND DATA FOR THE NAG ROUTINE
      DATA BLANK3/'   '/, BLANK4/'    '/
      DO 101 JI = 1,6
         WKSPCE(JI) = BLANK4
```

```
101     CONTINUE
              WKSPCE(7) = BLANK3
        IA = 5
        IFAIL = 1
        IUNIT = 5
        N = 3
        TOL = 5.D-5
C       SETTING UP THE JACOBIAN MATRIX
25      CONTINUE
        VAL1 = A1(B,V1,KA,PS,P1(COUNT))
        AA(1,1) = 1.D0 + (DEXP(VAL1*H)-1.DO)*(AR1*U(COUNT))/
     1            (2.D0*KA*DSQRT(PS-P1(COUNT)))
        AA(1,2) = 0.D0
        AA(1,3) = -(DEXP(VAL1*H)-1.D0)*(AR1*SQRT(PS - P1(COUNT)))/KA
C
        VAL2 = A2(B,V2,KB,P2(COUNT))
        AA(2,1) = 0.D0
        AA(2,2) = 1.D0 + ((DEXP(VAL2*H) - 1.D0)*AR2*U(COUNT))/
     1            (2.D0*KB*SQRT(P2(COUNT)))
        AA(2,3) = (DEXP(VAL2*H) - 1.D0)*(AR2*SQRT(P2(COUNT)))/KB
C
        VAL3 = A3(EF,M)
        AA(3,1) = (DEXP(VAL3*H) - 1.D0)*AR1/EF
        AA(3,2) =  -(DEXP(VAL3*H) - 1.D0)*AR2/EF
        AA(3,3) = 1.D0
C       CALLING THE NAG ROUTINE TO INVERT THE JACOBIAN
        CALL F01AAF(AA,IA,N,UNIT,IUNIT,WKSPCE,IFAIL)
C       IF IFAIL IS STILL EQUAL TO 1 THEN A PROBLEM HAS ARISEN
        IF(IFAIL .EQ. 1) THEN
          PRINT*,' TROUBLE IN INVERTING THE MATRIX '
        ENDIF
C       SETTING THE COUNTER USED BY THE ITERATION SCHEME
        INCREM = 0
C       PERFORMING NEWTON ITERATIONS, HOLDING THE COEFFICIENT
C       VALUES CONSTANT
30      CONTINUE
C       IF THE ITERATION IS NOT CONVERGING, THEN THE COEFFICIENT
C       VALUES ARE UPDATED, AND THE JACOBIAN IS RE-EVALUATED
        IF (INCREM .GE. 5) THEN
             IF( (DUM(1) .GE. 0.005).AND.(DUM(2) .GE. 0.005) .AND.
     1            (DUM(3) .GE. 0.005)) GOTO 25
```

```
         ENDIF
      DUM(1) = P1(COUNT) - DEXP(VAL1*H)*P1(COUNT-1) -
     1   (DEXP(VAL1*H)-1.D0)*B1(AR1,V1,B,PS,U(COUNT),KA,
     1   P1(COUNT))/(VAL1)
      DUM(2) = P2COUNT) - DEXP(VAL2*H)*P2(COUNT-1) -
     1   (DEXP(VAL2*H)-1.D0)*B2(B,AR2,V2,U(COUNT))/(VAL2)
      DUM(3) = U(COUNT) - DEXP(VAL3*H)*U(COUNT-1) -
     1   (DEXP(VAL3*H)-1.D0)*B3(AR1,AR2,M,P1(COUNT),
     1   P2(COUNT))/(VAL3)
C     CHECKING TO SEE IF THE ITERATION HAS CONVERGED
      IF ((DABS(DUM(1)) .LT. TOL) .AND. (DABS(DUM(2)) .LT. TOL) .AND.
     1  (DABS(DUM(3)) .LT. TOL)) THEN
         GOTO 40
      ELSE
         DUM(1) = UNIT(1,1)*DUM(1)+UNIT(1,2)*DUM(2)+UNIT(1,3)*DUM(3)
         DUM(2) = UNIT(2,1)*DUM(1)+UNIT(2,2)*DUM(2)+UNIT(2,3)*DUM(3)
         DUM(2) = UNIT(3,1)*DUM(1)+UNIT(3,2)*DUM(2)+UNIT(3,3)*DUM(3)
         R1 = P1(COUNT)
         R2 = P2(COUNT)
         R3 = U(COUNT)
         P1(COUNT) = P1(COUNT) - DUM(1)
         P2(COUNT) = P2(COUNT) - DUM(2)
         U(COUNT)  = U(COUNT) - DUM(3)
         RATIO1 = DABS((P1(COUNT)-R1)/(1.D0+DABS(R1))
         RATIO2 = DABS((P2(COUNT)-R2)/(1.D0+DABS(R2))
         RATIO3 = DABS((U(COUNT)-R3)/(1.D0+DABS(R3))
         IF((RATIO1 .LE. TOL) .AND. (RATIO2 .LE. TOL) .AND.
     1  (RATIO3 .LE. TOL)) GOTO 40
         INCREM = INCREM + 1
C     A SECOND CHECK IS MADE ON THE INCREMENTS TO THE STATE
C     VARIABLES TO SEE IF THE ITERATIONS HAVE CONVERGED
         IF ((DABS(DUM(1)) .LT. TOL) .AND. (DABS(DUM(2)) .LT. TOL)
     1   .AND. (DABS(DUM(3)) .LT. TOL)) GOTO 40
         GOTO 30
      ENDIF
40    CONTINUE
      RETURN
      END
```

```
C*******************************************************************
C       THE OUTPUT ROUTINE IS NOT LISTED HERE. THE ROUTINE
C       COLLECTS THE DATA AND PUTS IT IN A FILE CALLED CADRES.DAT
C*******************************************************************
C       BELOW ARE THE FUNCTIONS USED THROUGH THE PROGRAM TO
C       EVALUATE THE "A" AND "B" COEFFICIENTS. THE Zi's ARE DUMMY
C       VARIABLES USED TO HOLD THE PRESENT RELEVANT STATE
C       VARIABLE VALUE THAT THE FUNCTION NEEDS
        DOUBLE PRECISION FUNCTION A1(B,V1,KA,PS,Z1)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DOUBLE PRECISION KA
C       THIS FUNCTION HAS A LAMINAR REGION, SO THAT WHEN P1 AND
C       PS (Z IS P1) ARE VERY CLOSE, THE COEFFICIENT DOES NOT TEND
C       TOWARDS INFINITY
        IF ( DABS(PS-Z1) .LE. 1.D0) THEN
          A1 = -B*KA/(V1*(1.D0))
          GOTO 98
        ENDIF
        A1 = -B*KA/(V1*DSQRT(DABS(PS-Z1)))
98      CONTINUE
        RETURN
        END
C*******************************************************************
        DOUBLE PRECISION FUNCTION A2(B,V2,KB,Z2)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DOUBLE PRECISION KB
C       THIS FUNCTION ALSO HAS A LAMINAR REGION, SO THAT WHEN
C       P2 ( Z2 IS P2) IS VERY CLOSE TO ZERO, THE COEFFICIENT DOES
C       NOT TEND TOWARDS INFINITY
        IF ( DABS(Z2) .LE. 1.D0) THEN
          A2 = -B*KB/(V2*(1.D0))
          GOTO 99
        ENDIF
        A2 = -B*KB/(V2*DSQRT(DABS(Z2)))
99      CONTINUE
        RETURN
        END
```

```
C******************************************************************
      DOUBLE PRECISION FUNCTION A3(EF,M)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION M
      A3 = -EF/M
      RETURN
      END
C******************************************************************
      DOUBLE PRECISION FUNCTION B1(AR1,V1,B,PS,Y3,KA,Z3)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION KA
C     THIS FUNCTION ALSO HAS A LAMINAR REGION, SO THAT WHEN
C     PS IS VERY CLOSE TO P1, (Z3 IS P1), THE COEFFICIENT DOES
C     NOT TEND TOWARDS INFINITY
      IF ( DABS(PS-Z3) .LE. 1.D0) THEN
        B1 = B/V1*(KA*PS/(1.D0) - AR1*Y3)
        GOTO 100
      ENDIF
      B1 = B/V1*(KA*PS/DSQRT(DABS(PS-Z3)) - AR1*Y3)
100   CONTINUE
      RETURN
      END
C******************************************************************
      DOUBLE PRECISION FUNCTION B2(B,AR2,V2,Y4)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      B2 = B*AR2*Y4/V2
      RETURN
      END
C******************************************************************
      DOUBLE PRECISION FUNCTION B3(AR1,AR2,M,Z4,Z5)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION M
      B3 = (AR1*Z4 - AR2*Z5)/M
      RETURN
      END
```

```
C*******************************************************************
C       THE NEXT FUNCTION IS THE SYSTEM TIMING FUNCTION THAT
C       HAS BEEN USED THROUGHOUT THE WORK TO TIME THE C.P.U.
C       USED BY DIFFERENT PROGRAMS, ROUTINES AND METHODS
        INTEGER FUNCTION TIM(ATIME)
        IMPLICIT INTEGER*4(A-Z)
        INCLUDE '($JPIDEF)'
        INTEGER*4 ITEMLST(4)
        INTEGER*2 ITEMWORD(8)
        EQUIVALENCE (ITEMLST, ITEMWORD)
        ITEMWORD(1) = 4
        ITEMWORD(2) = JPI$_CPUTIM
        ITEMLST(2) = %LOC(BUFFER)
        ITEMLST(3) = %LOC(BUF_LEN)
        STATUS = SYS$GETJPI(,,,ITEMLST,,,)
        TIM = BUFFER
        RETURN
        END
C*******************************************************************
C       THIS ROUTINE CONVERTS THE TIME INTO DAYS, HOURS, MINS.
C       AND SECS
        SUBROUTINE CONVER(TTIME)
        IMPLICIT INTEGER*4(A-Z)
        INTEGER*4 TEMP(2), DELTA_TIME(2), ZERO(2)
        CHARACTER ASCII_TIME*23
        T = 100000
        STATUS = LIB$EMUL(TTIME, T, 0, TEMP)
        ZERO(1) = 0
        ZERO(2) = 0
        STATUS = LIB$SUBX(ZERO, TEMP, DELTA_TIME, 2)
        STATUS = SYS$ASCTIM(, ASCII_TIME, DELTA_TIME, )
        TYPE*, ' The elapsed C.P.U. time is ', ASCII_TIME
        RETURN
        END
C*******************************************************************
```

# APPENDIX B

# DETAILS OF THE TIME STEP CONTROL

## Introduction

**B.1** The object of this appendix is to explain the time step control employed by the new method, and present the commented coding for this step control. The control of the time step is based on a monitor of the local error, which is discussed in detail in chapter 4. A time step control is essential for a numerical method if the method is to be used with a package such as HASP. Firstly, it takes away the problem for the user, of choosing which steplength to use for each integration, and secondly, a step control mechanism allows for a much higher level of computational efficiency, since the optimal time step can be used throughout the integration.

## Time Step Control

**B.2** Having found the optimal h for the current step by evaluating the norm of the local error, the step control mechanism is then employed. First the current time step is compared with the optimal time step, and if the current steplength is much less than the optimal value of h, then the time step is increased for the next step. It is not increased immediately since this will require re-evaluations that are not necessary, since the results will not be unstable, or inaccurate. How much less than the optimal h the time step is before it is increased has been determined by experiment, the constant factor being chosen so as to be suitable for most problems. If the time step is not increased, then it is again compared with the optimal value of h, this time to see if it is too large. If the time step is determined as being too large, then it is reduced straight away, and the integration step is repeated. This is necessary, since the results that would result from a time step that is too large could affect the following numerical solution. How large the time step can be before it is deemed too large is again determined by a pre-set constant. This constant, which is also determined by experiment, and the previous constant, which determines when the time step is too small, must be chosen to ensure that during an integration, the number of times that the time step is first increased, and then decreased, or vice versa, is kept to a minimum. A commented version of the coding which implements the time step control is now given; it includes the code that evaluates the optimal value of the time step for each step.

## Algorithm For Time Step Control

**B.3** The strategy for finding the optimal value of the time step is explained in detail in chapter 4. The algorithm for the step control is as follows:

i) If the optimal value of h has not already been found for the current step, then the local error for each variable is found
ii) The optimal value of the time step for the current step is found
iii) The current time step is compared with the optimal value of the time step
iv) The current time step is increased if it is less than C1*HOPTIM, where C1 is a pre-determined constant, and HOPTIM is the calculated optimal value of h
v) The current time step is decreased, and the step re-run, if it is greater than C2*HOPTIM, where C2 is another pre-determined constant
vi) If neither iv) or v) apply, then the current step is unchanged, and the program continues

```
C       THE LOCAL ERROR IS NOW EVALUATED, AND THE OPTIMAL TIME
C       STEP FOR THIS STEP IS FOUND. THEN, THE CURRENT TIME STEP
C       THAT HAS BEEN USED IS COMPARED WITH THE OPTIMAL TIME STEP
C       AND THE NECESSARY CHANGES TO THE STEPLENGTH ARE MADE. IF
C       THE TIME STEP IS TOO SMALL, THEN IT IS INCREASED ON THE
C       NEXT STEP, BUT IF IT IS TOO LARGE, IT IS DECREASED STRAIGHT
C       AWAY, AND THE STEP IS RE-EVALUATED. THE CONSTANTS CHOSEN
```

```
C       WERE THE BEST ONES FOUND BY EXPERIMENT, THAT DID NOT
C       LEAD TO CONSTANT INCREASING AND DECREASING OF THE TIME
C       STEP. MORE EXPERIMENTAL VERIFICATION MAY STILL
C       BE NECESSARY
C*********************************************************************
C       VARIABLES USED
C       FAIL    -   INTEGER VARIABLE USED TO DETERMINE IF OPTIMAL
C                   TIME STEP HAS ALREADY BEEN EVALUATED FOR
C                   PRESENT STEP. IT IS SET TO 1 IF THE OPTIMAL H HAS
C                   ALREADY BEEN EVALUATED
C       Y1      -   ARRAY THAT HOLDS THE CORRECTED VALUES OF THE
C                   STATE VARIABLES
C       STORE   -   ARRAY THAT HOLDS THE PREDICTED VALUES OF THE
C                   STATE VARIABLES
C       N       -   NUMBER OF STATE VARIABLES
C       ERROR   -   DOUBLE PRECISION ARRAY USED TO STORE THE
C                   VALUES FOR THE LOCAL ERROR
C       ERRMAX-     DOUBLE PRECISION VARIABLE THAT HOLDS THE NORM
C                   OF THE LOCAL ERROR
C       HOPTIM  -   DOUBLE PRECISION VARIABLE THAT HOLDS THE
C                   OPTIMAL VALUE FOR H ON THE PRESENT STEP
C       H       -   PRESENT TIME STEP EMPLOYED BY THE METHOD
C       HNEXT   -   EVALUATED STEPLENGTH FOR THE NEXT STEP
C       TOL     -   PRE-SET TOLERANCE; ALSO USED BY THE ITERATION
C                   SCHEME
C       INCREM1,INCREM2 -  INTEGER VARIABLES USED BY THE ITERATION
C       SCHEME TO ENSURE ITERATES ARE CONVERGING
C*********************************************************************
C       CHECKING THAT OPTIMAL H HAS NOT ALREADY BEEN FOUND FOR
C       THIS STEP
        IF (FAIL .EQ. 0) THEN
          IF ((INCREM1 .EQ. 0) .AND. (INCREM2 .EQ. 0))THEN
C       EVALUATING THE LOCAL ERROR FOR EACH VARIABLE
            DO 15 I = 1,N
              ERROR(I) = DABS(Y1(I) - STORE(I))/2.D0
15          CONTINUE
            ERRMAX = 0
C       EVALUATING THE NORM OF THE LOCAL ERROR
            DO 20 I = 1,N
              ERRMAX = DMAX1(ERRMAX,ERROR(I))
```

```
20          CONTINUE
C       EVALUATING THE OPTIMAL VALUE OF THE TIME STEP FOR
C       THE PRESENT STEP
            HOPTIM = H*DSQRT(TOL/ERRMAX)
C       COMPARING THE PRESENT TIME STEP WITH OPTIMAL TIME STEP
            IF (H .LT. (2.5D-1*HOPTIM)) THEN
C       INCREASING STEPLENGTH FOR NEXT STEP
              HNEXT = HOPTIM*2.D0/3.D0
              FAIL = 1
            ENDIF
C       COMPARING THE PRESENT TIME STEP WITH OPTIMAL TIME STEP
            IF (H .GT. (8.4D-1*HOPTIM)) THEN
C       DECREASING STEPLENGTH FOR PRESENT STEP
              HNEXT = HOPTIM/2.D0
              FAIL = 1
              GOTO 2
            ENDIF
          ENDIF
        ENDIF
C**********************************************************************
```

# APPENDIX C

# EXAMPLES OF CHANGED COMPUTER CODING FOR THE IMPLEMENTATION OF THE NEW METHOD IN HASP

## Introduction

**C.1** The purpose of this appendix is twofold: one is to present a commented version of the general coding implementing the new method, and the other is to present the changed code within HASP that allows the use of the new method, rather than Gear's method, for the first example given in chapter 5. The code for the new method includes a perturbation technique for the formulation of the Jacobian matrix, and the particular code given here employs a Newton-Jacobi iteration scheme, where only the diagonal elements of the Jacobian matrix are used in the iteration scheme. This process was explained in detail in chapter 2. The examples of the changed code for the implementation of the method are three of the routines introduced in chapter 5 that are written individually by the program generator for each simulation. The routines given are MAIN, AUX and CAD.OPT. No changes had to be made to CONTRL or OUT, so they are not presented here. An algorithm for the new method is given, but for a detailed explanation of the coding for the routines written by HASP, the reader is referred to [3] and [4].

**C.2** The first code given is the general purpose version of the new method. This consists of two routines; one which sets up the necessary parameters and finds the initial coefficient values, and the other which employs the new method to give the numerical solution to the problem at the next time step. The algorithm describing the action taken by the program is as follows:

i) The variables are set up, e.g. maximum time step, starting time step

ii) The coefficient values are found, in this case from the relevant models

iii) The routine implementing the method is called

iv) The local variables for the routine implementing the method are set up, initialised, etc.

v) The predicted values for the solution are made using the explicit method

vi) The Jacobian matrix is formed by a perturbation technique, explained in chapter 5, and checks are carried out to ensure there is no division by zero. This will require further coefficient evaluations

vii) The implicit method is applied, with Newton iterations being used to find the solution

viii) The local error is evaluated if necessary, and the optimal time step calculated

ix) The time step control mechanism is used, and either the step is re-run, or an error test is carried out on the iterates to see if they have converged

x) If necessary the iterations continue, either with the same Jacobian or a newly evaluated Jacobian

xi) If the iterates have not converged sufficiently quickly, then the time step is reduced and the step re-started

xii) When the iterates have converged, then a final check is made to the subroutine AUX to ensure that no physical conditions have been violated

xiii) The results at the present step are stored, and the step is incremented by the current value of the time step

xiv) If required, the results are output (linear interpolation may be necessary)

xv) If the value of time is equal to the end time then the subroutine will finish, if not, then the program returns to ii)

```
C*********************************************************************
C     SUBROUTINE FOM - GENERAL PURPOSE SOLVER
C*********************************************************************
C     MAIN VARIABLES USED
C     T     -     INDEPENDENT VARIABLE
C     HT    -     TIME STEP
C     TEND  -     FINAL INTEGRATION TIME
C     Y     -     ARRAY HOLDING STATE VARIABLE VALUES
C     N     -     NUMBER OF STATE VARIABLES
C     TOL   -     TOLERANCE USED BY THE ITERATION SCHEME, AND
C                 BY THE TIME STEP CONTROL MECHANISM
```

```
C      ITEST -      HASP VARIABLE (UNUSED BY FOM)
C      TAB   -      USED-DEFINED PRINT INTERVAL
C      IFAIL -      INTEGER WARNING OF ERROR. UNUSED HERE
C      YY    -      ARRAY USED TO HOLD PRETURBED STATE VARIABLES
C      A,B   -      ARRAYS HOLDING COEFFICIENT VALUES
C      A1,A2 -      ARRAYS USED FOR HOLDING COEFFICIENT VALUES
C                   FOR THE EVALUATION OF THE JACOBIAN
C      BBB   -      ALSO USED WHEN EVALUATING THE JACOBIAN
C      YINT,YOLD    USED IN THE LINEAR INTERPOLATION PROCESS
C      HSTART       STARTING TIME STEP FOR INTEGRATION METHOD
C      HMAX -       MAXIMUM TIME STEP ALLOWED
C      LIMIT -      INTEGER VARIABLE USED TO RELAY INFORMATION
C                   FROM THE MODELS
C*****************************************************************
       SUBROUTINE FOM(T,TEND,Y,N,AUX,TOL,ITEST,TAB,OUT,IFAIL)
       IMPLICIT DOUBLE PRECISION(A-H,O-Z)
       DIMENSION  YY(20),A(20),B(20),UNIT(20),Y(N),A1(20),B1(20),A2(20),B2(20)
       DIMENSION AAA(20),BBB(20),YOLD(20),YINT(20)
       COMMON /TSTEP/HT
       INTEGER LIMIT
       LIMIT = 0
       HSTART = 1.D-6
       HMAX = 1.D-2
C      OUTPUTTING INITIAL CONDITIONS
       CALL OUT(T,Y,N,TAB,0)
C      SETTING THE PETURBATION FACTOR FOR THE JACOBIAN
       EPS = 1.D-2
       HT = HSTART
       TTAB = T + TAB
5      CONTINUE
       LIMIT = 1
C      FINDING COEFFICIENT VALUES FROM THE MODELS
       CALL AUX(AAA,Y,T,N,LIMIT)
       DO 10 I = 1,N
          A(I) = AAA(I)
          B(I) = AAA(N+I)
10     CONTINUE
C      CALLING THE ROUTINE IMPLEMENTING THE METHOD
       CALL FSTORM(Y,YOLD,YY,A,A1,A2,B,B1,B2,BBB,T,N,EPS,HMAX,
     1                   TOL,UNIT,LIMIT)
```

```
C       CHECKING TO SEE IF LINEAR INTERPOLATION IS REQUIRED
        IF (T .EQ. TTAB) THEN
C       OUTPUTTING RESULTS
          CALL OUT(T,Y,N,TAB,1)
          TTAB = TTAB + TAB
        ENDIF
        IF (T .GT. TTAB) THEN
          D0 15 I = 1,N
             YINT(I) = YOLD(I)*(1.D0 - (TTAB - (T-HT))/HT) +
     1                    Y(I)*(TTAB - (T-HT))/HT
15        CONTINUE
C       OUTPUTTING RESULTS
          CALL OUT(TTAB,YINT,N,TAB,1)
          TTAB = TTAB + TAB
        ENDIF
        IF (T .GE. TEND) GOTO 20
        GOTO 5
20      CONTINUE
        RETURN
        END
C*******************************************************************
C       A GENERAL PURPOSE SUBROUTINE TO SOLVE O.D.E.'S USING THE
C       NEW FIRST ORDER METHOD
C*******************************************************************
        SUBROUTINE FSTORM(Y,YOLD,YY,A,A1,A2,B,B1,B2,BBB,T,N,EPS,
     1                          HMAX,TOL,UNIT,LIMIT)
        IMPLICIT DOUBLE PRECISION(A-H,O-Z)
        DIMENSION Y(N),A(N),A1(N),A2(N),B(N),B1(N),B2(N),STORE(20),YY(N),
     1              DUM(20), R(20), RATIO(20), UNIT(N), DERIV(20), BBB(2*N),
     1             VALA(20), VALB(20), ERROR(20), Y1(20), CCC(20), YOLD(N)
        COMMON /TSTEP/HT
        DOUBLE PRECISION MAXERR
        INTEGER INCREM1, INCREM2, FAIL, N, LIMIT
C       INITIALISING COUNTERS FOR THE ITERATION SCHEME
        HNEXT = HT
        FAIL = 0
        INCREM1 = 0
        INCREM2 = 0
25      CONTINUE
        HT = HNEXT
```

```
C     EVALUATING THE PREDICTED RESULTS AT EACH TIME STEP
      DO 30 I = 1,N
C     CHECKING TO SEE IF THE A COEFFICIENTS ARE ZERO
        IF ((A(I) .EQ. 0.D0) .AND. (B(I) .EQ. 0.D0)) THEN
          Y1(I) = Y(I)
        ELSEIF ((A(I) .EQ. 0.D0) .AND. (B(I) .NE. 0.D0)) THEN
          Y1(I) = B(I)*HT + Y(I)
        ELSE
          Y1(I) = (B(I)/A(I))*(DEXP(A(I)*HT) - 1.D0) + Y(I)*DEXP(A(I)*HT)
        ENDIF
        STORE(I) = Y1(I)
30    CONTINUE
40    CONTINUE
C     PERTURBING THE STATE VARIABLE VALUES AND MODIFYING THE
C     COEFFICIENT VALUES READY TO FORM THE JACOBIAN MATRIX
      D0 45 I = 1,N
        YY(I) = Y1(I)
45    CONTINUE
      CALL AUX(CCC, YY, T, N, LIMIT)
      DO 50 I = 1,N
        A2(I) = CCC(I)
        B2(I) = CCC(N+I)
50    CONTINUE
      DO 55 I = 1,N
        YY(I) = Y1(I) + EPS*Y1(I)
        CALL AUX(BBB, YY, T, N, LIMIT)
        YY(I) = Y1(I)
        A1(I) = BBB(I)
        B1(I) = BBB(N+I)
55    CONTINUE
60    CONTINUE
      CALL AUX(CCC, Y1, T, N, LIMIT)
      DO 65 I = 1,N
        VALA(I) = CCC(I)
        VALB(I) = CCC(N+I)
65    CONTINUE
C     THIS DO-LOOP EVALUATES THE ELEMENTS OF THE JACOBIAN USED
C     BY THE ITERATION SCHEME, USING A PERTURBATION TECNIQUE.
C     THE COEFFICIENTS CORRESPONDING TO THE PERTURBED STATE
C     VARIABLES HAVE ALREADY BEEN EVALUATED
```

```
      IF ( INCREM1 .EQ. 0 ) THEN
        D0 70 I = 1,N
          IF (A2(I) .EQ. 0.D0) THEN
            DIVIS1 = B2(I)*HT
          ELSE
            DIVIS1 = B2(I)/A2(I)
          ENDIF
          IF (A1(I) .EQ. 0.D0) THEN
            DIVIS2 = B1(I)*HT
          ELSE
           DIVIS2 = B1(I)/A1(I)
          ENDIF
          IF (Y(I) .LT. 1.D-4) THEN
            DERIV(I) = (DIVIS2 - DIVIS1)/(1.D-4)
          ELSE
            DERIV(I) = (DIVIS2 - DIVIS1)/(Y1(I)*EPS)
          ENDIF
          IF (( A2(I) .EQ 0.D0 ) .AND. ( A1(I) .EQ. 0.D0)) THEN
            UNIT(I) = 1.D0/(1.D0 - ((B1(I) - B2(I))*HT)/(Y1(I)*EPS))
          ELSE
            UNIT(I) = 1.D0/(1.D0 - (DEXP(VALA(I)*HT) - 1.D0)*DERIV(I))
          ENDIF
70    CONTINUE
C     PERFORMING NEWTON ITERATIONS, HOLDING "A"
C     COEFFICIENTS CONSTANT
      DO 75 I = 1,N
        IF (VALA(I) .EQ. 0.D0) THEN
          DUM(I) = Y1(I) - Y(I) - VALB(I)*HT
        ELSE
          DUM(I) = Y1(I) - DEXP(VALA(I)*HT)*Y(I) -
     1              (DEXP(VALA(I)*HT) - 1.D0)*VALB(I)/VALA(I)
        ENDIF
        DUM(I) = DUM*UNIT(I)
        R(I) = Y1(I)
        Y1(I) = Y1(I) - DUM(I)
75    CONTINUE
C     EVALUATING THE LOCAL ERROR AND THE OPTIMAL TIME STEP,
C     AND COMPARING PRESENT TIME STEP WITH OPTIMAL TIME STEP
```

```
      IF (FAIL .EQ. 0 ) THEN
        DO 80 I = 1,N
          ERROR(I) = DABS(Y1(I) - STORE(I))/2.D0
80      CONTINUE
        ERRMAX = 0.D0
        DO 85 I = 1,N
          ERRMAX = DMAX1(ERRMAX,ERROR(I))
85      CONTINUE
        IF (ERRMAX .LE. 1.D-15) GOTO 90
        HOPTIM = HT*DSQRT(TOL/ERRMAX)
        IF (HT .LT. 2.5D-1*HOPTIM) THEN
          HNEXT  = HOPTIM*(2.D0/3.D0)
          IF ( HNEXT .GT. HMAX) THEN
            HNEXT = HMAX
          ENDIF
          FAIL = 1
        ENDIF
        IF (HT .GT. 8.4D-1*HOPTIM) THEN
          HNEXT = HOPTIM
          FAIL = 1
          GOTO 25
        ENDIF
      ENDIF
90    CONTINUE
C     PERFORMING A MIXED ERROR TEST ON THE ITERATES TO SEE IF
C     THEY HAVE CONVERGED YET
      DO 95 I = 1,N
        RATIO(I) = DABS((Y1(I) - R(I))/(1.D0 + DABS(R(I))))
95    CONTINUE
      MAXERR = 0.D0
      DO 100 I = 1,N
        MAXERR = DMAX1(MAXERR,RATIO(I))
100   CONTINUE
      IF (MAXERR .LT. TOL) GOTO 105
C     CHECKING TO SEE HOW MANY ITERATIONS HAVE TAKEN PLACE. IF
C     MORE THAN  5, THEN THE JACOBIAN IS RE-EVALUATED. IF THE
C     JACOBIAN HAS ALREADY BEEN RE-EVALUATED, THEN THE TIME
C     STEP IS DECREASED AND THE STEP IS RE-RUN
      INCREM1 = INCREM1 + 1
      IF (INCREM1 .LE. 5) THEN
        GOTO 60
        ELSE
          IF (INCREM2 .EQ. 0) THEN
            INCREM2 = 1
            INCREM1 = 0
            GOTO 40
          ELSE
            HNEXT = HT/4.D0
            GOTO 25
          ENDIF
        ENDIF
105     CONTINUE
C       CALLING AUX TO CHECK NO PHYSICAL VIOLATIONS HAVE
C       OCCURRED
        LIMIT = 2
        CALL AUX(CCC,Y1,T,N,LIMIT)
        IF (( LIMIT .EQ. 3) .OR. ( LIMIT .EQ. 5)) THEN
          HNEXT = HT/4.D0
          GOTO 25
        ENDIF
        DO 110 I = 1,N
          YOLD(I) = Y(I)
          Y(I) = Y1(I)
110     CONTINUE
        T = T + HT
        RETURN
        END
C*********************************************************************
```

**C.3** The next pieces of code are generated by HASP for use with Gear's method. The circuit for which this code has been written is shown in Figure 5.2. The code is given in the same form as it is written by HASP.

```
C
C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C %%%%%SUBROUTINE AUX - CALLS UP MODEL CALCULATION ROUTINES
C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C
      SUBROUTINE AUX(DOT, Y, T, N, LIMIT)
      IMPLICIT DOUBLE PRECISION(A-H, O-Y)
      DIMENSION DOT( 5), Y( 5)
      COMMON EFF(13), FLO(13), IWRITE, IPOS, NOL, IPTS, SIG( 2)
     + , CON1(13), ICON1(1), CON2(1), CON3(12), ICON3(3), CON4(3), CON5(3)
     + , CON6(10), CON7(12), ICON7(3), CON8(29), ICON8(4), CON9(12)
     + , ICON9(3), CON10(31), ICON10(3)
      COMMON /TSTEP/HT
C
C %%%ASSIGN THE STATE VARIABLES CALCULATED BY THE INTEGRATOR
C %%%TO THE APPROPRIATE LINKS
      EFF( 2)=Y( 1)
      EFF( 9)=Y( 1)
      EFF(10)=Y( 1)
      EFF( 6)=Y( 2)
      EFF( 3)=Y( 2)
      EFF(13)=Y( 3)
      FLO(13)=Y( 4)
      EFF( 7)=Y( 5)
      EFF( 4)=Y( 5)
C
C %%%CALL EACH MODEL CALCULATION SUBROUTINE IN THE ORDER
C %%%DEFINED BY THE CALLING SEQUENCE DETERMINED IN THE
C %%%PROGRAM GENERATOR SEGMENT PGCMP
C %%%IF LIMIT IS INCLUDED IN THE ARGUMENT LIST OF A MODEL
C %%%AND IT IS SET TO 3 OR 5, THEN RETURN TO THE INTEGRATOR
C %%%AND INTERVAL HALVE
      CALL PM00(FLO(12), CON2)
      CALL TKO3(EFF(8), EFF(11), EFF(5), CON5)
      CALL AL00(EFF(6), FLO(6), EFF(7), FLO(7), EFF(13), FLO(13),
     +    SIG(1), SIG(2), LIMIT, CON8, ICON8, DOT(3), DOT(4))
```

```
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      CALL DE01(EFF(1), T, LIMIT, CON10, ICON10)
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      CALL PU00(EFF(8), EFF(9), FLO(12), FLO(8), FLO(9), EFF(12),
     +   CON1, ICON1)
      CALL PC01(EFF(10), EFF(11), FLO(10), FLO(11), CON4)
      CALL DC1T(EFF(5), EFF(1), EFF(2), EFF(3), EFF(4), FLO(5),
     +   FLO(1), FLO(2), FLO(3), FLO(4), CON6)
      CALL PI06(EFF(6), FLO(6), FLO(3), 0.D0, 0.D0, 0.D0, 0.D0, 0.D0,
     +   SIG(1), LIMIT, CON7, ICON7, DOT(2))
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      CALL PI06(EFF(7), FLO(7), FLO(4), 0.D0, 0.D0, 0.D0, 0.D0, 0.D0,
     +   SIG(2), LIMIT, CON9, ICON9, DOT(5))
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      CALL PI05(EFF(2), FLO(2), FLO(9), 0.D0, 0.D0, 0.D0,
     +   0.D0, 0.D0, LIMIT, CON3, ICON3, DOT(1))
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      END

C
C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C %%%SIMULATION PROGRAM - MAIN SEGMENT
C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C
      IMPLICIT DOUBLE PRECISION(A-H, O-Y)
      DIMENSION DOT( 5,4), Y( 5), DFDY( 5, 5), DIF( 5,7,2)
      COMMON EFF(13), FLO(13), IWRITE, IPOS, NOL, IPTS, SIG( 2)
     +  , CON1(13), ICON1(1), CON2(1), CON3(12), ICON3(3), CON4(3), CON5(3)
     +  , CON6(10), CON7(12), ICON7(3), CON8(29), ICON8(4), CON9(12)
     +  , ICON9(3), CON10(31), ICON10(3)
      COMMON /TSTEP/HT
      EXTERNAL AUX, OUT
      PARAMETER (N= 5)
      DATA ITEST, IFAIL, T/1, 0, 0.D0/
C
C %%%SET INDICATOR IPASS TO 1 (REQUIRED IN CONTRL)
C %%%ALSO SET NUMBER OF DATA POINTS ALREADY CALCULATED TO 0
      IPASS=1
      IPTS=0
C
```

```
C %%%CALL CONTRL - USER INPUT SUBROUTINE
  1000 CALL CONTRL(Y, TOL, N, T, TEND, TAB, IPASS, LAST)
C
C %%%INTEGRATOR CALLS SUBROUTINES AUX AND OUT
        CALL GEARKC(T, TEND, Y, N, AUX,
     +  TOL, ITEST, TAB, OUT, DOT, DFDY, DIF, IFAIL)
        NPTS=(IPOS-2)/(2*NOL+1)+1
        LAST=IPOS
C
C %%%WRITE NUMBER OF POINTS AND NUMBER OF LINKS
C %%%INTO THE FIRST RECORD OF CADRES.DAT
        WRITE(11, REC=1)NPTS, NOL
        IPASS=2
C
C %%%SIMULATION COMPLETED - RETURN TO CONTRL
        GOTO 1000
        END


! %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! %%%CAD.OPT - COMPONENT MODEL SELECTOR FILE
! %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! ******************************************************************
! *****GENERATED SEGMENTS ******************************************
! ******************************************************************
!
        MAIN, AUX, OUT, CONTRL
!
! ******************************************************************
! *****STANDARD UTILITIES AND INTEGRATOR ***************************
! ******************************************************************
!
[HASP.COMPON]MESAGE, PPROP, FPROP, GPROP, CUBIC,-
[CAPLEN.INHASP.INHASP1.INHASP2]GEAR4, GEAR5, GEAR6,-
!
! ******************************************************************
! *****STANDARD INPUT ROUTINES *************************************
! ******************************************************************
!
[HASP.COMPON]PU00IN, PM00IN, PI05IN, PC01IN, TK03IN,-
[HASP.COMPON]DC1TIN, PI06IN, AL00IN, DE01IN,-
```

```
!
! ******************************************************************
! *****STANDARD CALCULATION ROUTINES **************************
! ******************************************************************
!
[HASP.COMPON]PU00, PM00, PI05, PC01, TK03,-
[HASP.COMPON]DC1T, PI06, AL00, DE01
!
! ******************************************************************
! *****END OF OPTIONS FILE  ****************************************
! ******************************************************************
```

**C.4** The routines MAIN, AUX and CAD.OPT that have been written by HASP for use with Gear's method have been changed to allow the successful implementation of the new method. These changes are a result of modifications made to the program generator and they allow the new method to be used with the test circuit given in Figure 5.2. The coding that is written by the HASP package when the new method is used is now given.

```
C
C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C %%%%%SUBROUTINE AUX - CALLS UP MODEL CALCULATION ROUTINES
C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C
      SUBROUTINE AUX(DOT, Y, T, N, LIMIT)
      IMPLICIT DOUBLE PRECISION(A-H, O-Y)
      DIMENSION DOT( 10), Y( 5), COEFF( 5)
      COMMON EFF(13), FLO(13), IWRITE, IPOS, NOL, IPTS, SIG( 2)
     + , CON1(13), ICON1(1), CON2(1), CON3(12), ICON3(3), CON4(3), CON5(3)
     + , CON6(10), CON7(12), ICON7(3), CON8(29), ICON8(4), CON9(12)
     + , ICON9(3), CON10(31), ICON10(3)
      COMMON /TSTEP/HT
C
C %%%ASSIGN THE STATE VARIABLES CALCULATED BY THE INTEGRATOR
C %%%TO THE APPROPRIATE LINKS
```

```
      EFF( 2)=Y( 1)
      EFF( 9)=Y( 1)
      EFF(10)=Y( 1)
      EFF( 6)=Y( 2)
      EFF( 3)=Y( 2)
      EFF(13)=Y( 3)
      FLO(13)=Y( 4)
      EFF( 7)=Y( 5)
      EFF( 4)=Y( 5)
C
C %%%CALL EACH MODEL CALCULATION SUBROUTINE IN THE ORDER
C %%%DEFINED BY THE CALLING SEQUENCE DETERMINED IN THE
C %%%PROGRAM GENERATOR SEGMENT PGCMP
C %%%IF LIMIT IS INCLUDED IN THE ARGUMENT LIST OF A MODEL
C %%%AND IT IS SET TO 3 OR 5, THEN RETURN TO THE INTEGRATOR
C %%%AND INTERVAL HALVE
      CALL PM00(FLO(12), CON2)
      CALL TKO3(EFF(8), EFF(11), EFF(5), CON5)
      CALL AL00(EFF(6), FLO(6), EFF(7), FLO(7), EFF(13), FLO(13),
     +   SIG(1), SIG(2), LIMIT, CON8, ICON8, DOT)
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      CALL DE01(EFF(1), T, LIMIT, CON10, ICON10)
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      CALL PU00(EFF(8), EFF(9), FLO(12), FLO(8), FLO(9), EFF(12),
     +   CON1, ICON1, COEFF(1))
      CALL PC01(EFF(10), EFF(11), FLO(10), FLO(11), CON4, COEFF(2))
      CALL DC1T(EFF(5), EFF(1), EFF(2), EFF(3), EFF(4), FLO(5),
     +   FLO(1), FLO(2), FLO(3), FLO(4), CON6, COEFF)
      CALL PI06(EFF(6), FLO(6), FLO(3), 0.D0, 0.D0, 0.D0, 0.D0, 0.D0,
     +   SIG(1), LIMIT, CON7, ICON7, DOT(2), DOT(7), COEFF(4))
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      CALL PI06(EFF(7), FLO(7), FLO(4), 0.D0, 0.D0, 0.D0, 0.D0, 0.D0,
     +   SIG(2), LIMIT, CON9, ICON9, DOT(5), DOT(10), COEFF(5))
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      CALL PI05(EFF(2), FLO(2), FLO(9), 0.D0, 0.D0, 0.D0,
     +   0.D0, 0.D0, LIMIT, CON3, ICON3, DOT(1), DOT(6), COEFF)
      IF(LIMIT .EQ. 3 .OR. LIMIT .EQ. 5) RETURN
      END
```

```
C
C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C %%%SIMULATION PROGRAM - MAIN SEGMENT
C %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C
      IMPLICIT DOUBLE PRECISION(A-H, O-Y)
      DIMENSION Y( 5)
      COMMON EFF(13), FLO(13), IWRITE, IPOS, NOL, IPTS, SIG( 2)
   +  , CON1(13), ICON1(1), CON2(1), CON3(12), ICON3(3), CON4(3), CON5(3)
   +  , CON6(10), CON7(12), ICON7(3), CON8(29), ICON8(4), CON9(12)
   +  , ICON9(3), CON10(31), ICON10(3)
      COMMON /TSTEP/HT
      EXTERNAL AUX, OUT
      PARAMETER (N= 5)
      DATA ITEST, IFAIL, T/1, 0, 0.D0/
C
C %%%SET INDICATOR IPASS TO 1 (REQUIRED IN CONTRL)
C %%%ALSO SET NUMBER OF DATA POINTS ALREADY CALCULATED TO 0
      IPASS=1
      IPTS=0
C
C %%%CALL CONTRL - USER INPUT SUBROUTINE
 1000 CALL CONTRL(Y, TOL, N, T, TEND, TAB, IPASS, LAST)
C
C %%%INTEGRATOR CALLS SUBROUTINES AUX AND OUT
      CALL FOM(T, TEND, Y, N, AUX, TOL, ITEST, TAB, OUT, IFAIL)
      NPTS=(IPOS-2)/(2*NOL+1)+1
      LAST=IPOS
C
C %%%WRITE NUMBER OF POINTS AND NUMBER OF LINKS
C %%%INTO THE FIRST RECORD OF CADRES.DAT
      WRITE(11, REC=1)NPTS, NOL
      IPASS=2
C
C %%%SIMULATION COMPLETED - RETURN TO CONTRL
      GOTO 1000
      END
```

```
! %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! %%%CAD.OPT - COMPONENT MODEL SELECTOR FILE
! %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
! ******************************************************************
! *****GENERATED SEGMENTS *************************************
! ******************************************************************
!
        MAIN, AUX, OUT, CONTRL
!
! ******************************************************************
! *****STANDARD UTILITIES AND INTEGRATOR **********************
! ******************************************************************
!
[HASP.COMPON]MESAGE, PPROP, FPROP, GPROP, CUBIC,-
[CAPLEN.INHASP.INHASP1]MC,-
!
! ******************************************************************
! *****STANDARD INPUT ROUTINES ********************************
! ******************************************************************
!
[HASP.COMPON]PU00IN, PM00IN, PI05IN, PC01IN, TK03IN,-
[HASP.COMPON]DC1TIN, PI06IN, AL00IN, DE01IN,-
!
! ******************************************************************
! *****STANDARD CALCULATION ROUTINES ***********************
! ******************************************************************
!
[HASP.COMPON]PM00, TK03, DE01-
[CAPLEN.INHASP.INHASP1]PI05, PI06, AL00, PU00, PC01, DC1T
!
! ******************************************************************
! *****END OF OPTIONS FILE ************************************
! ******************************************************************
```

**C.5** The main problem with the implementation of the new method is that each model must be re-written so that it passes coefficient values back to AUX, rather than a derivative. This is a long and arduous task, and before it can be undertaken, the accuracy of the method must be improved, so as to make it worthwhile.

# APPENDIX D

# USER-WRITTEN CODE AND TEST PROBLEMS FOR THE DETEST PACKAGE

## Introduction

**D.1** The purpose of this appendix is to provide an example of the routines needed to drive the DETEST package, discussed in chapter 6, and to give the full set of stiff test problems that have been collected together by Enright, Hull et al [76]. Also included are a set of discontinuous test problems and a set of chemical kinetics problems. The FORTRAN code given below must be written by the user of the package, and this particular code will produce the results found in Figure 6.7. A more detailed explanation of the DETEST package, and the reasons for choosing the particular test problems, is given by Enright [17], and another set of test examples, which have been collected to test methods developed to solve non-stiff problems, are also included.

**D.2** The user of the package must write the routines METHOD and SOLVER, plus the main driving program. SOLVER contains the code for the method to be tested, and the subroutine METHOD declares the workspace, initialises the parameters required by SOLVER, signals the appropriate options, invokes SOLVER, and then returns to the main body of code. The main program is used to define: which problems are to be solved; the required tolerances with which to solve these problems; the value of OPTION for the statistical information required; whether normalised statistics are needed and which title to print with the output. The code for the SOLVER routine can be found in Appendix C

**Main Program**

**D.3** An example of the main program which must be written is now given

```
C     SAMPLE  DRIVER FOR STDTST, WITH ONE GROUP CONSISTING ONLY
C     OF PROBLEM A1 SOLVED IN UNSCALED FORM, AT THREE
C     TOLERANCES AND WITH OPTION=2 AND NORMEF=0.
C     IN THIS CASE THE ARRAYS IDLIST, TOL NEED NOT BE SO LONG.
      INTEGER TITLE(20), IDLIST(60)
      REAL TOL(11)
      DATA TITLE / ' SECD ',' ER ',' ADDI ',' SON- ',' ENRI ',' GHT ',
     *      ' SECO ', ND D ',' ERIV ',' ATIV ',' E ME ',' THOD ',8*' '/
     *      ,TOL /1.E-2, 1.E-4, 1.E-6, 7*E0 /
     *      ,IDLIST / 11, 0, 58*0 /
      CALL STDTST( TITLE, 2, 0, TOL, IDLIST, FLAG )
      STOP
      END
C
```

**METHOD**

**D.4** An example of the routine, METHOD, is:

```
      SUBROUTINE METHOD (N, X, Y, XEND, TOL, HMAX, HSTART)
C     DRIVER FOR THE SECDER CODE WHICH IS PART OF THE PACKAGE.
C     IT IS SOMEWHAT LENGTHY BECAUSE ITS INTERRUPT MECHANISM
C     DOES NOT ALLOW INTERRUPT IMMEDIATELY AFTER ACCEPTING
C     A STEP
      IMPLICIT REAL*8(A-H, O-Z)
      REAL*8 X, Y(N), XEND, TOL, HMAX, HSTART
      EXTERNAL FCN, PDERV
      REAL*8 C(20), YP(20,11), W(400), P(400), WK(20, 11)
      INTEGER INF(40)
      COMMON / STCOM6 / NFCN, NJAC, NLUD
      DATA NDIM /20/
      IND = 2
      DO 20 I = 1, 5
            INF(I) = 0
            C(I) = 0.D0
20    CONTINUE
C     SET ABS. ERROR CONTROL .. INF (1), INTERRUPT NO. 2 .. INF(5),
C     MIN, MAX & STARTING STEP SIZE .. C(2), C(4), C(5)
      INF(1) = 1
      INF(5) = 1
      C(2) = 1.D-12
      C(4) = HMAX
      C(5) = HSTART
50    CALL TRUE(FCN,PDERV,NDIM,N,X,Y,XEND,TOL,IND,C,INF,YP,W,P,WK)
      IF ( IND .EQ. 6) GOTO 50
      IF ( IND .NE. 5 ) GOTO 60
            TEMP = C(13)
            CALL STATS( C(13), WK(1,1), C(15), TOL)
            IF (C(13) .NE. TEMP ) GOTO 70
      GOTO 50
60    IF ( IND .NE. 3 ) GOTO 70
            X = XEND
      GOTO 80
C     FAILURE EXIT OF SOME KIND
70    X = C(13)
80    CONTINUE
      RETURN
      END
```

**Specification Of Problems**

**D.5** The most valuable part of the DETEST package is the large set of test problems that it provides. These are now given

Problem Class A. Linear with real eigenvalues

A1: $y_1' = -.5y_1$ $\quad y_1(0) = 1$

$y_2' = -y_2$ $\quad y_2(0) = 1$

$y_3' = -100y_3$ $\quad y_3(0) = 1$

$y_4' = -90y_4$ $\quad y_4(0) = 1$

$x_f = 20$ $\quad h_0 = 10^{-2}$

A2: $y_1' = -1800y_1 + 900y_2$ $y_1(0) = 0$

$y_i' = y_{i-1} - 2y_i + y_{i+1}$ $y_i(0) = 0$ ( $i = 2, ..., 8$)

$y_9' = 1000y_8 - 2000y_9 + 1000$ $y_9(0) = 0$

A3: $y_1' = -104y_1 + 100y_2 - 10y_3 + y_4$ $y_1(0) = 1$

$y_2' = -103y_2 + 10y_3 - 10y_4$ $y_2(0) = 1$

$y_3' = -y_3 + 10y_4$ $y_3(0) = 1$

$y_4' = -.1y_4$ $y_4(0) = 1$

$x_f = 20$ $h_0 = 10^{-5}$

A4: $y_i' = -i^5 y_i$ $y_i(0) = 1$ ( $i = 1,2, ..., 10$ )

$x_f = 1$ $h_0 = 10^{-5}$

Problem class B. Linear with non-real eigenvalues

B1: $y_1' = -y_1 + y_2$ $y_1(0) = 1$

$y_2' = -100y_1 - y_2$ $y_2(0) = 0$

$y_3' = -100y_3 + y_4$ $y_3(0) = 1$

$y_4' = -10000y_3 - 100y_4$ $y_4(0) = 0$

$x_f = 20$ $h_0 = 7 \times 10^{-3}$

B2: $y_1' = -10y_1 + \alpha y_2$ $y_1(0) = 1$

$y_2' = -\alpha y_1 - 10y_2$ $y_2(0) = 1$

$y_3' = -4y_3$ $y_3(0) = 1$

$y_4' = -y_4$ $y_4(0) = 1$

$y_5' = -.5y_5$ $y_5(0) = 1$

$y_6' = -.1y_6$ $y_6(0) = 1$

$x_f = 20 \qquad h_0 = 10^{-2} \qquad \alpha = 3$

B3: As in B2 with $\alpha = 8$

B4: As in B2 with $\alpha = 25$

B5: As in B2 with $\alpha = 100$

Problem Class C. Non-linear coupling

C1: $y_1' = -y_1 + (y_2)^2 + (y_3)^2 + (y_4)^2 \qquad y_1(0) = 1$

$y_2' = -10y_2 + 10( (y_3)^2 + (y_4)^2 ) \qquad y_2(0) = 1$

$y_3' = -40y_3 + 40(y_4)^2 \qquad y_3(0) = 1$

$y_4' = -100y_4 + 2 \qquad y_4(0) = 1$

$x_f = 20 \qquad h_0 = 10^{-2}$

C1: $y_1' = -y_1 + 2 \qquad y_1(0) = 1$

$y_2' = -10y_2 + \beta(y_1)^2 \qquad y_2(0) = 1$

$y_3' = -40y_3 + 4\beta( (y_1)^2 + (y_2)^2 ) \qquad y_3(0) = 1$

$y_4' = -100y_4 + 10\beta( (y_1)^2 + (y_2)^2 + (y_3)^2) \qquad y_4(0) = 1$

$x_f = 20 \qquad h_0 = 10^{-2} \quad \beta = 0.1$

C3: As in C2 with $\beta = 1$

C4: As in C2 with $\beta = 10$

C5: As in C2 with $\beta = 20$

Problem Class D. Non-linear with real eigenvalues

D1: $y_1' = .2(\, y_2 - y_1\, )$ $\qquad y_1(0) = 0$

$y_2' = 10y_1 - (60- .125y_3)y_2 + .125y_3$ $\qquad y_2(0) = 0$

$y_3' = 1$ $\qquad y_3(0) = 0$

$x_f = 400 \qquad h_0 = 1.7 \times 10^{-2}$

D2: $y_1' = -.04y_1 + .01y_2y_3$ $\qquad y_1(0) = 1$

$y_2' = 400y_1 - 100y_2y_3 - 3000(y_2)^2$ $\qquad y_2(0) = 0$

$y_3' = 30(y_2)^2$ $\qquad y_3(0) = 0$

$x_f = 40 \qquad h_0 = 10^{-5}$

D3: $y_1' = -y_3 + 100y_1y_2$ $\qquad y_1(0) = 1$

$y_2' = y_3 + 2y_4 - 100y_1y_2 - 2x10^4(y_2)^2$ $\qquad y_2(0) = 1$

$y_3' = -y_3 + 100y_1y_2$ $\qquad y_3(0) = 0$

$y_4' = -y_4 + 10^4(y_2)^2$ $\qquad y_3(0) = 0$

$x_f = 20 \qquad h_0 = 2.5 \times 10^{-5}$

D4: $y_1' = -.013y_1 - 1000y_1y_3$ $\qquad y_1(0) = 1$

$y_2' = -2500y_2y_3$ $\qquad y_2(0) = 1$

$y_3' = -.013y_1 - 1000y_1y_3 - 2500y_2y_3$ $\qquad y_3(0) = 0$

$x_f = 50 \qquad h_0 = 2.9 \times 10^{-4}$

D5: $y_1' = .01 - [1+(y_1+1000)(y_1+1)](.01+y_1+y_2)$ $\qquad y_1(0) = 0$

$y_2' = .01 - (1+(y_2)^2)(.01+y_1+y_2)$ $\qquad y_2(0) = 0$

$x_f = 100 \qquad h_0 = 10^{-4}$

Problem Class E. Non-linear with non-real eigenvalues

E1: $y_1' = y_2$ $\qquad y_1(0) = 0$

$y_2' = y_3$ $\qquad y_2(0) = 0$

$y_3' = y_4$ $\qquad y_3(0) = 0$

$y_4' = ( (y_1)^2 - \sin(y_1) - \Gamma^4)y_1 +$ $\qquad y_4(0) = 0$

$(y_2y_3\ (1/((y_1)^2 + 1)) - 4\Gamma^3)y_2 +$

$(1 - 6\Gamma^2)y_3 + (10e^{-(y_4)^2} - 4\Gamma)y_4 + 1$

$x_f = 1 \qquad h_0 = 6.8 \times 10^{-3} \qquad \Gamma = 100$

E2: $y_1' = y_2$ $\qquad y_1(0) = 2$

$y_2' = 5(1 - (y_1)^2 )y_2 - y_1$ $\qquad y_2(0) = 2$

$x_f = 1 \qquad h_0 = 10^{-3}$

E3: $y_1' = -(55 + y_3)y_1 + 65y_2$ $\qquad y_1(0) = 1$

$y_2' = .0785( y_1 - y_2)$ $\qquad y_2(0) = 1$

$y_3' = .1y_1$ $\qquad y_3(0) = 0$

$x_f = 500 \qquad h_0 = 0.02$

E4: $$\underline{y}' = -U^T \begin{bmatrix} -10 & -10 & 0 & 0 \\ 10 & -10 & 0 & 0 \\ 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & .01 \end{bmatrix} U\underline{y} + G(\underline{y}); \quad \underline{y}(0) = \begin{bmatrix} 0 \\ -2 \\ -1 \\ -1 \end{bmatrix}$$

$$\underset{\sim}{U} = 1/2\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}; \quad G(\underset{\sim}{y}) = U^T \begin{bmatrix} (z_1^2 - z_2^2)/2 \\ z_1 z_2 \\ z_3^2 \\ z_4^2 \end{bmatrix} \quad \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = U\underset{\sim}{y}$$

$x_f = 1000 \qquad h_0 = 10^{-3}$

E5: $y_1' = -7.89\text{x}10^{-10}y_1 - 1.1\text{x}10^7 y_1 y_3$ $\qquad y_1(0) = 1.76\text{x}10\text{-}3$

$y_2' = -7.89\text{x}10^{-10}y_1 - 1.13\text{x}10^9 y_2 y_3$ $\qquad y_2(0) = 0$

$y_3' = -7.89\text{x}10^{-10}y_1 - 1.1\text{x}10^7 y_1 y_3 +$ $\qquad y_3(0) = 0$

$1.13\text{x}10^3 y_4 - 1.13\text{x}10^9 y_2 y_3$

$y_4' = 1.1\text{x}10^7 y_1 y_3 - 1.13\text{x}10^3 y_4$ $\qquad y_4(0) = 0$

$x_f = 1000 \qquad h_0 = 5 \text{ x } 10^{-5}$

Problem Class F. Chemical kinetics

F1: $y_1' = 1.3(y_3 - y_1) + 10400ky_2$ $\qquad y_1(0) = 761$

$y_2' = 1880[y_4 - y_2(1 + k)]$ $\qquad y_2(0) = 0$

$y_3' = 1752 - 269y_3 + 267y_1$ $\qquad y_3(0) = 0$

$y_4' = .1 + 320y_2 - 321y_4$ $\qquad y_4(0) = 0.1$

$x_f = 1000 \qquad h_0 = 10^{-4} \qquad k = e^{(20.7\ -\ 1500/(y_1))}$

F2: $y_1' = -y_1 - y_1 y_2 + 294y_2$ $\qquad y_1(0) = 1$

$y_2' = y_1 (1 - y_2)/98 - 3y_2$ $\qquad y_2(0) = 0$

$x_f = 240 \qquad h_0 = 10^{-2}$

F3: $y_1' = -10^7 y_2 y_1 + 10 y_3$ $\qquad y_1(0) = 4\mathrm{x}10^{-6}$

$y_2' = -10^7 y_2 y_1 - 10^7 y_2 y_5 + 10 y_3 + 10 y_4$ $\qquad y_2(0) = 1\mathrm{x}10^{-6}$

$y_3' = 10^7 y_2 y_1 - 1.001\mathrm{x}104 y_3 + 10^{-3} y_4$ $\qquad y_3(0) = 0$

$y_4' = 10^4 y_3 - 10.001 y_4 + 10^7 y_2 y_5$ $\qquad y_4(0) = 0$

$y_5' = 10 y_4 - 10^7 y_2 y_5$ $\qquad y_5(0) = 0$

$x_f = 100 \qquad h_0 = 10^{-6}$

F4: $y_1' = 77.27(y_2 - y_1 y_2 + y_1 - 8.375\mathrm{x}10^{-6}(y_1)^2)$ $\qquad y_1(0) = 4$

$y_2' = (-y_2 - y_1 y_2 + y_3)/77.27$ $\qquad y_2(0) = 1.1$

$y_3' = .161(y_1 - y_3)$ $\qquad y_3(0) = 4$

$x_f = 300 \qquad h_0 = 10^{-3}$

F5: $y_1' = 10^{11}(-3 y_1 y_2 + .0012 y_4 - 9 y_1 y_3)$ $\qquad y_1(0) = 3.365\mathrm{x}10^{-7}$

$y_2' = -3\mathrm{x}10^{11} y_1 y_2 + 2\mathrm{x}10^7 y_4$ $\qquad y_2(0) = 8.261\mathrm{x}10^{-3}$

$y_3' = 10^{11}(-9 y_1 y_3 + .001 y_4)$ $\qquad y_3(0) = 1.624\mathrm{x}10^{-3}$

$y_4' = 10^{11}(3 y_1 y_2 - .0012 y_4 + 9 y_1 y_3)$ $\qquad y_4(0) = 9.38\mathrm{x}10^{-6}$

$x_f = 100 \qquad h_0 = 10^{-7}$

Problem Class G. Problems with discontinuities

G1: $y_1' = y_2$ $\qquad y_1(0) = 0$

$$y_2' = \begin{cases} 2a y_2 - (\pi^2 + a^2) y_1 + 1 \text{ for } [x] \text{ even} \\ 2a y_2 - (\pi^2 + a^2) y_1 - 1 \text{ for } [x] \text{ odd} \end{cases} \qquad y2(0) = 0$$

$x_f = 20 \quad a = 0.1 \quad [x] = \text{largest integer} \le x$

G2: $y_1' = \begin{cases} 55 - 3y_1/2 & \text{for } [x] \text{ even} \\ 55 - y_1/2 & \text{for } [x] \text{ odd} \end{cases}$ $\qquad y_1(0) = 110$

$x_f = 20$

G3: $y_1' = y_2 \qquad y_1(0) = 0$

$y_2' = .01y_2(1-(y_1)^2) - y_1 - |\sin(\pi x)| \qquad y_2(0) = 0$

$x_f = 20$

G4: $y_1' = \begin{cases} -(2/21)-120(x-5)/[1+4(x-5)^2]^{16} & \text{for } x \le 10 \\ -2y_1 \end{cases}$ $\qquad y_1(0) = 1.0$

$x_f = 20$

G5: $y_1' = c^{-1}p'(x)y_1 \qquad y_1(0) = 1.0$

where

$$p(x) = \sum_{i=1}^{19}(x-i)^{4/3} \text{ and } c = \sum_{i=1}^{19} i^{4/3}$$

$x_f = 20$

## References

[1] Dugdale, S.K., Further Development of a C.A.D. Package for the Dynamic Simulation of Fluid Power Systems, M.Sc. thesis, University of Bath, 1981

[2] Leung, P.S., The Development and Testing of a Numerical Integration Method for the Digital Simulation of Fluid Power Systems, M.Sc. thesis, University of Bath, 1986

[3] Hull, S.R., The Improvement Of An Automatic Procedure For The Digital Simulation Of Hydraulic Systems, Ph.D. thesis, University of Bath, 1986

[4] Tomlinson, S.P., The Hydraulic Automatic Simulation Package (H.A.S.P.), Modelling And Simulation Aspects, Ph.D. thesis, University of Bath, 1987

[5] Chu, Y., 'Digital Simulation of Continuous Systems', McGraw Hill, 1969

[6] Anonymous, MIMIC Digital Simulation Language Reference Manual, Control Data Coporation, 1968

[7] Green, W.L., and F.H. Speckhart, 'CSMP (Continuous Systems Modelling Program )', Simulation (SCS), 34, 1980

[8] Anonymous, CSMP Program Reference Manual, IBM Corp. 1972

[9] Anonymous, ACSL User Guide/Reference Manual, Mitchell and Gausthier Associate, 1975

[10] Anonymous, CSSL-IV Refence Manual, Simulation Services, 1984

[11] Hay, J.L., and R.E. Crosbie, 'ISIM. A simulation language for microprocessors', Simulation (SCS), 43, 1984

[12] Gottwald, B.A., 'KISS. A digital simulation system for coupled chemical reactions', Simulation (SCS), 37, 1981

[13] Kays, P., and P. Rentrop, 'Generalised Runge-Kutta Methods of Order Four with Step size Control for Stiff Ordinary Differential Equations', Numerische Mathematik, 33, 55-68, (1979)

[14] Amies, G., R. Levek, and D. Streussel, Aircraft Hydraulic System Dynamic Analysis, Volume II - Transient Analysis (HYTRAN), Computer Program Technical Description, McDonnel Douglas Corporation, Report AFAPL-TR-76-43, 1977

[15] Backe, W., 'The hydraulics simulation program DSH', Proc. Inst. Mech. Eng., 199, 1985

[16] Skarbeck-Wazynski, C.M., Hydraulic System Analysis by the Method of Characteristics, Ph.D. thesis, University of Bath, 1981

[17] Enright, W.H., and J.D. Pryce, 'Two Fortran Packages for Assessing Initial Value Methods', Technical Report No. 167/83, University of Toronto, 1983

[18] Butcher, J.C., 'Implicit Runge-Kutta Processes', Math. Comp., 18, 50-54, (1964)

[19] Norsett, S.P., 'Semi-explicit Runge-Kutta methods', Mathematics and Computation, Report No. 6, University of Trondheim, 1974

[20] Alexander, R., 'Diagonally Implicit Runge-Kutta Methods For Stiff O.D.E.'s', SIAM J. Numer. Anal., 1006-1022, (1977)

[21] Petzold, L., 'Automatic selection of Methods for solving Stiff and Non-Stiff Systems of Ordinary Differential Equations', SIAM J. SCI. STAT. COMPUT 4, 136-148, (1983)

[22] Robertson, B.C., 'Detecting Stiffness with Explicit Runge-Kutta Formulas', Technical Report No. 193/87, Dept. of Computing Science, University of Toronto, 1986

[23] Balfour, A., and D.H. Marwick, Programming in Standard FORTRAN 77, Heinemann Educational Books, 1979

[24] Curtiss, C.F., and J.O. Hirschfelder, 'Integration of Stiff Equations', Proc. Nat. Acad. Sci. U.S.A., 38, 235-243, (1952)

[25] Nering, E.D., Linear Algebra and Matrix Theory, John Wiley & Sons, 1970

[26] Henrici, P., Discrete Variable Methods in Ordinary Differential Equations, John Wiley & Sons, 1962

[27] Gear, C.W., Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall Inc., 1971

[28] Johnson, L.W., and R.D. Reiss, Numerical Analysis, Second Edition, Addison Wesley, 1982

[29] Aiken, R.C., Stiff Computation, Oxford University Press, 1985

[30] Derrick, W.R., and S.I. Grossman, Elementary Differential Equations with Applications, Second Edition, Addison Wesley, 1981

[31] Lambert, J.D., and S.T. Sigurdson,'A Generisation of Multi-Step Methods for Ordinary Differential Equations', Numer. Maths., 8, 250-263, (1966)

[32] Hall, G., and J.M. Watt, Modern Numerical Methods for Ordinary Differential Equations, Clarendon Press, 1976

[33] Atkinson, K.E., An Introduction to Numerical Analysis, John Wiley & Sons, 1978

[34] Gerald, C.F., Applied Numerical Analysis, Second Edition, Addison Wesley, 1978

[35] Lambert, J.D., Computational Methods in Ordinary differential Equations, John Wiley & Sons, 1973

[36] Lapidus, L., and J.H. Seinfeld, Numerical Solution of Ordinary Differential Equations, Academic Press Inc., 1971

[37] Dahlquist, G., 'Convergence and stability in the numerical integration of ordinary differential equations', Maths. Scand., 4, 33-53, (1956)

[38] Runge. C., 'Uter die numerische Auflosung von Differentialgleichungen', Math. Ann., 46, 167-178, (1895)

[39] Kutta, W., 'Beitrag zur naherungsweisen Integration totaler Differentialgleichungen', Z. Math. Phys., 46, 435-453, (1901)

[40] Heun, K., 'Neue Methods zur approximatwen Integration der Differentialgleichungen einer unabhangigen Veranderlichen', Z. Math. Phys., 45, 23-38, (1900)

[41] Gourlay, A.R., 'A Note on Trapezoidal Mehtods for the Solution of Initial Value Problems', Maths. Comput., 24, 629-633, (1971)

[42] Aiken, A.C., Determinants and Matrices, University Mathematical Texts, Oliver and Boyd Ltd., 1964

[43] Pipes, L.A., Matrix Methods for Engineering, Prentice-Hall Inc., 1963

[44] Watson, H.D.D., and A.R. Gourlay, 'Implicit Integration for CSM III and the problem of Stiffness', Simulation, 72, 57-61, (1976)

[45] Gupta, G.K., R. Sacks-Davis, and P.E. Tischer, 'A Review of Recent Developments in solving Ordinary Differential Equations', Comput. Surv., 17, 5-47, (1985)

[46] Cragie, J.A.I., 'A variable order multi-step method for the numerical solution of stiff systems of ordinary equations', M.Sc. thesis, University of Manchester, 1975

[47] Ferziger, J.H., Numerical Methods for Engineering Application, John Wiley & Sons, 1981

[48] Hull, T.E., W.H. Enright, B.M. Fullen, and A.E. Sedwick, 'Comparing numerical methods for ordinary differential equations', SIAM. Journ. Num. Anal., 9, 603-637, (1979)

[49] Gear, C.W., 'The automatic integration of ordinary differential equations', Comput. & Comput. Machin., 14, 176-179, (1971)

[50] Lambert, J.D., 'Non-linear methods for stiff systems of ordinary differential equations', Proc. Conf. Numerical Solution of Differential Equations, ed. G.A. Watson, Springer Lecture Notes, 363, 75-88, (1973)

[51] Gautschi, W., 'Numerical integration of ordinary differential equations based on trigonometric polynomials', Numer. Math., 3, 381-397, (1961)

[52] Pizer, S.M., Numerical Computing and Mathematical Analysis, S.R.A., 1975

[53] Hull, T.E., and A.L. Creemer, 'Efficiency of predictor-corrector procedures', J. Assoc. Comput. Mach., 10, 291-301, (1963)

[54] Milne, W.E., 'Numerical Integration of ordinary differential equations', Amer. Math. Monthly. 33, 455-460, (1926)

[55] Strang, G., Linear Algebra and Its Applications, Academic Press, 1976

[56] Carnahan, B., H.A. Luther, and J.O. Wilkes, Applied Numerical Methods, Wiley, 1969

[57] Churchhouse, R.F., Numerical Analysis, University College Cardiff Press, 1978

[58] Ortega, J.M., and W.C. Rheinbolt, Iterative Solution of Non-Linear Equations in Several Variables, Academic Press, 1969

[59] Keener, M.E., and G.E. Meyer, 'Solving Differential Equations by First Order Explicit integration', Simulation, 38, 122 - 130, (1982)

[60] Spiegel, M.R., Mathematical Handbook of Formula and Tables, Schaun Outline Series, McGraw-Hill, 1968

[61] Anonymous, VAX/VMS System Services Reference Manual, Digital Equipment Corporation, 1982

[62] Anonymous, NAg Library Manual, Numerical Algorithm Group, 1981

[63] Gear, C.W., 'The Automatic Integration of Stiff Ordinary Differential Equations', I.F.I.P. Congress 68 North Holland, Information Processing 187-193 (1969)

[64] Morton, K.W., and R.D. Richtmeyer, Difference Methods For Initial Value Problems, John Wiley & Sons, 1979

[65] Bowns, D.E., R.L. Ballard, and L. Stiles, 'The effect of seal friction on the dynamic performance of pneumatic actuators', 3rd International Fluid Power Symposium, Turin, Italy, 1973

[66] Caplen, M.J.S., 'Initial Development and Testing of a New Integration Method for the solution of Stiff Ordinary Differential Equations arising in the Simulation of Fluid Power Systems', Report No. 865, School of Mechanical Engineering, University of Bath, 1986

[67] Richardson, L.F., 'The deferred approach to the limit, I - single lattice', Trans. Roy. Soc. London, 226, 299-349 (1927)

[68] Smith, G.D., Numerical Solution of Partial Differential Equations: Finite Difference Methods, Third Edition, Clarendon Press, 1985

[69] Varga, R.S., Matrix Iterative Analysis, Prentice-Hall Inc. 1962

[70] Robertson, H.H., 'The solution of a set of reaction rate equations', in Numerical Analysis: An Introduction, ed. J. Walsh., Academic Press, 1966

[71] Bowns, D.E. and A.C. Rolfe, 'Computer Simulation as a First Step Towards Computer Aided Design of Fluid Power Systems', 5th International Fluid Power Symposium, University of Durham, 1979

[72] Caney, K.C. 'Integration Across Discontinuities in Ordinary Differential Equations using Gear's Method', Internal Report No. 478, School of Engineering, University of Bath, 1979

[73] Caplen, M.J.S. [CAPLEN.HASP.INHASP], Subdirectory, VAX 750 Computer, School of Engineering, University of Bath, 1986

[74] Wang, L.M., Untitled, Ph.D. thesis, to be published, University of Bath, 1988

[75] Bowns, D.E. and A.C. Rolfe, 'The Digital Computation of Pressures and Flows in Interconnected Fluid Volumes, using Lumped Parameter Theory', 4th International Fluid Power Symposium, B.H.R.A. Fluid Engineering, 1975

[76] Enright, W.H., T.E. Hull, and B. Lindberg, 'Comparing Numerical Methods for Stiff Systems of O.D.E.'s ', B.I.T., 15, 10-48, (1975)

[77] Addison, C.A., 'Implementing a Stiff Method based upon the Second Derivative Formulas', Dept. of Computer Sc. Tech., Rep. No. 130, University of Toronto, 1980

[78] Merson, R.H., 'An operational method for the study of integration processes', Proc. Symp. Data processing, Weapons Research Establishment, Salisbury, S. Australia, 1957

[79] Fehlberg, E., 'Klassische Runge-Kutta- Formeln funfter und siebenter Ordnung mit Schrittweiten-Kontrolle', Computing, 4, 93 106; Corrigendum: Computing, 5, 184, (1969)

[80] Fehlberg, E., 'Klassische Runge-Kutta- Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Warmeleitungsprobleme, Computing, 6, 61-71, (1970)

[81] Butcher, J.C. 'Integration processes based on Radau quadrature formulas', Math. Comp. 18, 233-244, 1964

[82] Ehle, B., On Pade approximations to the exponential function and A-stable methods for the solution of initial value problems', Ph.D. thesis, University of Waterloo, Ontario, Canada, 1969

[83] Dahlquist, G., 'A special stability problem for linear multistep methods', BIT, 3, 27-43, (1963)

[84] Crouziex, M., Sur l'approximation des equations differentielles operationnelles lineaires par des methodes de Runge-Kutta, These presentee a l'Universite Paris VI, Paris, 1975

[85] Prothero, A., and A. Robinson, 'On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations', Math. Comp., 28, 145-162, (1974)

[86] Hindmarsh, A.C., GEAR: Ordinary differential equation system solver, LRL Rep. UCID-30001, Revision 3, 1974

[87] Cash, J.R., 'Diagonally Implicit Runge-Kutta Formulae with Error Estimates', J. Inst. Maths Applics, 24, 293-301, 1979

[88] Richards, C.W., K. Wade, and M.G. Everett, 'SARK - a type-insensitive Runge-Kutta code', Internal report, Thames Polytechnic, 1987

[89] Shampine, L.F., and M.K. Gordon, Computer Solution of Ordinary Differential Equations, W.H. Freeman, San Francisco, 1975

[90] Shampine, L.F., 'Stiffness and non-stiff differential equation solvers, II: Detecting stiffness with Runge-Kutta methods', ACM Trans. Math. Software, 3, 44-53, (1977)

[91] Shampine, L.F., 'Lipschitz constants and robust o.d.e. codes', Computational Methods in Nonlinear Mechanics, North-Holland, Amsterdam, 427-449, (1980)

[92] Shampine, L.F., 'Type-insensitive O.D.E. codes based on implicit A-stable formulas', SAND79-244, Sandia National Laboratories, Livermore, CA, 1979

[93] Hindmarsh, A.C., LSODE and LSODI, two new initial value ordinary differential equation solvers, ACM SIGNUM Newsletter, 15, 4, 1980

[94] Cash, J.C., 'A class of implicit Runge-Kutta methods for the numerical integration of stiff O.D.E.'s ', S. ACM., 22, 504-511, 1975

Correction Camera

D

Please film: Target Number

Leave a space: then film

Correction ends here!

At exposure setting

Finish with:

8spaces

16spaces

Date filmed Sig.