Digital differential analysers

Author: Webster, Max B.

Publication Date: 1984

DOI: https://doi.org/10.26190/unswworks/6570

License: https://creativecommons.org/licenses/by-nc-nd/3.0/au/
Link to license to see what you are allowed to do with this resource.

Downloaded from http://hdl.handle.net/1959.4/59841 in https://unswworks.unsw.edu.au on 2023-10-31
DIGITAL DIFFERENTIAL ANALYSERS

by

Max B. Webster

A Thesis
submitted for the degree of
Doctor of Philosophy

at

the University of New South Wales
December 1984
I hereby certify that the work contained in this thesis has not been submitted for a higher degree to any other university or institution.

SIGNATURE ..................................  

DATE...
Declaration

I hereby declare that this Thesis is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of a University or other Institute of higher learning, except where due acknowledgement is made in the text of the Thesis.

Max Webster
Abstract

Hitherto, Digital Differential Analysers (DDAs) have been restricted to the use of a predetermined, constant integration step-size. Second Order Difference transmission, developed by Bywater, allows the potential problem of variable intermodule data path width associated with variable step-size (VSS) integration to be avoided, and VSS overcomes the starting problem of Second Order Differences.

This Thesis details the modifications necessary for Second Order Difference, Extended Resolution DDA modules to be able to cope with VSS. The resulting VSSDDA structure is eminently suitable for special purpose, dedicated applications where foreknowledge of solution characteristics permits external control of step-size. To allow for the possibility of more general use, two proposed techniques for achieving autonomous step-size control are described.

The application of truncation error analysis to nonlinear differential equations (DEs) is investigated, and a new class of nonlinear DEs, developed for the testing of VSS algorithms, is presented. With these equations, VSS integration can yield speed improvement factors in excess of 100 to 1 over constant step-size integration. A selection of these equations is combined with published test problems to yield a comprehensive set for testing VSSDDAs.

Finally, a new method for solving stiff DEs, based on existing explicit integration methods and suitable for DDA hardware implementation, is proposed. Solution speed improvement factors of up to 10 to 1 over standard explicit methods have been achieved.
Acknowledgements

Dr. Paul W. Baker has been responsible for the supervision of the Thesis, and many of the ideas presented in it originate from him. Diligent proofreading by Paul has also had significant influence on the final presentation of the manuscript.

Dr. Phil G. McCrea played a similar, though unofficial, role during the latter stages of the project, making valuable contributions to my understanding of DDAs and to the organisation of the Thesis.

Dr. Peter C. Maxwell offered assistance earlier in the project, especially in the proofreading of Chapter 7, and the encouragement of Trevor N. Vickers ensured the completion of the Thesis.

The tolerance and understanding shown by Alice and my flatmates, especially during the final stages, also contributed significantly through preserving my wellbeing.

Last, but not least, I would like to thank God, without whom none of this would have been possible.
CONTENTS

Declaration.......................................................... ii
Abstract.............................................................. iii
Acknowledgements.................................................... iv

INTRODUCTION....................................................... 1

Numerical Integration Error...................................... 3
The Three Alternatives........................................... 4
Motivation for Continued Research into DDAs.................. 6
Aims and Contributions of this Thesis.......................... 7
Thesis Organisation............................................... 8

1. A SHORT HISTORY OF DDAs...................................... 9

1.1 Applications of DDAs.......................................... 12

1.2 Developments in DDA Design.................................. 13

1.2.1 Binary Floating Point DDAs ................................. 14
1.2.2 Extended Resolution DDAs................................... 15

1.3 Recent Design Proposals and Realisations................ 17

1.4 A Potential Future for DDAs.................................. 20

2. BASIC DDA ARCHITECTURE AND ERROR ANALYSIS.............. 21

2.1 DDA modules..................................................... 22

2.1.1 Integration................................................... 25
2.1.2 Multiplication............................................... 27
2.1.3 Summation..................................................... 28
2.1.4 Forcing Functions........................................... 29

2.2 Truncation Error Analysis..................................... 29

2.2.1 Preliminary Definitions..................................... 30
2.2.2 Analysis....................................................... 31
2.2.3 Suppression of Second Order Truncation Error........... 33
2.2.4 Suppression of Third Order Error.......................... 34
2.2.5 Further Error Analysis..................................... 36

2.3 Second Order Differences...................................... 36

2.3.1 Integration................................................... 38
2.3.2 Step-size Restriction and Starting Problem............. 38
2.3.3 Multiplication by Constants................................ 41
2.3.4 Multiplication by Variables................................ 42
3. INTRODUCING VARIABLE STEP-SIZE TO DDAS....................... 45
   3.1 Effect of VSS on DDA Architecture.............................. 46
      3.1.1 Effect of VSS on Second Order Differences 47
      3.1.2 VSS Integrator Design 49
      3.1.3 VSS Multiplier Design 51
   3.2 Error Suppression with Variable Step-Size....................... 52
      3.2.1 Second Order Suppression 54
      3.2.2 Third Order Suppression 55
      3.2.3 The Incorrect Alternative 57
   3.3 Summary.......................................................... 58
4. AN APPROPRIATE INTEGRATION METHOD FOR VSSDDAs...................... 60
   4.1 Choice of Integration Algorithm Class............................ 61
   4.2 Choice of Order.................................................... 62
      4.2.1 Third Order Advantage 62
      4.2.2 Second Order Advantage 63
      4.2.3 Step-size Restriction 66
   4.3 Second Order P-C Register Transfers............................... 67
   4.4 Special Restrictions on Step-size Change........................ 69
      4.4.1 When Halving is Error Prone 70
      4.4.2 Simplified Error Estimates 72
   4.5 Summary.......................................................... 73
5. AUTONOMOUS DDA STEP-SIZE CONTROL.................................... 74
   5.1 Differences between Residues..................................... 77
   5.2 Employing Redundant Accuracy.................................... 79
      5.2.1 First Order Difference Transmission 80
      5.2.2 Second Order Difference Transmission 80
      5.2.3 Higher Order Difference Transmission 82
   5.3 Summary.......................................................... 84
6. DDA SOLUTION OF NONLINEAR DEs........................................ 85
   6.1 Truncation Error Analysis with Nonlinear Equations............. 86
      6.1.1 Suppression of Second Order Error 89
      6.1.2 Suppression of Third Order Error 90
   6.2 When m×M-bit Multiplication is Inadequate.................... 91
      6.2.1 m×m-bit Multiplication 93
      6.2.2 Misuse of Residues 95
      6.2.3 M×M-bit Multiplication 96
   6.3 Second Order P-C Integration..................................... 96
   6.4 Summary.......................................................... 98
7. A NEW CLASS OF DEs FOR TESTING VSS INTEGRATION ............... 99
  7.1 Test Problems for VSS Integration .......................... 99
  7.2 VSS Equations : Coefficient and Superscript Selection ...... 104
  7.3 Comparing Test Problems ...................................... 105
  7.4 Comparison Results ........................................... 107
  7.5 Conclusions .................................................. 110
7.6 COMPARISON RESULTS ............................................ 110

8. TEST PROBLEMS AND RESULTS .................................... 111
  8.1 Test Problems suitable for VSSDDAs ......................... 112
     8.1.1 Single Equations - from Hull et al ........................ 114
     8.1.2 Small Systems - from Hull et al .......................... 118
     8.1.3 Moderate Systems - from Hull et al ...................... 122
     8.1.4 Orbit Equations - from Hull et al ....................... 129
     8.1.5 Higher Order Equations - from Hull et al................ 131
     8.1.6 Problems from Krogh ...................................... 135
     8.1.7 VSS Equations ........................................... 140
  8.2 Test Results .................................................. 146
     8.2.1 Easy Test Problems ....................................... 149
     8.2.2 Moderate Test Problems ................................... 152
     8.2.3 Hard Test Problems ...................................... 156
     8.2.4 Second Order Solution Accuracies ....................... 159
  8.3 Summary and Comment .......................................... 160

9. SOLUTION OF STIFF DEs ........................................... 161
  9.1 Explicit Integration ........................................... 164
  9.2 Matrix Representation of DEs .................................. 167
  9.3 Implicit Integration .......................................... 168
     9.3.1 Hardware Implicit Integration ............................ 171
  9.4 New Explicit Stiff Integration Technique .................... 173
     9.4.1 Multiple Step-size Integration ........................... 174
     9.4.2 Choice of Explicit Algorithm ............................. 176
     9.4.3 Dual Step-size Heun Integration ........................... 177
  9.5 Summary ...................................................... 179
  9.6 Further Research .............................................. 179

CONCLUSIONS .......................................................... 181

VSSDDAs .............................................................. 181
Solution of Nonlinear DEs ........................................... 182
New Test Problems ................................................... 183
Solution of Stiff DEs ................................................ 184
REFERENCES................................................................. 185

A. DISCRETISATION ERROR ANALYSIS................................... 191

A.1 Accuracy of Second Order P-C Algorithm......................... 192
A.1.1 Second Order Algorithm - Halved Step-size 193
A.1.2 Second Order Algorithm - Doubled Step-size 193

A.2 Accuracy of Third Order P-C Algorithm............................ 194
A.2.1 Third Order Algorithm - Halved Step-size 195
A.2.2 Third Order Algorithm - Doubled Step-size 197

B. DERIVATION OF PREDICTOR-CORRECTOR ALGORITHMS.................... 199

B.1 Second Order Predictor-Corrector Algorithm....................... 201
B.1.1 Second Order - Halved Step-size 202
B.1.2 Second Order - Doubled Step-size 203

B.2 Third Order Predictor-Corrector Algorithm......................... 204
B.2.1 Third Order - Halved Step-size 205
B.2.2 Third Order - Doubled Step-size 208

C. TRUNCATION ERROR ANALYSIS.............................................. 211

C.1 Register Transfers for Linear Equations......................... 211
C.1.1 Second Order Suppression 214
C.1.2 Third Order Suppression 215

C.2 Register Transfers for Nonlinear Equations....................... 217
C.2.1 Second Order Suppression 220
C.2.2 Third Order Suppression 221

D. FINE CONTROL OF SECOND ORDER P-C STEP-SIZE....................... 222

D.1 Two adjacent bits of h set.............................................. 223
D.1.1 Step-size Decreased by 1/3 224
D.1.2 Step-size Decreased by 1/4 225
D.1.3 Step-size Increased by 1/3 226
D.1.4 Step-size Increased by 1/2 227

D.2 Any two bits of h set.................................................. 228
INTRODUCTION

The computer aided simulation of complex systems is a well established industry which continues to grow in importance. Some areas of application, such as flight simulation for pilot training and nuclear plant simulation for the testing of safety mechanisms, are well known through recent media coverage, but most are not of sufficiently immediate public interest to attract such attention. For example, computer simulation is applied extensively in the analysis of stress on building structures and mechanical devices under a range of loading conditions, the modelling of complex industrial equipment to permit simulation of the effects of various component failures, electrical power distribution system fault analysis, and so on.

The basis of most simulation is the modelling of a physical system using a set of differential equations (DEs), which, given appropriate initial conditions, allow the response of the physical system to be predicted from the response of the model. The advantage of the simulation of physical systems is obvious: the response of a system to conditions which may lead to the destruction of some of its components can be examined without incurring the risk of expensive "accidents". This facility is of particular benefit where human life and health are at stake, as they may be in the case of aircraft or nuclear reactor malfunctions.

Simulation is generally implemented using software numerical integration subroutines to solve the system of DEs which model the physical system. For complex systems, this is a very time-consuming operation, because not only does a large system of DEs require considerable computer time to solve, but the nature of simulation is
that many computer runs will be desirable - to simulate a variety of conditions. One of the prime examples is the analysis of the stability of electrical power distribution systems, which, according to Stott [1] involves the solution of thousands of equations, starting from a variety of initial conditions, each case requiring possibly an hour of CPU time on a large general purpose digital computer. More recently, Frowd et al [2] describe the required computation for a long-term power system simulation as "prohibitive".

There is an alternative to the software solution of DEs which permits significant improvement in solution speed: special purpose hardware. The hardware numerical solution of DEs was originally performed only by analog computers, but the accuracy limitations associated with analog devices led to the invention of a second hardware method: the Digital Differential Analyser (DDA). The DDA is a short word-length, special purpose digital computing device designed specifically for the numerical solution of DEs.

DDAs were considered at one time to show great potential but are now commonly overlooked in favour of software. The popularity of software for the solution of DEs is due mainly to the superior flexibility of software, and to the improved performance and continuing reduction in the cost of general purpose digital computers. In spite of the resulting dominance of software, two features continue to make the DDA attractive for the solution of DEs: the improved solution speed inherently available with special purpose hardware, and the DDA's unique ability to control numerical integration error.
Numerical Integration Error

Digital methods of DE solution, both hardware and software, utilise one or more of the well known numerical integration formulae to produce a series of points which approximate the solution of the DE, and which are separated by increments along the axis of the independent variable. The points are produced by accumulating the area under successive discrete sections of the derivative of the solution, but are of limited accuracy due to two types of error: discretisation error and truncation error.

Discretisation error arises from a combination of the discrete (rather than continuous) nature of the solution, and the inability of practical integration formulae to calculate exactly the area under each discrete section of the derivative. This type of error could "theoretically" be eliminated, either by reducing the increment size to 0 to generate a continuous solution, or by implementing an integration formula containing an infinite number of terms in order to calculate the areas exactly - obviously, numerical integration of any DE could not proceed beyond its initial point under either of these conditions. The practical application of the theory is that discretisation error can be reduced by implementing a higher order integration formula (with more terms), and/or by reducing the size of the increment (the integration step-size).

The other source of solution error introduced by digital methods is the loss of accuracy inherent in the use of finite word-lengths. It is commonly termed "round-off error", but is, in the context of DDAs, more correctly referred to as truncation error, because no rounding is performed in a DDA. Truncation error
accumulates with each step of integration; hence it can be reduced by increasing the step-size so that fewer steps are required, and/or by realizing longer word-lengths to directly reduce the truncation error introduced at each step.

The apparent conflict of interests concerning the integration step-size, which should be reduced to limit discretisation error, but increased to reduce truncation error, is resolved in the DDA by the use of residue retention registers. Residue registers are used to retain the less significant bits from high-precision arithmetic calculations for addition at the appropriate point during the next integration step, so that no information is lost due to the normal truncation process. Residue retention registers are unique to DDAs, and allow the accumulation of truncation error to be restricted to within a predefinable upper bound - in contrast to other numerical integration techniques, which allow low significance bits to be lost, and truncation error to accumulate unchecked.

The Three Alternatives

As indicated above, the three basic approaches to the numerical solution of DEs employ either analog computers, appropriately programmed general purpose digital computers, or special purpose digital hardware structures (commonly called DDAs).

Analog computers provide low accuracy solutions to DEs at the high speed inherent in the use of hardware. To avoid the inconvenience of patch-board programming and initialisation procedures, analog computers are now rarely used unless interfaced to general purpose digital hardware in a hybrid configuration, which
provides the flexibility of software initialisation and interconnection of modules. The accuracy of an analog solution is not determined by the discretisation and truncation errors described above, since numerical integration is not employed, and because the solution generated is continuous. Instead, their low accuracy is due mainly to amplifier drift, and also to other limitations in the accuracy of components.

Advances in technology and design during the past 25 years have made possible the real-time solution of small systems of DEs in software, so that the general purpose digital computer has largely replaced hardware methods of solution. Software solutions offer greater flexibility and avoid the cost of special purpose hardware. To reduce the effects of discretisation and truncation error, a combination of high (and sometimes variable) order integration methods with long machine word-lengths is relied on, and these features are coupled with variable integration step-size to maximise solution speed. In addition, the flexibility of software permits the application of implicit integration methods, which were shown by Gear [3] to be particularly useful for the solution of stiff DEs.

Digital Differential Analysers avoid the accuracy limitations of analog hardware while retaining the speed improvement of dedicated hardware over software. The technique of residue retention allows the accumulation of truncation error to be suppressed, in spite of the typically short word-length of DDAs, so that overall solution accuracy can be guaranteed by the choice of appropriately small step-size to reduce discretisation error.
Motivation for Continued Research into DDAs

The motivation for the research described in this Thesis can be attributed to three factors, which are outlined in the following three paragraphs.

Firstly, dedicated hardware offers DE solution speeds superior to those attainable with software. DDAs are thus most appropriate in those special purpose applications where very high speed solution of a particular DE is required. A DDA would be designed specifically for such a DE, with modules hardwired together for maximum speed - avoiding the time delays otherwise introduced with software interconnection. DDAs are also to be favoured for the simulation of large systems, such as electrical power distribution systems as mentioned above, where software solution is too slow for practical simulation.

Secondly, the ability of the DDA to guarantee the suppression of the accumulation of truncation error is itself sufficient reason to continue research in the field, because no other method offers this mathematically attractive advantage. The long word-lengths relied on in software packages are able only to delay the accumulation of truncation error, not guarantee its suppression.

Thirdly, two major limitations in DDA techniques remain:
1. DDAs still use fixed step-size integration;
2. they are not able to adequately handle stiff DEs.
The implicit integration methods found to be useful for the software solution of stiff DEs are deemed unsuitable for hardware implementation due to the need for matrix inversion.
Introduction

In spite of the overwhelming emphasis on the software solution of DEs over the past decade or more, articles describing the construction and features of new DDAs continue to appear in the literature. Publications from researchers in a wide range of countries are available, including England [4,5,6,7], India [8], Japan [9,10], Israel [11], Italy [12], Germany [13], Australia [14,15], and Czechoslovakia [16,17], suggesting that the DDA still retains the interest of a significant number of research establishments.

Aims and Contributions of this Thesis

The three principle aims of the research described in the following Chapters are:

1. to investigate the feasibility of incorporating automatically varying step-size into DDA integration;
2. to analyse DDA suppression of truncation error during solution of nonlinear equations;
3. to introduce a new stiff integration technique, based on existing explicit integration methods and suitable for DDA hardware implementation.

A result of this research has been the discovery of a new class of DEs, which has proved to be particularly useful in the testing of VSS integration algorithms. These new equations have been combined with a selection from the literature to produce a comprehensive set of DEs for testing VSS DDA integration.

Thus original contributions are considered to have been made in a total of four separate areas within the DDA field.
Thesis Organisation

The first Chapter presents a summary of the history of DDAs, which is followed in Chapter 2 by a detailed description of the state of the art.

The introduction of variable step-size (VSS) integration to DDAs is developed through Chapters 3, 4, and 5, beginning with the modifications to current DDA modules necessary to permit VSS integration (Chapter 3). This is followed by the selection of the most appropriate integration method for autonomous DDA step-size control (Chapter 4), then by a description of two alternative means of producing suitable discretisation error estimates to be used as the basis for autonomous DDA step-size control (Chapter 5).

Nonlinear error analysis is tackled in Chapter 6, and the new class of nonlinear DEs described in detail in Chapter 7.

Chapter 8 contains the combined set of test problems used to test the performance of VSSDDAs, and presents the test results, obtained by software simulation, along with discussion of their implications.

The solution of stiff DEs is investigated in Chapter 9, and the new explicit stiff integration technique is explained with reference to preliminary simulation results obtained so far.
1. A SHORT HISTORY OF DDAs

Historical accounts, such as those contained in monographs by Mayorov and Chu [18] and by Sizer et al [19], provide insight into the technological and political factors which led to the development of the DDA. According to Mayorov [18], DDAs were first developed in 1949 out of a desire to combine the advantages of the two major categories of computers: analog and digital.

The history of analog computers is traced back by Sizer [19] to the invention of the slide rule by Oughred in 1630, in which scale length is the physical analog of numbers in a logarithmic base. A device more easily recognisable as an ancestor of today's analog computer, called a planimeter, appeared in the first half of the 19th century. It was a form of mechanical integrator, which was significantly improved on in 1876 by Professor J. Thompson, who introduced a mechanical integrator based on a disc-sphere-cylinder assembly. Two years later, the use of mechanical integrators in a machine capable of the solution of DEs was described by Professor Sir W. Thompson (later Lord Kelvin). At that stage the level of technology did not allow production of components giving sufficient accuracy or integrator gain for the generation of meaningful solutions to DEs, but such solutions were possible by the 1930s, when a number of mechanical differential analysers capable of solving DEs of up to eighth order were built.

Digital computers evolved somewhat later than their analog counterparts, possibly because the concept of discrete valued variables is further removed from nature than the underlying concepts of analog machines, in which all variables are continuous functions of...
Chapter 1

History of DDAs

time. Sizer [19] recounts that Babbage’s vision, in the 1820s, of a universal machine capable of dealing with a wide variety of problems, led him to establish the principles on which current general purpose digital computers are based. Due to financial and technological limitations, Babbage was unfortunately unable to realise his ideas of a "store" for holding results, a "mill" for performing logic, and his decision elements, but the rapid expansion of industry and commerce over the latter half of the 19th century increased the demand for mechanisation of the handling of commercial data. This led eventually to the completion in 1944 of the first general purpose digital computer, known as MARK I, by Professor Aiken of Harvard in collaboration with International Business Machines. MARK I, an electro-mechanical version of Babbage’s analytical engine, was followed a year later by ENIAC, the first thermionic computer. Bywater [20] records that parallel development in England led to the production of the PILOT ACE and English Electric DEUCE machines soon afterwards.

In the 1940s then, the advantages of the analog computer were seen to be the high speed with which DEs could be solved, and the continuity of the solution generated - both factors of great importance for the ideal operation of real-time systems. The major disadvantages included the dependence of solution accuracy on component accuracy and stability, the fact that the independent variable always had to be time, and the inconvenience of the patch board programming procedure. Digital computers were inherently more accurate, stable and flexible than analog computers. They were more easily reprogrammed to solve different DEs, but for the pressing military applications of the day were too big, consumed too much...
power, and had a maximum solution rate which required inaccurate interpolation between successive solution points - they were too slow!

The reliability of both digital and analog computers was greatly improved during the 1942-1946 period, when vigorous worldwide research motivated by military zeal guaranteed the design and production of a new class of electronic and electro-mechanical devices. Hybrid computers were also produced, permitting the speed of analog hardware to be coupled to some extent with digital flexibility. This simplified the setting of initial conditions, but the accuracy limitations remained, patch board programming of the analog section was still necessary, A-to-D and D-to-A conversions were now required, and time was still the only possible independent variable.

At this point, the DDA entered history, offering the inherent speed of a hardware DE solver together with the flexibility and accuracy of digital technology. DDAs were also smaller and more economical than general purpose digital computers, and permitted the solution of DEs dependent on a variable other than time, thus overcoming the major limitation of the analog computer. The first DDA was built in the USA in January 1950 [18], and within a few years a wide range of machines was available. Although the DDA was initially unable to match the speed of analog computers, advances in design quickly reduced the performance gap. TRICE [21] was produced in 1958, with parallel configuration of integrators giving an increase in speed of three orders of magnitude over previous DDAs, then a further order of magnitude improvement was achieved by the introduction of parallel-serial arithmetic by Bradley and Genna [22], who achieved an iteration rate of one megacycle in 1962.
As well as potentially solving the problems then current in the solution of DEs by either analog or general purpose digital computers, the DDA also offered a remedy for the remaining shortcomings of hybrid computers. Replacing the analog section of a hybrid machine by a DDA obviously eliminated the need for A-to-D/D-to-A conversion of data, and in addition made the manual patchboard obsolete as software interconnection of elements provided a preferable alternative in the resulting all-digital machine.

1.1 Applications of DDAs

Developed as they were soon after World War 2, early applications of DDAs were predominantly inspired by a desire for military supremacy. In summarising the DDA research effort of many of the aircraft companies and electronics corporations in the 1950s, Mayorov [18] continually refers to the application of each newly produced DDA to the solution of systems of DEs relevant to such problems as the navigation of naval vessels and aircraft (notably fighters and bombers), trajectory calculations for ballistic missiles, point of interception of moving objects, inertial navigation and guidance systems for intercontinental ballistic missiles and other missiles, and so on. Special types of DDAs were produced for use on artillery ranges, and in aircraft, submarines, missiles, rockets and satellites where light weight was required.

Of course civilian air and sea travel also benefitted from the extensive research, and some of the more pacific members of the global community found applications for DDAs in the kinetics of chemical reactions, analysis of faults on electrical systems, and generally in
real-time automatic control systems (e.g., machine tool control) and sampled data applications.

More recently DDAs have been used to solve the reentry equations [23], and DDA principles have been applied to digital filters [24,25,26], signal processing [27] and stepping motor control [28]. DDAs have also been employed in the high speed generation of complex functions for computer graphics [6,29].

1.2 Developments in DDA Design

For a short time, DDAs captured the interest of a large number of researchers because the DDA was a relatively simple device, able to provide high speed solutions to DEs at low cost. At one time, according to Stratton [19], "the DDA, operating with a small general purpose computer to set initial conditions and perform logic functions, represented the most advanced form of computer in the navigation field". However, technological advances permitted large increases in the speed of, and reductions in the cost of, general purpose machines, which led to the relegation of DDAs. The major shortcomings of the DDA seem to have been scaling problems associated with the fixed point nature of DDA register contents, and the necessity to decrease step-size if higher accuracy was required.

Early responses to the scaling problem led to comprehensive sets of rules [30,31] for the prescaling of variables/constants based on expected maximum values. Goldman [23] was responsible for a design using shared integrators and implementing automatic rescaling in order to eliminate problems of register overflow in solving the Chapman predictive reentry guidance equations.
More significant developments occurred around 1970, when the simplicity of early DDA hardware structures was abandoned. The reduction in the cost of hardware, which originally led to the downfall of the DDA as mentioned above, encouraged researchers to implement more complex designs in their attempts to reinstate the DDA. The principle innovations were the extended resolution DDA (ERDDA), introduced by McGhee and Nilsen [32], and the binary floating point DDA (BFPDDA), described shortly after by Elshoff and Hulina [33].

1.2.1 Binary Floating Point DDAs

The BFPDDA was a direct attempt to solve the scaling problem of the traditional fixed point DDA. The new machine performed all arithmetic operations using floating point notation, and required full floating point representation of all values. Unfortunately, the need for increment values to be accompanied by exponents produces a new type of interconnection problem: where data paths originally needed to be only one or two bits wide for the transmission of incremental values between DDA elements, they were now required to carry the exponents as well, thus magnifying the interconnection bit count and associated construction costs.

Subsequent research involving BFPDDAs has been limited. An attempt was made by Philokyprou and Halatsis [34] to combine the advantages of BFPDDAs with those of ERDDAs, but because ERDDAs also required multiple interconnections between elements, the combination of the two designs further magnified the new interconnection problem. More recently, Hannington [35], has discovered a novel way to produce residues in BFPDDAs, but no solution to the remaining major problems has emerged.
1.2.2 Extended Resolution DDAs

The distinguishing feature of the ERDDA is the multi-bit increment. Transmitting multiple bits between modules, instead of the original 1-bit increments, allows larger steps to be taken without reduction of register length and corresponding loss of machine accuracy. Second (or higher) order integration methods were found to be necessary to maintain accuracy with these larger step-sizes, and multiple interconnections were now needed for the transmission of the multi-bit increments. Function evaluation was also complicated by the need for actual multiplications where simple additions were previously sufficient.

In contrast to the limited interest shown in BFPDDAs, the research effort into solving the shortcomings of fixed point ERDDAs has produced significant positive results in the areas of interconnection problems, scaling problems, and error analysis.

Bywater's proposal [36] of Second Order Difference transmission has made possible the reduction of the number of interconnections necessary between ERDDA modules. The concept depends on an appropriate choice of DDA parameters (see Chapter 2), which guarantees that the difference between successive increments will be small. Simulation indicates that this Second Order Difference rarely exceeds 1 in magnitude, which allows significant reduction of interconnections, except of course at the first step. The lack of a previous increment value for use in the first iteration produces a starting problem, as a full precision increment must be transmitted initially, requiring multiple interconnections.

To solve the scaling problem, Baker and McCrea [29,37] describe the introduction of Global Sliding Point (GSP), which
involves the use of a single, universal exponent register which applies to all machine registers. When overflow is detected in any register, all registers are shifted right one bit, and the exponent register incremented. This method retains the simplicity of fixed point architecture and arithmetic, while providing the extended range of floating point representation. The exponent does not need to be transmitted along with the increments, as was the problem in BFP DDAs, so interconnection does not become complicated. Potential problems arise with GSP if states and/or constants differ greatly in magnitude, as accuracy may then be lost due to right-shifting of low magnitude values.

For a complete error analysis, both discretisation error and truncation error need to be considered. Discretisation error, being a function of the integration method chosen, has been extensively analysed since before the invention of DDAs, and is not affected by DDA implementation of an integration algorithm. Truncation error is of much greater interest in DDAs. A major contribution to truncation error analysis has been made by Baker and McCrea [38,39,40,41], who developed a new method of analysis which can be used to prove that the DDA residue register permits the accumulation of truncation error to be suppressed to within a specified upper limit. The analysis method has proven useful for detecting errors in proposed sets of register transfers, and has also led to modified sets of register transfers for the suppression of higher orders of truncation error. Previous DDA error analysis had concentrated on discretisation error (for unknown reasons), with the exception of the analysis by statistical methods reported by Turtle [42].
In addition to the above contributions to the theory and design of ERDDAs, considerable effort has also been put into the use of higher (than second) order integration algorithms [29,43,44,45] in DDAs. Problems arise with these higher order algorithms because division by 3, or by a multiple of 3, is inherent in the formulae and cannot be implemented simply by right shifting. In a DDA, multiplication by \(\frac{1}{3}\) is implemented in preference to division by 3, so that residue retention can be applied for truncation error suppression, but because \(\frac{1}{3}\) does not have an exact binary representation, multiplication by the nearest approximation will inevitably lose some of the accuracy gained by choice of the higher order method.

### 1.3 Recent Design Proposals and Realisations

Since ERDDAs and BFPDDAs were first introduced in 1970, interest in DDAs has revived significantly, leading to a number of proposed and implemented systems. A summary is given here to indicate the extent of the interest exhibited during the period from 1972 to 1984.

In the early 1970s, Bywater and Lovering described two realisations of ERDDAs. The first [4] employed Euler integration with a trapezoidal correction, 12-bit word-length with 4-bit increment transmission paths, and was composed of 16 elements. Their second DDA [5] realised Second Order Difference transmission with 16-bit word-length and 64 elements, each of which was capable of performing any of the basic functions of integration, multiplication and summation required in the solution of DEs. The integration method was
improved by the use of an Adams prediction (instead of Euler). In both machines, the desired interconnection of elements was achieved by software, which was said to account for 75% of the minimum iteration time.

Hannington and Whitehead [6,7] have produced a 16 element ERBFPDDA using 16-bit mantissa with 8-bit exponent, parallel arithmetic, and serial configuration of integrators. This machine had a minimum iteration time of 10.5μS, and a worst case iteration time of 1mS due to its serial configuration.

Some interest has been indicated in India [8], with the realisation of a two element DDA out of standard ICs. Integrator configurations are given for the generation of linear and circular motion.

In Japan, Stieltjes integration has been realised [9] in a DDA boasting high speed and accuracy. Also in Japan, a DDA based on an array processor system has been built [10], with software interconnection of elements which are realised in the form of microprogrammed 16-bit processors. Integrator/multiplier times of around 25μS are quoted, using second order Adams-Bashforth integration with multi-bit increments.

Brafman and Reuter [11], from Israel, report the simulation of a proposed system, similar in design to a microprocessor, which employs floating-point arithmetic, and uses a stack to avoid redundant operations and reduce access time for frequently needed quantities. Unfortunately, they have apparently failed to understand the importance of the residue register for error control, for they state that residue registers are unnecessary, but make no attempt to support this claim.
Another microprogrammed system has been built more recently in Italy [12], aiming to provide a low cost DDA emulator for use in medium speed large dynamic systems on-line simulation. The implementation provides 1000 elements for interconnection as desired.

Realisation of a DDA in Germany [13], capable of generating accurate results for both linear and nonlinear DEs, is the subject of the first article on DDAs to appear in the 1980s.

A 16 element DDA, realising second order integration, third order truncation error suppression, and global sliding point arithmetic has been constructed at The University of New South Wales, Australia [15], proving that these innovations in DDA theory and design do function effectively in practice.

From Czechoslovakia, two reports of research and development indicate that work has been done on automatic patching [16] for both varieties of hybrid computers, and that a DDA has been built [17] for the solution of ordinary or partial DEs.

Emphasis in DDA activity has noticeably shifted over the past two decades, from the initial striving for maximum speed to a greater interest in economical and flexible implementations. References to microprogramming and microprocessor based designs have naturally become more common in the literature, as the DDA residue register allows accurate solution of DEs using the short word-lengths common in microprocessors. A publication by McCrea and Witten [46] describes the success of a microprocessor software implementation of DDA principles in conjunction with partial double-precision arithmetic, which is frequently employed to reduce the effect of truncation error. The program was found to guarantee much greater accuracy than partial double-precision alone, in return for a very slight increase in
computation, as might be expected from the theory behind residue register suppression of truncation error.

1.4 A Potential Future for DDAs

Although recent emphasis has shifted towards economical and flexible DDA implementations, the growth in the popularity of general purpose digital computers, combined with the ready availability of sophisticated software numerical integration subroutines, makes the use of DDAs as general purpose DE solvers very unlikely. Since the majority of consumers interested in the solution of DEs now have access to high-speed general purpose computers, most will choose to purchase software in preference to extra hardware.

As suggested in the Introduction, DDAs are more likely to find application where a specific DE is to be solved or where a very large system is to be simulated. Both of these classes of problem call for maximum solution speed, which will not be available if DDA modules are implemented through microprogramming and interconnected by software to reduce costs and improve flexibility.
Digital Differential Analysers

2. BASIC DDA ARCHITECTURE AND ERROR ANALYSIS

When DDAs were first produced, the only means of modelling systems was the input-output approach, based on Laplace Transforms. Analog computers implemented this modelling method, employing integrators connected in series when the system DE contained higher (than first) derivatives. DDAs were produced by substituting equivalent digital devices for the various analog computer elements, hence early DDAs employed the manual patch board of the analog computer, and the DDA modules were generally interconnected serially. Mayorov and Chu [18] present diagrams of interconnection patterns for the generation of functions ranging from algebraic through to trigonometric and inverse trigonometric functions, and other functions of greater complexity.

In order to gain speed, the serial configuration of integrators in DDAs was abandoned in favour of a parallel configuration [21], for which the state space representation of DEs is particularly appropriate. This state space representation forms the basis of an alternative approach to system modelling, which uses a set of state variables to provide a model which permits examination of not only the input-output characteristics of the system, but of internal characteristics as well. Assuming that a given system can be represented by a DE of order n, the state space method will require definition of a set of n states to model the system, and will use a corresponding set of n independent first order DEs in place of the single, high order DE. Each first order DE defines the derivative of one state variable in terms of the n state variables, leading naturally to a convenient matrix representation of the system.
2.1 DDA modules

The basic functions performed by the digital modules in a DDA are integration (with respect to time or another variable), multiplication (by constants or variables) and summation (of multiple increments). Thus a DDA may typically be composed of three types of modules, to be interconnected in accordance with the requirements of the DE being solved.

For the sake of simplicity and clarity, it is most convenient to analyse a basic DDA, consisting of only one integrator and one multiplier, which employs Euler integration to solve the linear first order DE defined by

\[ \dot{y} = Ay, \quad y(0) = 1 \quad (2.1a) \]

using residue retention registers suitable for the suppression of second order truncation error. The results have been shown to be directly applicable to higher order integration formulae \([29, 44, 45]\) and to more complex DEs \([47]\).

Most commonly, the independent variable in a numerical integration problem is time, and the increment size can be referred to as \(\Delta t\). After \(i\) iterations, integration can be considered to have proceeded to a point \(t_i = i \times \Delta t\) on the time axis, where the exact numerical value of the solution to equation (2.1a) will be referred to as \(y_i\).

\(y_i\) is defined to be the numerical value which would be obtained if registers of infinite length were used. It will differ from the analytical solution because of discretisation error, but will not suffer from truncation error, hence is particularly relevant for analysis of the truncation error introduced into DDA solutions of DEs.
The estimate of the solution at \( t_i \) produced by Euler's method will be referred to as \( y_{e_i} \), to distinguish it from the exact (numerical) solution, \( y_i \). From this point, the task is to compute an approximation to \( \Delta y_i = y_{i+1} - y_i \), then add this to \( y_{e_i} \) to generate \( y_{e_{i+1}} \). Euler's method uses the slope (derivative) at \( t_i \) to produce an estimate of \( \Delta y_i \), relying on the relation \( \dot{y} = \frac{dy}{dt} = \frac{\Delta y}{\Delta t} \), that is, \( \Delta y_i = \dot{y}_i \Delta t \).

In order to generalise the following discussion, \( h \) will be used, instead of \( \Delta t \), to represent the increment size of the independent variable. Use of \( h \) avoids the implication that integration is with respect to time, in recognition of the ability of DDAs to perform integration where time is not the independent variable. An appropriate expression of Euler's method, when applied to solve equation (2.1a), is thus

\[
y_{e_{i+1}} = y_{e_i} + \Delta y_i \\
= y_{e_i} + A \times y_{e_i} \times h .
\]

(2.1b)

The DDA implementation of equation (2.1b) is a two phase process, requiring both integration and multiplication. If residue retention is implemented fully, that is, in both the integrator and the multiplier, separate residue registers are necessary, and these will be referred to as IR (for the Integrator Residue register) and MR (for the Multiplier Residue register). The state register is named YE (rather than the usual \( Y \)) to imply Euler integration, leading to the following set of register transfers:

\[
\begin{align*}
IR + YE & \rightarrow \Delta Z,IR \quad \text{(2.1c)} \\
MR + A \times \Delta Z \times h & \rightarrow \Delta YE,MR \quad \text{(2.1d)} \\
YE & \rightarrow \Delta YE + YE , \quad \text{(2.1e)}
\end{align*}
\]
Figure 2.1: ERDDA Integrator
which are executed repeatedly to generate a solution to equation (2.1b). The integration and multiplication DDA functions are present, in (2.1c) and (2.1d) respectively, but because of the simplicity of the problem (only one state) no summation is required.

Since DDAs were first produced, h values have been restricted to negative powers of 2 in order to simplify DDA hardware. This restriction allows multiplication by h to be achieved through a right shift of $A \times \Delta Z$ by an appropriate number of bits, which is emphasised in (2.1d) by the use of different symbols for the two multiplications: '·' implies multiplication by shifting, and '×' is used to indicate the need for a genuine multiplication of A by $\Delta Z$.

Another aspect of the register transfers requiring explanation is the use of ',' in (2.1c). A '+' has typically been used to separate increments from residues in the register transfers appearing in the literature, but since addition of these registers is never performed, the ',' notation is introduced to avoid potential confusion.

2.1.1 Integration

DDA integrators contain the state variable registers; hence their design allows the state values to be updated at each iteration (transfer (2.1e)), as well as performing the integration function (transfer (2.1c)). The conventional ERDDA integrator receives $\Delta YE$ as input, and produces as output $\Delta Z$: a truncated version of the updated YE register, as shown in figure 2.1.

The length, M bits, of the YE register is of fundamental importance in DDAs, as this parameter defines the accuracy of the machine. According to Baker [41], for the solution of a stable linear
Figure 2.2: ERDDA Multiplier
DE, the use of residue retention registers allows the accumulation of truncation error to be suppressed such that its value is less than \( \log_2(N+1) \) bits, where \( N \) is the number of states in the equation.

Of similar importance is the parameter \( k \), which is not only the length of the IR register as indicated in figure 2.1, but also defines both the integration step-size and the required interconnection path width between ERDDA modules (for a given value of \( M \)). The interconnection path needs to be \( m+1 \) bits wide, where \( m = M-k \), to correctly transmit the \( m \) most significant bits of the YE register. The extra bit (at the most significant end) of \( \Delta Z \) is required when the addition in (2.1c) produces overflow beyond \( M \) bits.

More will be said about the integration step-size, \( h \), in the following Section.

2.1.2 Multiplication

The multiplier receives as input the output of the integrator, \( \Delta Z \). In keeping with the philosophy of fixed point ERDDAs, the binary point in figure 2.1 is assumed to be to the immediate left of the YE register, so, given the manner in which \( \Delta Z \) values are generated, the maximum \( \Delta Z \) value possible is 1.0: if the top bit of \( \Delta Z \) is set due to arithmetic overflow all other bits will obviously be zero. This guarantees that the product, \( A \times \Delta Z \), generated in figure 2.2 will never exceed \( M+m \) bits in length, even though the lengths of the operands total \( M+m+1 \) bits.

The length chosen for the A register governs the length of the MR register, and, ignoring the trivial multiplication by \( h \) at this stage, the possibility of overflow during the addition of \( A \times \Delta Z \) to MR dictates that the \( \Delta YE \) register be \( m+1 \) bits long, with the same
potential range of values as \( \Delta Z \). The binary point in figure 2.2 is positioned to the immediate left of the A register, and, except that the multiplication by \( h \) in (2.1d) is not included, the relative horizontal displacements of the other registers are chosen to indicate the relative significance of their contents.

Referring back to figure 2.1, it is now clear that in order to add \( AYE \) to \( YE \) such that their least significant bits are aligned as shown, \( AYE \) must be shifted right by \( k \) bits. This right shift is equivalent to the multiplication by \( h \) required in (2.1d), indicating that the integration step-size, \( h \), is indeed defined by \( h = 2^{-k} \).

Applying this result to figure 2.2, the \( AYE \) and MR registers should actually be positioned \( k \) bits further to the right for correct alignment relative to the binary point.

2.1.3 Summation

The summation problem is best illustrated with a higher order DE, expressed in state space form, such as problem B2 from Hull, et al [48]:

\[
\begin{align*}
\dot{y}_1 &= -y_1 + y_2, & y_1(0) &= 2 \\
\dot{y}_2 &= y_1 + 2y_2 + y_3, & y_2(0) &= 0 \\
\dot{y}_3 &= y_2 - y_3, & y_3(0) &= 1.
\end{align*}
\]

DDA solution of this problem would require three integrators: one for each state. Assuming that Euler integration is used, the following three transfers would be required:

\[
\begin{align*}
IR1 + YE1 & \rightarrow \Delta Z1, IR1 \\
IR2 + YE2 & \rightarrow \Delta Z2, IR2 \\
IR3 + YE3 & \rightarrow \Delta Z3, IR3
\end{align*}
\]

to perform the integration phase. No true multiplication would be
needed, as multiplication by 2 is a simple left shift, but in this problem summing elements are required, to generate \( \Delta YE \) values from the \( \Delta Z \) values. The appropriate register transfers are:

\[
-\Delta Z_1 + \Delta Z_2 \rightarrow \Delta YE_1 \\
\Delta Z_1 + 2 \cdot \Delta Z_2 + \Delta Z_3 \rightarrow \Delta YE_2 \\
\Delta Z_2 - \Delta Z_3 \rightarrow \Delta YE_3
\]

followed by three trivial transfers to update the YE registers. It should be noted that \( \Delta YE \) values generated by the summation of a number of \( m+1 \)-bit incremental values can easily exceed \( m+1 \) bits in length.

2.1.4 Forcing Functions

When DEs contain forcing functions, function generators may be required. Alternatively, in some cases the forcing function can be absorbed into the state space matrix by introducing more states. For example, the forcing function in Problem A3 from [48]:

\[
y = y \cos(x), \quad y(0) = 1
\]

can be generated using two extra states (a sine/cosine pair):

\[
\dot{y}_1 = -y_2, \quad y_1(0) = 1 \quad \{y_1 = \cos(x)\} \\
\dot{y}_2 = y_1, \quad y_2(0) = 0 \quad \{y_2 = \sin(x)\} \\
\dot{y}_3 = y_3 y_1, \quad y_3(0) = 1.
\]

2.2 Truncation Error Analysis

The truncation error analysis method of Baker and McCrea mentioned in the previous Chapter (see [38,39,40,41]) will now be demonstrated, using equation (2.1a) as an example. The analysis will show that the register transfers given above (in (2.1c-e)) allow suppression of the accumulation of truncation error to within the
least significant bit of the YE register, for a linear scalar stable DE.

To avoid confusion between the contents of registers and their names, upper case letters are used for register names, with the corresponding lower case letters reserved for register contents. Since register contents will generally change at each iteration, subscripts will also be attached to register contents to indicate the number of iterations performed.

2.2.1 Preliminary Definitions

For the purposes of truncation error analysis, discretisation error will be ignored by treating \( y_i \) as the true solution. Recalling the above definition of \( y_i \) as the solution which would be generated by \( i \) steps of numerical integration if registers of infinite length were used, treating this as the true solution provides ready access to the truncation error, which is simply the difference between \( y_i \) and \( ye_i \). Truncation error arises solely from the fact that real DDA registers are finite in length.

Given that YE is M bits long, and the binary point appears at the immediate left of the YE register, the truncation error, \( E_i \), in \( ye_i \) can be defined by

\[
y_i - ye_i = E_i 2^{-M}.
\]

(2.2.1a)

\( E_i \) then represents the contents of an imaginary infinitely long register attached to the less significant end of the YE register.

A similar definition of the truncation error in \( \Delta ye_{i-1} \) is suggested by the fact that the \( \Delta YE \) register is of limited length, with its least significant end M bits to the right of the binary point (as indicated in figure 2.1). Defining \( \Delta y_i \) to be the value calculated if
Chapter 2 Basic DDA Architecture and Error Analysis

registers of infinite length were used, $\Delta E_i$ is then defined by

$$\Delta y_i - \Delta y e_i = \Delta E_i 2^{-M}, \quad (2.2.1b)$$

where $\Delta E_i$ can be thought of as the contents of an infinitely long
register attached to the lower end of the $\Delta YE$ register.

Combining these definitions of $E_i$ and $\Delta E_i$ with equation
(2.1b) leads to an equation for the truncation error after one more
step of integration:

$$y e_{i+1} = y e_i + \Delta y e_i$$

$$= y_i - E_i 2^{-M} + \Delta y_i - \Delta E_i 2^{-M}$$

$$= y_i + \Delta y i 2^{-k} - (E_i + \Delta E_i) 2^{-M}$$

$$= y_{i+1} - (E_i + \Delta E_i) 2^{-M}$$

$$= y_{i+1} - E_i 2^{-M} \quad \{ \text{from (2.2.1a)} \}$$

i.e., $E_{i+1} = E_i + \Delta E_i$, \quad (2.2.1c)

which is actually a somewhat obvious result.

Two other useful definitions to have for the analysis are
definitions of the change in residue register contents from one
iteration to the next:

$$\Delta r_i = r_i - r_{i-1} \quad (2.2.1d)$$

$$\Delta r_{i} = m r_i - m r_{i-1} \quad (2.2.1e)$$

2.2.2 Analysis

Derivation of an equation for the error in $y e_{i+1}$ (compared
to $y_{i+1}$) proceeds as follows.

$$\Delta z_i = y e_i - \Delta r_i 2^{-m} \quad \{ \text{from (2.1c) and (2.2.1d)} \}$$

$$= y_i - E_i 2^{-M} - \Delta r_i 2^{-m} \quad \{ \text{from (2.2.1a)} \}$$
\[ \Delta y_i = A \Delta z_i 2^{-k} - \Delta m r_i 2^{-M} \]
\[ \quad \text{from (2.1d) and (2.2.1e)} \]
\[ = A(y_i - E_i 2^{-M} - \Delta i r_i 2^{-m})2^{-k} - \Delta m r_i 2^{-M} \]
\[ = A(y_i - E_i 2^{-M})2^{-k} - A \Delta i r_i 2^{-M} - \Delta m r_i 2^{-M} \]
\[ = A \Delta y_i 2^{-k} - A \times E_i 2^{-M+k} - (A \Delta i r_i + \Delta m r_i)2^{-M} \]
\[ = \Delta y_i - \Delta E_i 2^{-M} \]
\[ \text{from (2.2.1b)} \]

therefore

\[ \Delta E_i = A \times E_i 2^{-k} + A \Delta i r_i + \Delta m r_i \]
\[ = E_{i+1} - E_i \]
\[ \text{from (2.2.1c)} \]

which leads to

\[ E_{i+1} = E_i + A \times E_i 2^{-k} + A \Delta i r_i + \Delta m r_i \]
\[ = (1 + A \times 2^{-k})E_i + A \Delta i r_i + \Delta m r_i . \]

Now let

\[ X = 1 + A \times 2^{-k} \]
\[ \text{(2.2.2a)} \]
\[ R_i = A \Delta i r_i + \Delta m r_i \]
\[ \text{(2.2.2b)} \]

to yield a simplified recursive equation for \( E_{i+1} \):

\[ E_{i+1} = X \times E_i + R_i . \]

Repeated substitution of this equation into itself leads to another form:

\[ E_{i+1} = X(X \times E_i - 1 + R_{i-1}) + R_i \]
\[ = X^2 E_i - 1 + X \times R_{i-1} + R_i \]
\[ = X^2 (X \times E_i - 2 + R_{i-2}) + X \times R_{i-1} + R_i \]
\[ = X^3 E_i - 2 + X^2 R_{i-2} + X \times R_{i-1} + R_i \]
\[ = \ldots \]
\[ = X^{i+1} E_0 + \sum_{j=0}^{i} (X^{i-j} R_j) . \]

This can be reorganised by recognising that

\[ X^n = 1 + A \times n \times 2^{-k} + \text{higher order terms (HOTs)} , \]
\[ \text{from (2.2.2a)} \]

to give

- 32 -

Section 2.2.2
The first line of equation (2.2.2c) represents the second order truncation error, with the third order truncation error shown on the second line.

2.2.3 Suppression of Second Order Truncation Error

As initially stated, the register transfers given above, which led to equations (2.2.1d-e) and (2.2.2b), are suitable for the suppression of second order truncation error. The magnitude of the second order truncation error can thus be determined by first substituting equation (2.2.2b) into equation (2.2.2c), then substituting in equation (2.2.1d-e):

\[
E_{i+1} = E_0 + \sum_{j=0}^{i} (A\Delta r_j + \Delta m r_j) + A((i+1)E_0 + \sum_{j=0}^{i-1} (A\Delta r_n + \Delta m r_n))2^{-k} + \text{HOTs}
\]

Interpreting residue register contents as though the binary point appears at their left end, the contents of each register will lie in the range 0 to 1, so it can be seen that initialising all
residue registers to \( \frac{1}{2} \) will most effectively suppress the accumulation of the terms on the first line, representing the second order truncation error. Such initialisation ensures that

\[
E_0 - \frac{1}{2}(A + 1) \leq E_{i+1} \leq E_0 + \frac{1}{2}(A + 1),
\]

i.e., \( |E_{i+1}| \leq E_0 + \frac{1}{2}(A + 1) \) (2.2.3b)

To avoid possible misunderstanding, it is stressed that no modification of standard DDA residue registers is necessary to achieve this result. Equations (2.2.1d-e) represent the residue register addition process characteristic of the DDA, and this leads naturally to the suppression of second order truncation error accumulation as shown above. In the following Section, it is shown that minor modification of DDA register organisation allows the suppression of third order truncation error accumulation as well.

2.2.4 Suppression of Third Order Error

It may be readily observed that, as \( i \) increases, the magnitude of the third order truncation error in equation (2.2.3a) (represented by the second line) will grow. Hence the register transfers of (2.1c-e) do not achieve third order truncation error suppression.

Suppression of third order truncation error is effected by modification of the register transfers which lead to the definitions (equation (2.2.1d-e)) of \( \Delta r_i \) and \( \Delta m_i \). Rather than simply adding a residue in at the next iteration, each residue is added twice at the next iteration, then subtracted at the following iteration, giving the same net result. To implement this strategy, it is not sufficient to retain only the residues from the last iteration: \( r_{i-1} \) and \( m_{i-1} \). Extra storage must be provided for the retention of the residues from the previous iteration as well: \( r_{i-2} \) and \( m_{i-2} \). At each iteration,
ir_{i-1} and mr_{i-1} can then be doubled before the addition of ye_i, and
ir_{i-2} and mr_{i-2} subtracted from the result, leading to suppression of
third order truncation error. Equation (2.2.1d-e) needs to be
replaced by:

\[ \Delta ir_i = ir_i - 2 \times ir_{i-1} + ir_{i-2} \]
\[ \Delta mr_i = mr_i - 2 \times mr_{i-1} + mr_{i-2}. \]

Substituting these definitions into equation (2.2.2c) yields an error
equation which indicates that suppression of third order truncation
error is possible.

\[ E_{i+1} = E_0 + \sum_{j=0}^{i} \left( A \Delta ir_j + A \Delta mr_j \right) \]
\[ + A((i+1)E_0 + \sum_{j=0}^{i-1} \left( \sum_{n=0}^{j} (A \Delta ir_n + A \Delta mr_n) \right) )2^{-k} \]
\[ + \text{HOTS} \]

\[ = E_0 + \sum_{j=0}^{i} \left( A(ir_j - 2 \times ir_{j-1} + ir_{j-2}) + mr_j - 2 \times mr_{j-1} + mr_{j-2} \right) \]
\[ + A((i+1)E_0 + \sum_{j=0}^{i-1} \left( \sum_{n=0}^{j} (A(ir_n - 2 \times ir_{n-1} + ir_{n-2}) + mr_n - 2 \times mr_{n-1} + mr_{n-2}) \right) )2^{-k} \]
\[ + \text{HOTS} \]

\[ = E_0 + A(ir_i - ir_{i-1} + ir_{i-1}) + mr_i - mr_{i-1} + mr_{i-1} \]
\[ + A((i+1)E_0 + \sum_{j=0}^{i-1} \left( A(ir_j - ir_{j-1} + ir_{j-1}) + mr_j - mr_{j-1} + mr_{j-1} \right) )2^{-k} \]
\[ + \text{HOTS} \]

\[ = E_0 + A(ir_i - ir_{i-1} + ir_{i-1}) + mr_i - mr_{i-1} - (A \times ir_{-1} + mr_{-1}) \]
\[ + A((i+1)E_0 + \sum_{j=0}^{i-1} \left( A(ir_j - ir_{j-1}) + mr_j - mr_{j-1} - (A \times ir_{-1} + mr_{-1}) \right) )2^{-k} \]
\[ + \text{HOTS} \]
Clearly, the third order truncation error (represented by the second line) can be minimised by initialising all residue registers to 0. The effect of this on the suppression of second order error is adverse but not disastrous: allowing for worst cases, the range of values now possible for the second order term is

\[ E_0 - (A+1) \leq E_{i+1} \leq E_0 + (A+1) \]

i.e., \( |E_{i+1}| \leq E_0 + (A+1) \)

which is roughly twice the magnitude of equation (2.2.3b).

### 2.2.5 Further Error Analysis

This method of truncation error analysis is employed in Appendix C to verify the correctness of register transfers for a second order, predictor-corrector integration algorithm, and is applied in Chapter 6 to a nonlinear DE. The analysis results in Chapter 6 suggest that residue register suppression of truncation error accumulation with nonlinear DEs is not quite as successful as for linear DEs.

### 2.3 Second Order Differences

Second Order Difference transmission ([36]) permits two improvements to be made to ERDDA design. Firstly, the interconnection problems associated with multibit increment transmission can be reduced, provided that an appropriate step-size is chosen, and
Figure 2.3: Second Order Difference Integrator
secondly, the precision of the multiplication made necessary by multibit increments can be reduced. Second Order Difference transmission is transmission of the difference between successive integrator (or multiplier) outputs.

2.3.1 Integration

A Second Order Difference integrator is designed to produce the difference, \( \Delta^2Z \), between successive output increments, defined by

\[
\Delta^2z_i = \Delta z_i - \Delta z_{i-1}.
\]

(2.3.1a)

The input to the integrator should be a similar Second Order Difference, \( \Delta^2YE \), between successive input increments, defined by

\[
\Delta^2ye_i = \Delta ye_i - \Delta ye_{i-1}.
\]

\( \Delta ye_i \) is generated from \( \Delta^2ye_i \) and \( \Delta ye_{i-1} \) within the integrator, then the YE register is updated as usual.

To implement these features, the basic ERDDA integrator of figure 2.1 needs to be modified to the design shown in figure 2.3. When \( \Delta^2ye_i \) arrives at the input, the \( \Delta YE \) register will contain \( \Delta ye_{i-1} \) (from the previous iteration), to which \( \Delta^2ye_i \) is added to produce \( \Delta ye_i \). This result is used to update the YE register, and is also stored in the \( \Delta YE \) register ready for the next iteration. At the output, when \( \Delta z_i \) is produced by the addition of \( ye_i \) to \( r_{i-1} \), the \( \Delta Z \) register will still contain \( \Delta z_{i-1} \) from the previous iteration. The subtraction of \( \Delta z_{i-1} \) from \( \Delta z_i \) yields \( \Delta^2z_i \), and \( \Delta z_i \) is loaded into the \( \Delta Z \) register in preparation for the next iteration.

2.3.2 Step-size Restriction and Starting Problem

As has been stated, the success of Second Order Difference transmission in reducing the interconnection bit count is dependent on
appropriate choice of step-size. It will now be shown that $\Delta^2 Z$ values can be guaranteed never to exceed three bits in length (except at the first step) if step-size is chosen according to the restriction

$$m + 1 \leq k.$$  \hspace{1cm} (2.3.2a)

Consider figure 2.3, in which the binary point is assumed to be immediately to the left of the YE register. $\Delta z_{i-1}$ is generated by adding $ye_{i-1}$ to $ir_{i-2}$ and truncating the lower $k$ bits, that is,

$$\Delta z_{i-1} = ye_{i-1} + ir_{i-2}2^{-m} - \text{(last } k \text{ bits)}.$$  

Given the relationship between the $\Delta Z$ and IR registers in figure 2.3, the contribution of $ir_{i-2}$ to $\Delta z_{i-1}$ will obviously be a matter of arithmetic overflow into the low end of the $\Delta Z$ register, hence will be $\in \{0, 1\}$. To generate $\Delta z_i$, $\Delta ye_{i-1}$ is added to $ye_{i-1}$ to give $ye_i$, which is then added to $ir_{i-1}$ and the lower $k$ bits truncated:

$$\Delta z_i = ye_{i-1} + \Delta ye_{i-1}2^{-k} + ir_{i-1}2^{-m} - \text{(last } k \text{ bits)}.$$  

The contribution of $ir_{i-1}$ to $\Delta z_i$ will also be $\in \{0, 1\}$, depending on arithmetic overflow, but the contribution of $\Delta ye_{i-1}$ may be much greater. If the $\Delta YE$ register is shorter than the IR register, then the effect of $\Delta ye_{i-1}$ on $\Delta z_i$ can only be due to arithmetic overflow into the top $m$ bits of $ye_i$, and will be $\in \{-1, 0, +1\}$, but if the $\Delta YE$ register is longer than the IR register, as is the case in figure 2.3, then the most significant bits of $\Delta ye_{i-1}$ will be added directly into the least significant bits of $\Delta z_i$, making a much larger contribution.

Given the definition of $\Delta^2 z_i$ in equation (2.3.1a), it is now apparent that if equation (2.3.2a) is satisfied, the $\Delta YE$ register length will not exceed that of the IR register, and the combined contributions of $ir_{i-2}$, $ir_{i-1}$ and $\Delta ye_{i-1}$ to $\Delta z_{i-1}$ and $\Delta z_i$ will make $\Delta^2 z_i \in \{-2, -1, 0, 1, 2\}$: which has a maximum length of three bits.
Figure 2.4: Second Order Difference Constant Multiplier
Chapter 2  Basic DDA Architecture and Error Analysis

The starting problem of Second Order Difference transmission, which has been mentioned above, is that $\Delta z_{-1}$ does not exist, making $\Delta^2 z_0 = \Delta z_0$: up to $m+1$ bits in length. So, although Second Order Differences will generally only require a 3-bit interconnection path, the starting problem dictates that $m+1$-bit wide interconnections must be made.

Because the introduction of variable step-size integration to DDAs presents a solution to the Second Order Difference starting problem, the following example is included to indicate the extent of the restriction of step-size necessary to guarantee that 3-bit interconnections are sufficient after the first step.

Consider a DDA with 16-bit word-length, that is, $M = m+k = 16$. Applying equation (2.3.2a), step-size must be restricted according to $9 \leq k \leq 16$, allowing $m$ values in the range $7 \leq m \leq 0$. Simulation by Bywater [36] indicates that the restriction does not need to be so strict, and $8 \leq k \leq 16$ is reasonable, allowing a larger step-size and $8 \leq m \leq 0$.

In general, the appropriate restriction on $k$ is $\frac{1}{2}M \leq k \leq M$, which will allow $\frac{1}{2}M \leq m \leq 0$. With this restriction, and defining $L(X)$ to be the significant bit length of $X$,

$L(\Delta^2 z_i) \leq 3$, for $i > 0$, and
$L(\Delta^2 z_0) \leq m + 1$.  

(2.3.2b)  
(2.3.2c)

2.3.3 Multiplication by Constants

The Second Order Difference multiplier will receive $\Delta^2 Z$ as input, and is required to generate $\Delta^2 YE$ as output. As can be seen from figure 2.4, the modified output section is very similar to that of the Second Order Difference integrator (figure 2.3), but the input
section of the multiplier does not contain the expected $\Delta Z$ register. Instead, an adder is connected to the product register, $A \times \Delta Z$, allowing the product to be updated (rather than recalculated) at each iteration, with $A$ multiplied by $\Delta^2 Z$ instead of by $\Delta Z$.

When $\Delta^2 z_{i}$ arrives at the input, the product register will contain $A \times \Delta z_{i-1}$ (from the previous iteration) and $A \times \Delta z_{i}$ can be calculated using the definition of Second Order Differences:

$$A \times \Delta z_{i} = A \times (\Delta z_{i-1} + \Delta^2 z_{i}) \quad \text{[from (2.3.1a)]}$$

$$= A \times \Delta z_{i-1} + A \times \Delta^2 z_{i}.$$  

The advantage of multiplying $A$ by $\Delta^2 z_{i}$, rather than by $\Delta z_{i}$ as in figure 2.2, is that $\Delta^2 z_{i}$ is only three bits long, compared to $m+1$ bits for $\Delta z_{i}$. Multiplication by constants is consequently greatly simplified by Second Order Difference transmission, which allows the multiplication operation to be implemented using only an adder, since $\Delta^2 z_{i} \in \{-2,-1,0,+1,+2\}$.

### 2.3.4 Multiplication by Variables

When equations contain multiplicative nonlinearities, the constant in the multiplier must be replaced by the appropriate state variable, plus the associated hardware to update it. As has been shown by Baker [49], this leads to complications because the task of computing $ye_{i} \times \Delta z_{i}$ from a stored previous product, $ye_{i-1} \times \Delta z_{i-1}$, involves two increments:

$$ye_{i} \times \Delta z_{i} = (ye_{i-1} + \Delta ye_{i-1})(\Delta z_{i-1} + \Delta^2 z_{i})$$

$$= ye_{i-1} \times \Delta z_{i-1} + ye_{i-1} \times \Delta^2 z_{i} + \Delta ye_{i-1} \times \Delta z_{i-1} + \Delta ye_{i-1} \times \Delta^2 z_{i}$$

The two multiplications by $\Delta^2 z_{i}$ may be implemented simply by addition as shown above, but the product $\Delta ye_{i-1} \times \Delta z_{i-1}$ is not so easily generated. In order to avoid $(m+1) \times (m+1)$-bit multiplication,
Figure 2.5: Second Order Difference Variable Multiplier
\[ \Delta y_{i-1} \times \Delta z_{i-1} \text{ could also be stored, and updated ready for the next iteration:} \]

\[ \Delta y_i \times \Delta z_i = (\Delta y_{i-1} + \Delta^2 y_i)(\Delta z_{i-1} + \Delta^2 z_i) \]

\[ = \Delta y_{i-1} \times \Delta z_{i-1} + \Delta y_{i-1} \times \Delta^2 z_i + \Delta^2 y_i \times \Delta z_{i-1} + \Delta^2 y_i \times \Delta^2 z_i \]

The additional two multiplications by \( \Delta^2 z_i \) pose no problem (except that now four are necessary), but the remaining multiplication by \( \Delta^2 y_i \) is not necessarily so convenient. In large systems, \( \Delta^2 yE \) will generally be greater than 2, due to the summation of increments discussed above, and more than a simple addition will be necessary.

The apparent hardware complexity required to avoid actual multiplications suggests that, in a Second Order Difference variable multiplier, it may well be desirable to update \( \Delta z_{i-1} \) by \( \Delta^2 z_i \) then perform an \((m+1)\times M\)-bit multiplication directly, as shown in figure 2.5.
3. INTRODUCING VARIABLE STEP-SIZE TO DDAS

As stated in the Introduction, the most likely contemporary use of DDAs will be in dedicated applications, and, for maximum solution speed, a DDA should be constructed specifically to solve the DE involved. In this context, it is likely that the required pattern of step-size change will be known in advance, making external control of step-size feasible. Hence it is appropriate to treat as two separate tasks the development of DDAs capable of VSS integration:

1. modifying Second Order Difference architecture to allow VSS integration to be performed, irrespective of the manner in which step-size change is to be controlled;
2. investigating the feasibility of autonomous step-size control, that is, the variation of step-size on the basis of numerical values stored within the DDA registers.

This Chapter is devoted to the first of these two tasks, and to analysis of the effect of VSS integration on the suppression of truncation error accumulation.

The other major innovations in DDA design, such as Extended Resolution, Second Order Differences, and Global Sliding Point, have all been introduced at the expense of some of the simplicity of original DDA designs. The introduction of VSS to DDAs will inevitably have a similar cost, but the magnitude of the increase in design complexity can be limited by making some restrictions on the potential patterns of step-size change.

The most obvious restriction is to continue to limit potential step-size values to negative powers of 2, so that multiplication by \( h \) remains as simple as possible. This implies that the minimum possible
change in step-size will be by a factor of 2. It will be shown in
Appendix D that attempts to control step-size more finely not only
make multiplication by h less convenient, but also lead to excessive
complexity in the DDA modules themselves if Second Order Difference
transmission is implemented. Hence ERDDA step-size should continue to
be determined by the value of k according to the relation $h = 2^{-k}$,
making variation of step-size equivalent to variation of k. Recalling
the relationship between M, m and k given in Chapter 2: $M = m+k$,
there are two options open for implementing VSS:

1. maintain m constant and allow M to vary as k varies;
2. maintain M constant and allow m to change.

It should be evident that the second alternative is preferable,
because M defines the accuracy of the machine and should remain
constant. In addition, choice of the second option leads to a simple
solution to the starting problem of Second Order Differences, as will
be seen.

The potential increase in complexity can also be limited by
making the further restriction that k (and hence m) can change at most
by plus or minus unity at any step of the solution. This implies that
the step-size can only be halved or doubled if changed. Such a
restriction is considered reasonable for the solution of non-stiff
ordinary DEs, because a rate of change of step-size in excess of
halving/doubling at each step will rarely (if ever) be required.

3.1 Effect of VSS on DDA Architecture

Having chosen to allow the value of m to change with step-
size, DDA modules and their interconnecting paths will have to be
modified to cater for increments which can vary in length. If direct transmission is used, the interconnection paths have to cope with the maximum value that \( m+1 \) can reach: theoretically \( M+1 \) bits, which implies significant magnification of the interconnection problem associated with extended resolution. The 3-bit transmission paths available with Second Order Differences (after the first step) are clearly preferable, and will be shown to remain adequate with the introduction of VSS integration.

3.1.1 Effect of VSS on Second Order Differences

Changes in step-size do have a significant impact on Second Order Differences. If step-size is doubled after \( i \) iterations, \( m \) is increased by one, making \( \Delta z_i \) one bit longer than \( \Delta z_{i-1} \) (since \( \Delta Z \) values are \( m+1 \) bits in length), hence

\[
\Delta z_i = 2 \times \Delta z_{i-1}.
\]

Similarly, when step-size is halved, \( m \) is decreased by one, making

\[
\Delta z_i = \frac{1}{2} \times \Delta z_{i-1}.
\]

In either case, \( \Delta^2 z_i \) values calculated using equation (2.3.1a) would certainly exceed three bits in length.

To avoid the potential increase in interconnection requirements, the definition of \( \Delta^2 z_i \) must be generalised to

\[
\Delta^2 z_i = \Delta z_i - \Delta z_{i-1} \times \frac{h_{i+1}}{h_i}, \tag{3.1.1a}
\]

which guarantees that \( L(\Delta^2 z_i) \leq 3 \), even when the step-size is changed.

The way to overcome the starting problem of Second Order Differences is now clear: simply force the solution to start with a step-size of \( k \geq M-2 \), which makes \( m+1 \leq 3 \) at the first step, hence equations (2.3.2b-c) can be replaced by

- 47 - Section 3.1.1
Figure 3.1: VSSDDA Integrator
for all steps, including the first.

3.1.2 VSS Integrator Design

A number of changes must be made to the basic integrator design to accommodate VSS. Firstly, the lengths of the $\Delta Z$ and IR registers must be sufficient to contain the maximum possible increments and residues respectively, hence IR must be $M$ bits long, and $\Delta Z \frac{1}{2} M + 1$ bits long. Secondly, logic must be provided to correctly split $YE + IR$ into new $\Delta Z$ and IR values according to the current value of $k$, and load these values into the corresponding registers with right justification. Thirdly, the stored previous values of $\Delta YE$ and $\Delta Z$ required for Second Order Difference transmission must be appropriately shifted if step-size is changed (in accordance with equation (3.1.1a)).

These modifications lead to the block diagram of a VSS integrator shown in figure 3.1. The extra hardware consists of three combinational logic networks (CLNs) which perform simple shifting and addition operations. "sscc" represents a step-size-change-control signal, the generation of which is crucial for VSS integration, and will be described in Chapter 5. The value of sscc at any time will lead to one of the three following actions:

1. no change;
2. double step-size;
3. halve step-size.

CLN1 is required primarily because the boundary between IR and $\Delta Z$ varies, and $\Delta Z_1$ and ir_1 must be correctly separated, then loaded into the $\Delta Z$ and IR registers with right-justification. CLN2 implements...
Figure 3.2: VSSDDA Multiplier
equation (3.1.1a) via a shifter (which will either transmit $\Delta z_1$, shift one bit left, or shift one bit right depending on $sscc$) and an adder to form the correct $\Delta^2 Z$, and CLN3 performs similar operations to form the correct $\Delta Y E$ from $\Delta^2 Y E$.

3.1.3 VSS Multiplier Design

Because the length of the MR register is always $M$ (regardless of $k$), knowledge of the current value of $k$ is not necessary in the multiplier. It is sufficient that the register for storing the product $A \times \Delta Z$ (or $Y E \times \Delta Z$) be extended to cope with the maximum possible product length, $\frac{3}{2}M$ bits. Partitioning of this product will naturally lead to right justification as required.

One consequence of equation (3.1.1a), of importance particularly to constant multipliers, is that when step-size is halved, stored previous increments are all shifted right by one bit. If the least significant bit of $\Delta z_1$ is set, this shift will produce truncation, which implies that the incremental multiplication employed in figure 2.4 cannot be relied on. Specifically,

$$A(\Delta z_{1-1} \text{ div } 2 + \Delta^2 z_1) = (A \times \Delta z_{1-1}) \text{ div } 2 + A \times \Delta^2 z_1$$

is not always correct, so for constant multiplication a strategy similar to that required for variable multiplication is desirable.

This suggests a general multiplier design catering for multiplication by either variables or constants, as shown in figure 3.2. If multiplication by constants is performed, the incoming increments to the "AorYE" register will be 0.
Chapter 3

3.2 Error Suppression with Variable Step-Size

When step-size changes, L(IR) will change also, since both quantities are determined by k. This variation in residue length introduces some uncertainty into the question of exactly how YE and IR should be combined immediately after a change in step-size, because two apparently reasonable alternatives exist:

1. Align the most significant bit of \( \text{iri}_{i-1} \) with that of \( \text{iri}_i \), that is, add in the old residue at the same significance with which the next residue will be extracted. To achieve this, \( \text{iri}_{i-1} \) would be shifted one bit to the left (if step-size was halved) or to the right (for doubled step-size);

2. Align the least significant bit of \( \text{iri}_{i-1} \) with that of \( \text{iri}_i \), that is, add in the old residue at the same significance with which it was extracted. No residue shifting is necessary to implement this strategy.

Truncation error analysis will be used to determine the more accurate of the two alternatives, but slight modification of the definition of the change in the IR register contents is necessary. Conveniently, the modification provides a simple way to distinguish between the two alternatives.

Making the assumption that integration is proceeding with constant step-size, and that \( k = m = \frac{1}{2}M \), there is only one difference between the constant and variable step-size DDA designs which is significant for truncation error analysis. This difference is in the length of the IR register, as may be seen by comparison of figures 2.3 and 3.1. In figure 3.1, the most significant \( m \) bits of IR will never be used if step-size is constant, that is, IR will always contain \( m \)
leading zeros.

The significance of this is apparent in equation (2.2.3a), where the IR register contents are referred to as though the binary point appears immediately to the left of the IR register of figure 2.3. If this register had extra bits attached to its most significant end, as is the case in figure 3.1, and the binary point assumed to be at the newly extended left end, then the IR register contents would be interpreted with significance reduced by $2^{-m}$, and equation (2.2.3a) as it stands would have a different meaning. To produce an equivalent error equation based on figure 3.1, the $m$ leading zeros in the IR register would need to be accounted for by multiplying all references to the IR register contents by $2^m$.

Modification of the truncation error analysis thus begins with a new definition of the change in the contents of the IR register, equation (2.2.1d) being replaced by

$$\Delta r_i = r_i 2^m - r_{i-1} 2^m$$

(3.2a)

when step-size is constant. This permits a distinction to be made between the above two alternative approaches to residue retention when step-size (and hence $m$) changes: the first alternative specifies that the current residue ($r_{i-1}$) should be shifted to conform to the new value of $m$, whereas the second alternative is that the old $m$ value should be applied to the current residue, and the new value to subsequent residues. Attaching subscripts to $m$ in recognition that its value can change, the equivalent modified definitions of $\Delta r_i$ for VSS integration with second order truncation error suppression would be

$$\Delta r_i = r_i 2^{m_i+1} - r_{i-1} 2^{m_i+1}$$

(3.2b)

for the first alternative, and
The second alternative turns out to be correct, permitting the accumulation of both second and third order truncation error to be suppressed as shown in the following two Sections, but, as is indicated in Section 3.2.3, the first alternative is not so successful.

### 3.2.1 Second Order Suppression

To examine the effect of a change in step-size on truncation error suppression, it is assumed that integration starts with step-size $2^{-k}$ and increment size = m, and proceeds with constant step-size for the first $i+1$ steps, after which step-size changes. Equation (2.2.2c) will still hold, except that the definition of $R_i$ for second order truncation error suppression is now (from equation (3.2a))

$$R_i = A(i_1^2 - i_{i-1}^2) + m_{i} - m_{i-1}$$

due to the m extra bits on the most significant end of the IR register. The truncation error after $i+1$ steps is thus

$$E_{i+1} = E_0 + \sum_{j=1}^{i} R_j$$

$$+ A((i+1)E_0 + \sum_{j=0}^{i-1} (\sum R_n))2^{-k}$$

$$+ \text{HOTS}$$

$$= E_0 + A(i_1^2 - i_{i-1}^2)2^m + m_{i} - m_{i-1}$$

$$+ A((i+1)E_0 + \sum_{j=0}^{i-1} (A(i_1^2 - i_{i-1}^2)2^m + m_{j} - m_{j-1})2^{-k}$$

$$+ \text{HOTS}$$

which is the same as equation (2.2.3a) provided that the m leading zeros in the IR register are taken into consideration. For the next
step, the step-size changes (is halved or doubled) to a new value, $2^{-x}$, with an associated new value of $m$ ($m_{1+2}$) which will be referred to as $n$. This leads to

$$R_{i+1} = A(ir_{i+1}2^n - ir_i2^m) + mr_{i+1} - mr_i$$

from equation (3.2c), hence the error after one more step will be given by

$$E_{i+2} = E_{i+1} + AxE_{i+1}2^{-x} + R_{i+1}$$

$$= E_o + A(ir_i - ir_{-1})2^m + mr_i - mr_{-1}$$
$$+ A((i+1)E_o + \sum_{j=0}^{i-1} (A(ir_j - ir_{-1})2^m + mr_j - mr_{-1}))2^{-k}$$
$$+ A(E_o + A(ir_i - ir_{-1})2^m + mr_i - mr_{-1})2^{-x}$$
$$+ A(ir_i+12^n - ir_i2^m) + mr_{i+1} - mr_i$$
$$+ HOTs$$

$$= E_o + A(ir_i+12^n - ir_{-1}2^m) + mr_{i+1} - mr_{-1}$$
$$+ A((i+1)E_o + \sum_{j=0}^{i-1} (A(ir_j - ir_{-1})2^m + mr_j - mr_{-1}))2^{-k}$$
$$+ A(E_o + A(ir_i - ir_{-1})2^m + mr_i - mr_{-1})2^{-x}$$
$$+ HOTs$$

The first line represents the second order truncation error, which can be seen to be properly suppressed.

3.2.2 Third Order Suppression

When register transfers modified for the suppression of third order truncation error are employed, further modification of the definition of $R_i$ is required:

$$R_i = A(ir_i^{m_{i+1}} - 2xir_{i-1}^{m_1} + ir_{i-2}^{m_{i-1}}) + mr_i - 2xmr_{i-1} + mr_{i-2}$$
Assuming that step-size is constant at $2^{-k}$ for $i+1$ steps, the error will be

\[
E_{i+1} = E_0 + \sum_{j=0}^{i} R_j + A[(i+1)E_0 + \sum_{j=0}^{i-1} (\sum_{n=0}^{j} R_n)]2^{-k} + \text{HOTS}
\]

\[
= E_0 + A(i r_i - ir_{i-1})2^m + mr_i - mr_{i-1} + A[(i+1)E_0 + A(ir_{i-1} - ir_{i-1})2^m + mr_{i-1} - mr_{i-1}]2^{-k} + A(E_0 + A(ir_{i-1} - ir_{i-1})2^m + mr_i - mr_{i-1})2^{-x} + A(ir_{i+1}2^m - ir_i2^m) + mr_{i+1} - mr_i + \text{HOTS}
\]

If the step-size then changes (is halved or doubled) to a new value, $2^{-x}$, and the associated new value of $m$ is $n$, the error after one more step (a total of $i+2$ steps) will be given by

\[
E_{i+2} = E_{i+1} + AxE_{i+1}2^{-x} + R_{i+1}
\]

\[
= E_0 + A(ir_i - ir_{i-1})2^m + mr_i - mr_{i-1} + A[(i+1)E_0 + A(ir_{i-1} - ir_{i-1})2^m + mr_{i-1} - mr_{i-1}]2^{-k} + A(E_0 + A(ir_{i-1} - ir_{i-1})2^m + mr_i - mr_{i-1})2^{-x} + A(ir_{i+1}2^n - ir_i2^m) + mr_{i+1} - mr_i + \text{HOTS}
\]

then, if another step is taken with the same step-size, $2^{-x}$, the error after $i+3$ iterations will be:

\[
E_{i+3} = E_{i+2} + AxE_{i+2}2^{-x} + R_{i+2}
\]
Chapter 3 Introducing VSS to DDAs

\[ \begin{align*}
E_0 + A(\text{ir}_i &+ \text{ir}_{i+1}^2 - \text{ir}_i^2) + \text{mr}_i + \text{mr}_{i+1} \\
+ A((i+1)E_0 + A(\text{ir}_{i-1} - \text{ir}_i^2) \text{mr}_i + \text{mr}_{i-1})z^{-k} \\
+ A(E_0 + A(\text{ir}_1 - \text{ir}_{i-1}^2) \text{mr}_1 + \text{mr}_{i-1})z^{-x} \\
+ A(E_0 + A(\text{ir}_{i+1}^2 - \text{ir}_i^2) \text{mr}_{i+1} + \text{mr}_i)z^{-x} \\
+ A(\text{ir}_i^2 z^2 - 2 \times \text{ir}_{i+1} z^2 + \text{ir}_i^2 z^2 + \text{mr}_i + \text{mr}_{i+1}) + 2 \times \text{mr}_{i+1} + \text{mr}_i \\
+ \text{HOTS}
\end{align*} \]

The first line represents the second order truncation error, which remains properly suppressed. The second and third lines infer a quasi, "piecewise" suppression of third order truncation error.

3.2.3 The Incorrect Alternative

An abbreviated analysis of truncation error is sufficient to prove incorrect the alternative proposal outlined above, which involves the shifting of residues prior to addition whenever step-size changes. Since an implementation of this alternative would require greater hardware complexity, it is somewhat fortuitous that it proves less favourable mathematically as well.

Taking the same initial conditions as for the correct alternative in Section 3.2.1, the truncation error after \(i+1\) steps is

\[ \begin{align*}
E_{i+1}^* = E_0 + A(\text{ir}_i - \text{ir}_{i-1}^2) \text{mr}_i - \text{mr}_{i-1} \\
+ A((i+1)E_0 + \sum_{j=0}^{i-1} (A(\text{ir}_j - \text{ir}_{j-1}^2) \text{mr}_j - \text{mr}_{j-1}))z^{-k} \\
+ \text{HOTS .}
\end{align*} \]
When step-size is changed, equation (3.2b) leads to

\[ R_{i+1} = A(i_{i+1}2^n - i_r2^n) + m_{i+1} - m_i \]

hence the truncation error after one more step, considering only the second order term, is

\[ E_{i+2} = E_{i+1} + A\times E_{i+1}2^{-x} + R_{i+1} \]

\[ = E_0 + A(i_{i} - i_{-1})2^{m} + m_{i} - m_{-1} + A(i_{i+1}2^n - i_r2^n) + m_{i+1} - m_i + \text{HOTs} \]

\[ = E_0 + A(i_{i+1}2^n + i_{-1}(2^m-2^n) - i_r2^n) + m_{i+1} - m_{-1} + \text{HOTs} \]

i.e., whenever step-size is changed, residues are not properly cancelled, hence accuracy will be lost.

3.3 Summary

Assuming that Second Order Difference transmission will be implemented, and provided that changes in step-size are restricted to either halving or doubling of the current step-size, the necessary modification of DDA modules to enable them to cope with VSS integration is minor.

With VSS integration, it is possible to ensure that \( \Delta^2Z \) will be limited to three bits in length at all iterations, even the first. To achieve this it is sufficient simply to modify the input and output stages of Second Order Difference DDA integrators and multipliers to implement equation (3.1.1a), and to commence integration with sufficiently small step-size.
No special treatment of residues is necessary to maintain truncation error suppression with VSS integration, even though the integrator residue will change in length whenever step-size is altered.

Thus provided that the sscc signal is available, implementation of a VSSDDA is seen to be feasible. The generation of the sscc signal should ideally be done internally, so that step-size control is autonomous. However, since DDAs are most likely to be used where high speed solution of a known DE is called for - in which case the DDA would be built specifically for that DE, autonomous control may not be necessary. If the DE solution displays similar characteristics across the range of potential initial conditions, so that the required pattern of step-size change can be predetermined, external generation of sscc may be preferable.
4. **AN APPROPRIATE INTEGRATION METHOD FOR VSSDDAs**

The first step towards autonomous control of VSSDDA step-size requires determination of both the type and order of integration algorithm most suitable for hardware VSS integration.

The majority of today's software integration packages employ variable step-size integration algorithms of fourth or greater order accuracy, and many have a variable order capability as well. High order algorithms are chosen in an attempt to provide high speed, highly accurate solutions to DEs, relying on the inherently low discretisation error of the high order algorithm to permit step-sizes sufficiently large to compensate for the associated increase in function evaluations per step. The approach is not successful for all types of DE, and, according to Enright, et al [50], "to be efficient over a wide range of error tolerances, a numerical method must have the ability to vary its order". Subroutines are described in [50] with ranges of order as great as first to thirteenth.

The hardware implementation of such a complex integration package is not only impractical, but is also unnecessary for two reasons:

1. hardware implementations of integration methods are inherently faster than the equivalent software subroutines, hence the improvement in speed generally available through large step-sizes is not necessary;

2. in the engineering applications where DDAs are most likely to be used, the high accuracy demanded of software subroutines by mathematicians is not required, and the wide range of tolerances covered by variable order methods does not need to be catered for.
For these reasons, second or third order methods, with relatively small step-sizes, will generally provide adequate speed and accuracy in DDA implementations.

4.1 Choice of Integration Algorithm Class

A number of second and third order integration algorithms have been developed, but, according to Baker [41], some of these are not suitable for solving DEs on short word-length machines where truncation error suppression techniques are to be used. For example, if the midpoint method is used to solve equation (2.1a), truncation error analysis leads to the third order error term shown:

$$E_{i+1} = \ldots + 2A \left( \sum_{j=1}^{i \div 2} j \times \Delta r_{i-2j+1} \right) 2^{-k} + \ldots$$

The presence of the coefficient $j$ within the summation makes the suppression of third order truncation error accumulation impossible. As stated in [41], "the generalisation of the above (result) is that residue retention is only applicable to Runge-Kutta methods and to those Predictor-Corrector (P-C) methods of the Adams-Moulton type where $y_i$ is built on to compute $y_{i+1}$".

Software VSS integration packages typically calculate an estimate of the local discretisation error (i.e., the discretisation error for one iteration) at each step, and use this to determine the appropriate step-size for the following step. The aim is to meet global accuracy requirements by controlling the discretisation error contributed at each iteration. Making the reasonable assumption that a similar approach is appropriate for hardware VSS integration, the need for a means of estimating the local discretisation error during...
solution favours the P-C methods over the Runge-Kutta methods. P-C methods calculate two estimates of the solution at each iteration, first making a prediction, then a more accurate correction. Provided that the predictor and corrector are of the same order, the difference between them provides a reliable estimate of the local discretisation error.

Appendix A contains an analysis of the discretisation error in examples of both second and third order P-C algorithms. From equation (A.1e), an estimate of the local discretisation error for the second order method is given by:

\[
\text{Error Estimate}(\text{PC2}) = (\text{Corrector} - \text{Predictor}) + 6
\]

and, from equation (A.2e), an estimate of the local discretisation error for the third order method is given by:

\[
\text{Error Estimate}(\text{PC3}) = (\text{Corrector} - \text{Predictor}) + 10
\]

The Runge-Kutta methods, being single step in nature, do not automatically provide such an estimate.

4.2 Choice of Order

Having decided that second or third order algorithms are adequate for DDA implementation, and that P-C methods are preferable for VSS integration, the remaining decision is whether to use a second or a third order P-C algorithm for hardware VSS integration.

4.2.1 Third Order Advantage

The major advantage of a third order algorithm over a second order algorithm is its relatively low discretisation error. As shown in Appendix A, the discretisation error of an appropriate second order
P-C method with constant step-size is (equation (A.1d)):

$$E_{PC2} = - \frac{1}{12} h^3 y_i' + \text{HOTs},$$

and the discretisation error of an appropriate third order P-C method with constant step-size is (equation (A.2d)):

$$E_{PC3} = - \frac{1}{12} h^3 y_i'' + \text{HOTs}.$$

In general, the value of $h$ is much less than unity, so, to achieve a given accuracy, the third order method will clearly allow larger steps to be taken, hence will be faster than the second order method.

### 4.2.2 Second Order Advantage

The major advantage of the second order algorithm over the third order algorithm is its relative simplicity, which is apparent not only in the basic constant step-size algorithms, but even more so in the modifications necessary for VSS integration. Taylor series expansions are used in Appendix B to derive specially modified second and third order formulae for use when step-size is changed, as well as the standard algorithms for second and third order P-C integration with constant step-size.

#### Constant Step-size.

For CSS integration, the second order P-C formula, from equation (B.1a-b), is:

$$y_{p_{i+1}} = y_{ci} + \frac{1}{2} h (3y_{c_{i}} - y_{c_{i-1}})$$  \hspace{1cm} (4.2.2a)

$$y_{c_{i+1}} = y_{ci} + \frac{1}{2} h (y_{c_{i}} + y_{p_{i+1}}),$$  \hspace{1cm} (4.2.2b)

where $y_{p_{i+1}}$ refers to the predictor estimate of $y_{i+1}$, and $y_{c_{i+1}}$ refers to the corrector estimate. The third order formula, from equation (B.2a-b), is:

$$y_{p_{i+1}} = y_{ci} + \frac{1}{12} h (23y_{c_{i}} - 16y_{c_{i-1}} + 5y_{c_{i-2}})$$  \hspace{1cm} (4.2.2c)

$$y_{c_{i+1}} = y_{ci} + \frac{1}{12} h (8y_{c_{i}} - y_{c_{i-1}} + 5y_{p_{i+1}}).$$  \hspace{1cm} (4.2.2d)
Three aspects of the relative complexity of the third order method are immediately obvious: the increased number of terms compared to the second order formula; the larger coefficients, which require more complex multiplications; and, most importantly, the need to divide by 12 in the third order formula, instead of by 2 as in the second order formula. Division by 2 is a simple matter of a right shift by one bit, but division by 12 requires an actual division, or an equivalent multiplication by $\frac{1}{12}$. Because $\frac{1}{12}$ does not have an exact representation in binary, some of the accuracy gained by choice of the third order method would immediately be lost, as was noted in Chapter 1.

**Variable step-size.**

Other aspects of the relative simplicity of the second order method become obvious with VSS integration, where step-size changes are best restricted to halving and doubling in a hardware implementation, for reasons already noted. When step-size, $h$, is changed, modified integration formulae are necessary for the next one or two iterations, because the subscripts attached to the past $y_c$ values in equations (4.2.2a-d) contain an implicit reference to $h$. Correct redefinition of the past $y_c$ values (for halved and doubled step-size) is discussed in Appendix B, and discussion of some more complex cases can be found in Appendix D.

Summarising the results of Appendix B, the second order formula needs to be modified to:

\[
\begin{align*}
y_p_{i+1} &= y_{c_i} + h(a \cdot y_{c_i} - b \cdot y_{c_{i-1}}) + 4 \\
y_{c_{i+1}} &= y_{c_i} + h(y_{c_i} + y_{p_{i+1}}) + 2
\end{align*}
\]

where the values of the coefficients, $a$ and $b$, to be used when step-size is either constant, halved or doubled, are as shown in Table 4.1.
Table 4.1: Coefficients for Second Order P-C Algorithm

<table>
<thead>
<tr>
<th>step-size</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>halved</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>doubled</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

The modified third order formula is:

\[ y_{c+1} = y_{c} + h(ax_{c} - bx_{c-1} + c_{2}x_{c-2}) + 36 \]  \hspace{1cm} (4.2.2g)

\[ y_{p+1} = y_{c} + h(dx_{c} - ex_{c-1} + f_{2}y_{p+1}) + 36 \]  \hspace{1cm} (4.2.2h)

where the values of the six coefficients: a, b, c, d, e, and f, are given in Table 4.2.

Table 4.2: Coefficients for Third Order P-C Algorithm

<table>
<thead>
<tr>
<th>step-size</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>69</td>
<td>48</td>
<td>15</td>
<td>24</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>halved</td>
<td>1st step</td>
<td>51</td>
<td>21</td>
<td>6</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2nd step</td>
<td>64</td>
<td>33</td>
<td>5</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>doubled</td>
<td>1st step</td>
<td>114</td>
<td>120</td>
<td>42</td>
<td>30</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2nd step</td>
<td>74</td>
<td>78</td>
<td>40</td>
<td>24</td>
<td>3</td>
</tr>
</tbody>
</table>

Different coefficients are needed for the first two iterations after step-size is changed because the third order method uses two previous yc values in computing the next.

Comparing Tables 4.1 and 4.2, and noting that some form of multiplexing will be required to implement them in hardware, the advantage of the second order method for VSS integration is clear: second order VSS integration requires two 3-to-1 multiplexers, whereas...
six 5-to-1 multiplexers are necessary for third order VSS integration. As mentioned in Appendix B, both the second and third order formulae can be simplified by storing extra previous integration points for use when step-size is doubled, which in hardware would represent a reduction in the multiplexing requirements. However, the price paid would be not only in the form of the extra registers to contain \( y_{c_i-4} \) and \( y_{c_i-3} \) (for third order integration), or \( y_{c_i-2} \) (for second order integration), but also in the associated hardware to select these registers when required.

4.2.3 Step-size Restriction

An additional factor in the choice between second and third order P-C algorithms is the restriction imposed on potential step-sizes by Second Order Difference transmission. As was shown in Chapter 2, the range of step-sizes for which Second Order Difference transmission successfully limits the interconnection bit count is defined roughly by \( \frac{1}{2}M \leq k \leq M \), with step-size \( = 2^{-k} \).

In a DDA which implements P-C integration, each state variable will require two M-bit registers, one for the predictor and one for the corrector, and the preceding discussion suggests that the difference between the contents of these registers may be useful in providing an estimate of local discretisation error for step-size control. However, with the range of step-sizes compatible with Second Order Difference transmission, this is not the case, because in general the difference between the predictor and corrector values will not appear in the most significant M bits unless much larger step-sizes are allowed.
Consider the local discretisation error for the third order P-C method, from equation (A.2d):

\[ E_{PC3} = -\frac{1}{2^n} h^n \tilde{y}_1 + \text{HOTs}, \]

with \( h = 2^{-k} \). If \( k \) is restricted to the range \( \frac{1}{2} M \leq k \leq M \), the factor of \( h^n \) in the error equation is equivalent to a right shift of \( \tilde{y}_1 \) by at least \( 2M \) bits (up to \( 4M \) bits), which means that the value of \( \tilde{y}_1 \) would need to be extremely large for the error to appear in the top \( M \) bits, and thus be detected as a difference between the predictor and corrector register contents.

With the second order P-C method, the corresponding error equation (A.1d) contains the product \( h^3 \tilde{y}_1 \), which implies a right shift of \( \tilde{y}_1 \) by between \( \frac{3}{2}M \) and \( 3M \) bits. Although this still exceeds the length of DDA state registers, the excess is not as large as with higher order methods, making it clear that the second order method is to be favoured for hardware VSS integration.

The difficulty encountered in obtaining an estimate of local discretisation error with short word-lengths has been the major obstacle to the introduction of VSS to DDAs.

4.3 Second Order P-C Register Transfers

Assuming that a DE of the form \( \dot{y} = f(y) \) is being solved, and that \( f \) is a linear function, the second order Adams formula (equation (4.2.2a-b)) can be manipulated to a form convenient for generating a set of register transfers:
Chapter 4  

Choice of Integration Method

\[ y_{P_{i+1}} = y_{c_i} + \frac{3}{2} h \dot{y}c_i + \frac{1}{2} h \dot{y}c_{i-1} \]
\[ = y_{c_i} + h \dot{y}c_i + \frac{1}{2} h (\dot{y}c_i - \dot{y}c_{i-1}) \]
\[ = y_{c_i} + h \dot{y}f(y_{c_i}) + \frac{1}{2} h f(y_{c_i}) - f(y_{c_{i-1}}) \]
\[ = y_{c_i} + h \dot{y}f(y_{c_i}) + \frac{1}{2} h \dot{y}f(y_{c_{i-1}}) \]
\[ = y_{c_i} + h \dot{y}f(y_{c_i} + \frac{1}{2} (y_{c_{i-1}} - y_{c_{i-1}})) \]
\[ = y_{c_i} + h \dot{y}f(y_{c_i} + \frac{1}{2} \Delta y_{c_{i-1}}) \]

The fact that \( f \) is linear was used to derive the 4th line, hence the resulting register transfers are only suitable for linear DEs. The second order Trapezoidal formula can similarly be manipulated:

\[ y_{c_{i+1}} = y_{c_i} + \frac{1}{2} h \dot{y}c_i + \frac{1}{2} h \dot{y}p_{i+1} \]
\[ = y_{c_i} + h \dot{y}c_i - \frac{1}{2} h \dot{y}c_i + \frac{1}{2} h \dot{y}p_{i+1} \]
\[ = y_{c_i} + h \dot{y}c_i + \frac{1}{2} h (\dot{y}p_{i+1} - \dot{y}c_i) \]
\[ = y_{c_i} + h \dot{y}f(y_{c_i}) + \frac{1}{2} h \dot{y}f(y_{p_{i+1}} - y_{c_i}) \]
\[ = y_{c_i} + h \dot{y}f(y_{c_i} + \frac{1}{2} (y_{p_{i+1}} - y_{c_i})) \]
\[ = y_{c_i} + h \dot{y}f(y_{c_i} + \frac{1}{2} \Delta y_{p_{i+1}}) \]

This simplified version of the formula suggests the following set of register transfers for integration with constant step-size:

\[ (YC + \frac{1}{2} h \Delta YC) + PR + \Delta Z, PR \]  \hspace{1cm} (4.3a)
\[ f(\Delta Z) + \Delta YP \]  \hspace{1cm} (4.3b)
\[ (YC + \frac{1}{2} h \Delta YP) + CR + \Delta Z, CR \]  \hspace{1cm} (4.3c)
\[ f(\Delta Z) + \Delta YC \]  \hspace{1cm} (4.3d)
\[ YC + h \Delta YC \rightarrow YC \]  \hspace{1cm} (4.3e)

Proof that these register transfer allow suppression of the accumulation of both second and third order truncation error can be found in Appendix C.

Register transfers to implement VSS integration of linear equations can be produced in similar fashion from equation (4.2.2e-f):
Chapter 4 Choice of Integration Method

\[ y_{p_i+1} = y_{c_i} + h(a \cdot y_{c_i} - b \cdot y_{c_{i-1}}) \]
\[ = y_{c_i} + h((a-b) \cdot y_{c_i} + b \cdot y_{c_{i-1}} - b \cdot y_{c_{i-1}}) \]
\[ = y_{c_i} + h((a-b) \cdot f(y_{c_i}) + b(f(y_{c_i}) - f(y_{c_{i-1}})) \]
\[ = y_{c_i} + h \cdot f((a-b) \cdot y_{c_i} + b(y_{c_i} - y_{c_{i-1}})) \]
\[ = y_{c_i} + h \cdot f((a-b) \cdot y_{c_i} + b \Delta y_{c_{i-1}}) \]

Thus for VSS integration the register transfers become (noting that \((a-b) = 4\) in each row of Table 4.1):

\[ (Y_C + \frac{1}{4} b h \Delta Y_C) + P_R + \Delta Z, P_R \] (4.3f)
\[ f(\Delta Z) \rightarrow \Delta Y_P \] (4.3g)
\[ (Y_C + \frac{1}{4} h \Delta Y_P) + C_R + \Delta Z, C_R \] (4.3h)
\[ f(\Delta Z) \rightarrow \Delta Y_C \] (4.3i)
\[ Y_C + h \Delta Y_C + Y_C \] (4.3j)

where the appropriate values of \(b\) for constant, halved and doubled step-size can be found in Table 4.1 above.

An alternative set of register transfers, which is suitable for nonlinear DEs, is presented in Chapter 6.

4.4 Special Restrictions on Step-size Change

As has been stated, step-size changes will be restricted to either halving or doubling of the current step-size. The standard approach to step-size selection given this limitation is to derive, from a given global error tolerance, a parameter which defines the maximum acceptable local discretisation error: \(L_{\text{max}}\). When the estimate of local discretisation error exceeds \(L_{\text{max}}\), the offending step should be rejected, and recalculated using halved step-size so that the local discretisation error at all iterations falls below \(L_{\text{max}}\). To determine when step-size can safely be doubled in order to
save integration time, a corresponding parameter, $L_{\text{min}}$, can be used. When the estimated local discretisation error falls below $L_{\text{min}}$, step-size can be doubled, but in this case there is no need to recalculate the step, which would actually be a waste of time and not improve the accuracy of the solution.

The ideal relationship between $L_{\text{min}}$ and $L_{\text{max}}$ is determined by the order of the integration method used, e.g., for a second order method, a factor of 2 change in step-size produces a factor of 8 change in local discretisation error, hence, to prevent instability in the selection of step-size (i.e., thrashing), the ratio of $L_{\text{max}}:L_{\text{min}}$ should exceed 8:1. With a third order method, the corresponding ratio of $L_{\text{max}}:L_{\text{min}}$ would need to exceed 16:1.

4.4.1 When Halving is Error Prone

The different formulae derived in Appendix B for use when step-size is changed lead to additional restrictions on when step-size can be halved. In brief, because of the policy of step rejection when error estimates exceed $L_{\text{max}}$, the formulae dictate that step-size should not be halved on the first step after a change in step-size. If such changes in step-size are allowed to occur, the formula for calculating $y_{i+1}$ when step-size is halved does not apply.

Consider a case where step-size has been doubled to compute $y_{i+1}$. The three integration points of interest can be simply illustrated by

\[
\begin{array}{ccc}
  \times & \times & \times \\
  y_{i-1} & y_i & y_{i+1}
\end{array}
\]
where horizontal spacing implies integration step-size.

Now if the error estimate for $y_{i+1}$ exceeds $L_{\text{max}}$, indicating that step-size is too big, the unlikely situation presents itself of having to halve step-size immediately after it has been doubled. Such a case could arise if a DE solution tending towards a constant slope suddenly changes in slope. The response is to reject $y_{i+1}$, halve the step-size, and recalculate from the previous two points a new $y_{i+1}$. Renaming $y_{i-1}$ according to the new step-size, the situation becomes

$$
\begin{array}{ccc}
\times & \times & \times \\
y_{i-1} & y_{i} & y_{i+1}
\end{array}
$$

which is clearly a case of constant step-size, even though step-size has just been halved. Erroneous results will be calculated if the modified integration formula for halved step-size is applied. This problem could be catered for simply by application of the standard formula for constant step-size when necessary.

Now consider the more troublesome case when $y_{i+1}$ has been calculated with halved step-size, but error estimates indicate that step-size is still too large, and consequently needs to be halved immediately after having already been halved. After the first calculation of $y_{i+1}$ the situation will be

$$
\begin{array}{ccc}
\times & \times & \times \\
y_{i-2} & y_{i} & y_{i+1}
\end{array}
$$

where $y_{i+1}$ is to rejected, and step-size halved again. If this is allowed to happen, the situation becomes
for which the formula for halved step-size is obviously incorrect, and for which an entirely new formula needs to be derived. This sort of problem is only likely to occur if initial step-size is chosen too large, or there is a sudden, drastic increase in the slope of the solution.

4.4.2 Simplified Error Estimates

Mathematical analysis (see Appendix A) shows that an estimate of the local discretisation error for the second order P-C integration method, ignoring high order terms, is given by:

\[
\text{local error} = \frac{(\text{Corrector} - \text{Predictor}) + 6}{6}
\]

if step-size is constant (equation (A.1e)). It is not desirable to perform the division by 6 in practice, nor is it necessary, because all that is really needed for step-size selection is a quantity proportional to local discretisation error. The constant of proportionality (i.e., 6) can be taken care of in the initial selection of \( L_{\text{max}} \), so that only a subtraction need be performed at each integration step to produce the error estimate. This simply calculated estimate has obvious advantages in hardware implementation, however, if step-size changes, such an estimate of local discretisation error is unreliable. For example, if step-size is halved, the true error estimate is given by

\[
\text{local error} = \frac{(\text{Corrector} - \text{Predictor}) + 9}{9}
\]

from equation (A.1.1c), and, when step-size is doubled, by

\[
\text{local error} = \frac{2 \times (\text{Corrector} - \text{Predictor}) + 9}{9}
\]
from equation (A.1.2c). In both these cases, the constant of proportionality is no longer 6.

Consequently, and because halving step-size immediately after step-size change is undesirable as discussed above, the error estimate for the first step after changed step-size should be ignored, and any change in step-size at that point disallowed. This is considered to be a reasonable restriction, because DEs typically do not require such rapid changes in step-size.

4.5 Summary

The need for truncation error suppression restricts potential types of integration method to either Runge-Kutta or P-C, and the advantage of local discretisation error estimates, on which to base step-size control, favours selection of a P-C algorithm: in particular, one in which predictor and corrector are of equal order.

The inherent speed of hardware DE solution, and the relatively low accuracy required in engineering applications, make second or third order integration adequate, and, since lower order algorithms are simpler to implement, second order integration is most suitable. In addition, the greater accuracy of a higher order algorithm would make discretisation error estimates more difficult to obtain where short word-lengths are involved.

Consequently, second order Adams prediction, followed by second order Trapezoidal correction, is the chosen strategy. Special predictor formulae are required immediately after step-size is changed, and different error estimation equations then apply, so changing step-size on two consecutive steps is ruled out.
5. AUTONOMOUS DDA STEP-SIZE CONTROL

Having decided on the integration algorithm to be used, attention can now be focussed on the generation of the sscc (step-size-change-control) signal, the importance of which was discussed in Chapter 3. Further discussion in Chapter 4 suggests that the most reliable basis for step-size selection is local discretisation error, and that the generation of suitably accurate estimates of this error in hardware represents the final obstacle to autonomous control of VSSDDA step-size. For step-size control to be autonomous, estimates of local discretisation error need to be acquired from numerical values currently available within the DDA registers. The short word-lengths employed in DDAs to limit hardware costs, and the need to maintain solution accuracy, make such error estimates difficult to obtain.

As has already been made clear, overall numerical integration accuracy is determined by the cumulative effects of both truncation error and discretisation error. It has also been demonstrated that, in a DDA, it is possible to ensure that the effect of the accumulation of truncation error is restricted to the least significant few bits of the M-bit state registers, through the use of residue retention. To take full advantage of such effective suppression of truncation error accumulation, global discretisation error should also be limited to within a similar magnitude, so that the combination of both types of error still effects only the least significant bits of the state registers. For a given integration method, this result can only be achieved through choice of sufficiently small step-size.
For example, the second order P-C method selected in Chapter 4 would be applied with a step-size which ensures that the local discretisation error (of the order of $h^3$) is much less than $2^{-M}$, so that the overall accumulated discretisation error would be of the order of $2^{-M}$. This implies that $k >> \frac{1}{2}M$, and also that the difference between $Y_P$ and $Y_C$, both being of second order accuracy, will tend to be much less than $2^{-M}$. Since the $Y_P$ and $Y_C$ registers are only $M$ bits long, local discretisation error estimates generated by calculating $Y_C-Y_P$ would generally be zero in the desirable step-size range, hence inadequate for step-size control. This problem is magnified if Second Order Difference transmission is used to reduce the interconnection bit count, because step-size must be further reduced, as described in Section 4.2.3, according to $k \geq \frac{1}{2}M$.

Software integration packages depend on long word-lengths for their accuracy - long enough for the least significant bits to be expendable through the accumulation of local truncation and discretisation errors. These expendable least significant bits are actually vital for software VSS integration, since the estimates of local discretisation error necessary for step-size selection are generally derived from them.

Clearly, the major obstacle to the generation of an estimate of local discretisation error in a DDA is the absence of expendable least significant bits in the $Y_C$ and $Y_P$ registers. Two methods of obtaining such expendable bits will be investigated, the most obvious one being to choose a value of $M$ larger than the accuracy required, and allow step-sizes large enough for the least significant bits of $Y_C$ and $Y_P$ to differ. The less obvious alternative, which will be described first, involves the contents of the DDA residue registers.
Figure 5.1: Register Organisation for P-C Integration
5.1 Differences between Residues

A simplified representation of the DDA generation of \( Y_P \) and \( Y_C \) values from state increments (\( \Delta Zs \)), using second order P-C integration, appears in figure 5.1. The diagram is derived from register transfers to be presented in Chapter 6, which, unlike those in Chapter 4, are suitable for nonlinear as well as linear DEs.

As has been argued, the \( M \)-bit quantities, \( Y_C \) and \( Y_P \), are not of sufficient length to provide discretisation error estimates for step-size control, and extra bits on their less significant ends would be useful for that purpose. Observing from figure 5.1 that the DDA residue registers, \( PR \) and \( CR \), contain the less significant bits of the state values from which \( Y_P \) and \( Y_C \) are generated, consider the possibility of transmitting \( PR \), along with \( \Delta Z \), through the function evaluation "black box" to generate \( Y_P \), and \( CR \) along with \( \Delta Z \) to generate \( Y_C \). Of course \( PR \) and \( CR \) would still be retained as residues for the next iteration. All other factors remaining constant, the resulting \( Y_P \) and \( Y_C \) values would each have \( k \) extra bits added to their least significant ends - exactly what is needed for the production of discretisation error estimates.

The question now is whether these high precision values of \( Y_P \) and \( Y_C \) can be used to generate correct state values for the next iteration. Truncation error analysis of this proposal is extremely complicated, hence is not included here, but analysis of a similar "misuse" of residue registers in Chapter 6 reveals that truncation error suppression would not be successful if such an approach were chosen. Consequently, for accurate integration, the normal register transfers would need to be implemented, and, for the production of the
high precision $Y_P$ and $Y_C$ values, extra function generation hardware included in parallel - solely for the purpose of step-size control.

Clearly, the generation of discretisation error estimates in this way would require extra expense and hardware complexity, especially when DEs contain complicated functions. An alternative is to use the difference between $CR$ and $PR$ directly, rather than pass them through the function evaluation process. For a fairly wide range of DEs, this simplified method of error estimation permits very successful control of step-size, as the results in Chapter 8 indicate, however, with other DEs the results are not so encouraging. The reason for this is obvious when the true implication of using $CR-PR$ as an error estimate is examined: the concatenation of $\Delta Z, PR$ is closely related to $y_{C_i}$, while that of $\Delta Z, CR$ is similarly related to $y_{P_i+1}$. Thus the error estimate obtained from the difference between residues is actually equivalent to $y_{P_i+1}-y_{C_i}$, rather than to $y_{C_i+1}-y_{P_i+1}$ as would be most desirable. This sort of error estimate would be more compatible with a first order integration method, and is related to the desired estimate by differentiation, i.e., DE function evaluation. With DEs containing relatively simple derivative functions, such a first order estimate of discretisation error should be closely related to the desired second order estimate, permitting the most appropriate step-size to be chosen, and producing close to maximum solution speed. Otherwise, first order error estimates are unlikely to provide a reliable basis for step-size selection.

Hence for simple DEs, where the extra hardware for the generation of error estimates would not be too expensive, it is not critically important, but for complex DEs, with expensive function evaluation, sufficiently accurate error estimates may not be
obtainable without the extra hardware.

One redeeming feature of first order estimates is that they are available with all integration methods, not just the P-C methods. They also prove to be successful with a wider range of DEs than expected from the above reasoning, as will be seen in Chapter 8.

5.2 Employing Redundant Accuracy

To meet global accuracy requirements, the local discretisation error must be orders of magnitude smaller than the desired maximum global error. Software solution of a variety of DEs indicates that, with second order integration, the ratio of global error : local error can be as high as 1000:1. Hence VSSDDA state registers would require roughly 10 extra bits, added to their least significant ends, to permit error estimates to be obtained from YC-YP without sacrificing any of the original M bits of solution accuracy.

These extra bits make the step-size restriction imposed by Second Order Difference transmission more severe, because, to ensure that L(\Delta Z) \leq 3 with M+10-bit state registers, step-size would have to be limited according to \( \frac{1}{2} \)M+5 \leq k \leq M+10. Such a limitation on step-size defeats the purpose of VSS integration, the aim of which is to generate accurate DE solutions at a higher speed than is possible with CSS integration. Most DEs can be accurately solved using an M-bit CSSDDA with a step-size larger than the maximum possible with an M+10-bit VSSDDA which employs Second Order Difference transmission, making VSS integration actually slower than CSS integration, unless the step-size restriction associated with Second Order Difference transmission can be avoided.
Three potential means of avoiding the restriction will be discussed:

1. reverting to first order difference transmission;
2. retaining Second Order Difference transmission, but allowing increments to exceed three bits in length;
3. resorting to the transmission of higher order differences. Third order differences would be obtained by calculating the difference between successive Second Order Differences.

5.2.1 First Order Difference Transmission

With first order difference transmission, the step-size is governed only by the potential sizes of integrator residues: \(1 \leq k \leq M+10\). However, the interconnection bit count problem is made worse by the presence of the extra 10 bits on the state registers, which dictate that interconnection paths would have to be \(M+11\) bits wide to cater for the maximum length \(\Delta Z\) could attain.

5.2.2 Second Order Difference Transmission

With Second Order Difference transmission, the consequence of allowing \(k\) to fall below \(m+1\), in violation of equation (2.3.2a), is that \(\Delta^2 Z\) values will exceed three bits in length. In fact, with reference to Section 2.3.2 and figure 2.3, Second Order Differences up to \((m+1 - k)+3\) (i.e., \(m-k+4\)) bits in length will be generated.

Assuming that the aim is to allow \(k\) reach its original maximum limit of \(\frac{1}{2}M\), and thus to avoid the introduction of any further restriction on step-size due to the extended state registers, the potential precision of Second Order Differences will be:
Figure 5.2: Third Order Difference Register Lengths
m-k+4 = M+10-2k+4 \quad \{\text{since } m+k = M+10, \text{ i.e., } m = M-k+10\}
\leq M+10-M+4 \quad \{\text{since } k \geq \frac{1}{2} M\}
\leq 14 \text{ bits},

\text{i.e., } 14\text{-bit interconnection paths will be necessary. This is clearly preferable to the } M+11 \text{ bits necessary with first order difference transmission, but is still far in excess of the original three bits.}

5.2.3 Higher Order Difference Transmission

If higher order differences are transmitted, it is possible to further reduce the interconnection bit count necessary with a given step-size, and/or to permit larger step-sizes without having to increase the intermodule path width.

In figure 5.2, m+1 clearly exceeds k, and Second Order Difference registers are lengthened accordingly. Applying the same arguments as in Section 2.3.2, third order differences will not exceed three bits in length (as shown) if :

m+4-k \leq k
\text{i.e. } m+4 \leq 2k
\text{i.e. } M-k+10-k \leq 2k
\text{i.e. } M+14 \leq 3k
\text{i.e. } k \geq \frac{1}{3} M+5

While this expression is clearly better than \(\frac{1}{4} M+5\), for it to be equivalent to \(k \geq \frac{1}{4} M\), \(M\) would need to exceed 30 bits!

In terms of the interconnection bit count for \(k = \frac{1}{4} M\), the path width for third order difference transmission should be:
(m+4-k)-k+3 = m-2k+7
   = M-k+10-2k+7
   = M-\frac{3}{4}M+17
   = 17-\frac{1}{4}M
   = 9 \text{ bits if } M = 16.

As expected, third order differences do allow the intermodule path width to be further reduced, but since the requirement is still greater than three bits, fourth order differences will be considered. These will not exceed three bits in length if:

\[ m-2k+7 \leq k \]
\[ \text{i.e. } m+7 \leq 3k \]
\[ \text{i.e. } M+17 \leq 4k \]
\[ \text{i.e. } k \geq \frac{1}{4}M+4 \]

which is equivalent to \( k \geq \frac{1}{2}M \) if \( M=16 \).

In summary, if extra bits are added to the ends of VSSDDA state registers to facilitate the generation of local discretisation error estimates for autonomous step-size control, and 3-bit interconnection paths are required, then:

1. third order difference transmission is sufficient only if \( M \geq 30 \);
2. fourth order difference transmission is appropriate if \( 30 \geq M \geq 16 \);
3. still higher order differences will be necessary for \( M < 16 \).

In general, \( n \)th order difference transmission allows path widths to be limited to three bits if:

\[ m - (n-2)k + 3(n-2) + 1 \leq k \]
\[ \text{i.e. } k \geq (M + 11 + 3(n-2))/n \]

assuming that register lengths of \( M+10 \) bits are employed.
5.3 Summary

In this Chapter, the feasibility of controlling step-size through numerical values contained within DDA registers has been investigated. The differences between current register values are used to produce estimates of local discretisation error.

In a Second Order Difference DDA, a combination of two factors makes estimates of local discretisation error difficult to obtain. Because DDA state registers are of such limited precision, the step-size restriction associated with Second Order Differences ensures that local discretisation error is too small to be significant. Improved precision is the solution, rather than magnified discretisation error, which would jeopardise overall solution accuracy.

Error estimates, based on the extra precision retained in DDA residue registers, can be applied successfully for step-size control with some DEs, but, since the error estimate generated from them is equivalent to a first order estimate, it is not useful for all DEs.

Genuine second order estimates of discretisation error can be generated, without sacrificing solution accuracy, if register lengths are extended beyond M bits. Unfortunately, Second Order Differences far greater than three bits in length will be generated, because the alternative of further limitation on step-size is unacceptable, defeating the purpose of VSS integration. Higher order difference transmission then becomes necessary if excessive data path widths are to be avoided.

Higher order difference transmission may also find application in DDAs in general, since the restriction on step-size is not as severe as with Second Order Difference transmission.
6. **DDA SOLUTION OF NONLINEAR DEs**

A great variety of DEs fall within the nonlinear classification - if DE function evaluation requires any operation more complex than the multiplication of a state variable by a constant, that DE is considered nonlinear. In particular, if multiplication of one state by another is required, the DE is said to contain a multiplicative nonlinearity: the simplest and most common form of nonlinearity.

DDA solution of DEs containing multiplicative nonlinearities is catered for in the multiplier designs discussed in Chapters 2 and 3, which allow state increments to be multiplied by either constants or state variables. The product of more than two states, or a higher power of a state, may also be generated by the same designs, using a series of \( m \times M \)-bit multiplications simply by connecting the incremental output of one variable multiplier to the input of another. However, DEs containing more complex nonlinearities may require some "preprocessing" before DDA solution is possible. The aim of the preprocessing is to express the DE in a form which contains only multiplicative nonlinearities, and, as amply documented in [51], this can be achieved for most functions using polynomial approximations.

Thus it is of some interest to investigate truncation error suppression during the DDA solution of DEs containing multiplicative nonlinearities. The truncation error analysis following shows that the variable multiplier designs of Chapters 2 and 3 allow both second and third order truncation error to be suppressed adequately with multiplicative nonlinearities, though not quite as completely as is possible with linear DEs.
6.1 Truncation Error Analysis with Nonlinear Equations

The truncation error analysis procedure demonstrated in Chapter 2 on a simple linear DE needs no modification for application to nonlinear DEs, but manipulation of the error equations generated during the analysis is greatly complicated by the presence of nonlinearities. Consequently, to demonstrate the success of variable multipliers in the control of truncation error accumulation during the solution of DEs containing multiplicative nonlinearities, the analysis procedure will be applied to a DE containing only the most simple of all multiplicative nonlinearities - a first order nonlinear DE in which the state value is to be squared:

\[ \dot{y} = y^2. \]

Euler integration, the simplest of integration methods, is again utilised to avoid unnecessary complication of the analysis.

The results turn out to be fairly similar to those in Chapter 2 for the simple linear DE, which have been shown to be directly applicable to higher order integration formulae (see Appendix C, and [29,44,45]) and to more complex DEs (see [47]). Extensive software DDA simulation verifies the reasonable assumption that the results for the simple nonlinear equation are similarly applicable to higher order integration methods and more complex nonlinear DEs.

The correct register transfers for the DDA solution of the above nonlinear equation using Euler integration are:

\[ \begin{align*}
YE + IR & \rightarrow \Delta Z, IR \\
YE \times \Delta Z \cdot h + MR & \rightarrow \Delta YE, MR \\
YE + \Delta YE & \rightarrow YE
\end{align*} \]
YE being the Euler estimate of the solution. Derivation of an equation for the truncation error in \( y_{e_{i+1}} \) (compared to \( y_{i+1} \)) is achieved using the method of Section 2.2.2 and the same preliminary definitions as expressed in equations (2.2.1a-c):

\[
\Delta z_i = y_e - E_i 2^{-M} = y_i - E_i 2^{-M} - \Delta ir_i 2^{-m}
\]

\[
\Delta y_e_i = ye_i \times \Delta z_i 2^{-k} - \Delta mr_i 2^{-M}
\]

\[
= (y_i - E_i 2^{-M})(y_i - E_i 2^{-M} - \Delta ir_i 2^{-m}) 2^{-k} - \Delta mr_i 2^{-M}
\]

\[
= (y_i - E_i 2^{-M})^2 2^{-k} - (y_i - E_i 2^{-M})\Delta ir_i 2^{-M} - \Delta mr_i 2^{-M}
\]

\[
= y_i^2 2^{-k} - 2y_i E_i 2^{-M+k} + (E_i 2^{-M})^2 2^{-k} - y_i \Delta ir_i 2^{-M}
\]

\[
+ E_i 2^{-M} \Delta ir_i 2^{-M} - \Delta mr_i 2^{-M}
\]

\[
= \Delta y_i + \Delta E_i 2^{-M}
\]

therefore

\[
\Delta E_i = 2y_i E_i 2^{-k} - E_i 2^{-M+k} + y_i \Delta ir_i - E_i 2^{-M} \Delta ir_i + \Delta mr_i
\]

\[
= 2y_i E_i 2^{-k} + y_i \Delta ir_i + \Delta mr_i - (E_i 2^{-k} + E_i \Delta ir_i) 2^{-M}
\]

\[
= E_{i+1} - E_i
\]

Because \( M \) is typically approximately twice the size of \( k \), the term in \( 2^{-M} \) can be discarded as a higher order term, so that

\[
E_{i+1} = E_i + 2y_i E_i 2^{-k} + y_i \Delta ir_i + \Delta mr_i + \text{HOTS}
\]

\[
= (1 + 2y_i 2^{-k}) E_i + y_i \Delta ir_i + \Delta mr_i
\]

Now let

\[ X_i = 1 + 2y_i 2^{-k} \]

\[ R_i = y_i \Delta ir_i + \Delta mr_i \]  \hspace{1cm} (6.1a)

and to yield a simplified recursive equation to substitute into itself :
Chapter 6 DDA Solution of Nonlinear DEs

\[ E_{i+1} = X_i E_i + R_i \]
\[ = X_i (X_{i-1} E_{i-1} + R_{i-1}) + R_i \]
\[ = X_i X_{i-1} E_{i-1} + X_i R_{i-1} + R_i \]
\[ = X_i X_{i-1} (X_{i-2} E_{i-2} + R_{i-2}) + X_i R_{i-1} + R_i \]
\[ = X_i X_{i-1} (X_{i-2} E_{i-2} + X_{i-1} R_{i-2} + X_i R_{i-1} + R_i) \]
\[ = \]
\[ \prod_{j=0}^{i-1} X_j E_o + R_i + \sum_{j=0}^{i-1} \left( R_j \prod_{n=j+1}^{i} X_n \right) \]

It can be shown that
\[ \prod_{j=0}^{i} (1 + 2y_i 2^{-k}) = 1 + 2 \sum_{j=0}^{i} y_j 2^{-k} + \text{HOTs} \]

so, ignoring the higher order terms,

\[ E_{i+1} = (1 + 2 \sum_{j=0}^{i} y_j 2^{-k}) E_o + R_i + \sum_{j=0}^{i-1} \left( R_j (1 + 2 \sum_{n=j+1}^{i} y_n 2^{-k}) \right) \]
\[ = E_o + \sum_{j=0}^{i} R_j + 2 \left( \sum_{j=0}^{i-1} y_j E_o + \sum_{j=0}^{i-1} \left( \sum_{n=j+1}^{i} R_j \sum_{n=j+1}^{i} y_n \right) 2^{-k} \right) \quad (6.1b) \]

which is the nonlinear equivalent of equation (2.2.2c). Substituting of equation (6.1a) into (6.1b) leads to

\[ E_{i+1} = E_o + \sum_{j=0}^{i} (y_j \Delta r_j + \Delta m r_j) \]
\[ + 2 \left[ \sum_{j=0}^{i} y_j E_o + \sum_{j=0}^{i-1} \left( (y_j \Delta r_j + \Delta m r_j) \left( \sum_{n=j+1}^{i} y_n \right) \right) \right] 2^{-k} \]
\[ = E_o + \sum_{j=0}^{i} (y_j \Delta r_j) + \sum_{j=0}^{i} (\Delta m r_j) \]
\[ + 2 \left[ \sum_{j=0}^{i} y_j E_o + \sum_{j=0}^{i-1} \left( (y_j \Delta r_j) \left( \sum_{n=j+1}^{i} y_n \right) \right) \right] 2^{-k} \quad (6.1c) \]
6.1.1 Suppression of Second Order Error

Expansion and manipulation of the two most complex terms from the second and third lines of equation (6.1c), with the aid of equation (2.2.1d-e), gives

\[
\sum_{j=0}^{i-1} (y_j \Delta r_j) (\sum_{n=j+1}^{i} y_n)
\]

\[
+ \sum_{j=0}^{i-2} (\Delta r_j) (\sum_{n=j+2}^{i} y_n)
\]

\[
\sum_{j=0}^{i-1} y_{j+1}(y_{j+1} - y_{j+1}) + \sum_{j=0}^{i-2} (\Delta r_j) (\sum_{n=j+2}^{i} y_n)
\]

\[
\sum_{j=0}^{i-1} y_{j+1}(y_{j+1} - y_{j+1}) - \sum_{j=0}^{i-2} (\Delta r_j) (\sum_{n=j+2}^{i} y_n) 2^{-k}
\]

from the second line, and

\[
\sum_{j=0}^{i-1} (\Delta mr_j) (\sum_{n=j+1}^{i} y_n)
\]

\[
= \sum_{j=0}^{i-1} y_{j+1}(mr_j - mr_{j-1})
\]

from the third. Discarding the term in \(2^{-k}\), which becomes a higher order term, the error equation can thus be rewritten as:

\[
E_{i+1} = E_0 + (y_i r_i - y_i r_{i-1}) + \sum_{j=0}^{i-1} (y_j - y_{j+1})r_j + (mr_i - mr_{i-1})
\]

\[
+ 2(\sum_{j=0}^{i-1} y_j E_0 + \sum_{j=0}^{i-1} y_{j+1}(y_j r_j - y_j r_{j-1})
\]

\[
+ \sum_{j=0}^{i-1} y_{j+1}(mr_j - mr_{j-1})) 2^{-k}
\]

\[
= E_0 + (y_i r_i - y_i r_{i-1}) - \sum_{j=0}^{i-1} (\Delta y_j) r_j 2^{-k} + (mr_i - mr_{i-1})
\]

\[
+ 2(\sum_{j=0}^{i-1} y_j E_0 + \sum_{j=0}^{i-1} y_{j+1}(y_j r_j - y_j r_{j-1})
\]

\[
+ \sum_{j=0}^{i-1} y_{j+1}(mr_j - mr_{j-1})) 2^{-k}
\]
\[ E_0 + (y_{i+1} - y_i) + (m_{i+1} - m_i) \]
\[ + 2 \left( \sum_{j=0}^{i-1} y_j E_0 + \sum_{j=0}^{i-1} y_{j+1} (y_{j+1} - y_j + m_{j+1} - m_j) \right) 2^{-k} \]
\[ - \sum_{j=0}^{i-1} (\Delta y_j) m_{j+1} 2^{-k} \]  
(6.1.1a)

Examining the first line of equation (6.1.1a), and comparing it with that of equation (2.2.3a), it is clear that second order error accumulation is suppressed almost as well as for linear equations. Occurrences of "A" in equation (2.2.3a) are simply replaced by subscripted references to the value of "y" for the nonlinear equation. The potential magnitude of the error is slightly greater with the nonlinear equation than with the linear equation, due to the variation of y with time.

6.1.2 Suppression of Third Order Error

Similar analysis using the normal modification of register transfers for the suppression of third order truncation error may be carried out by substituting equations (2.2.4a-b) into equation (6.1c). The error equations so generated would obviously be complex and space consuming, so it is preferable to note further similarities between equations (6.1.1a) and (2.2.3a), and extrapolate based on the results for third order error suppression in Section 2.2.4, and on the results for second order error suppression with nonlinear equations obtained above.

Firstly, consider the second line of equation (6.1.1a), and note that, apart from the factor of 2, it is equivalent to the third order term of equation (2.2.3a) if all references to "y" are replaced by "A". It can reasonably be concluded that modified register transfers will allow the accumulation of this third order error term.
to be controlled as successfully as the second order term above, except that the third order term contains the extra factor of 2.

Secondly, equation (6.1.1a) contains an extra third order error term, which arises from the variation of y values at each iteration. The magnitude of this term is determined primarily by the rate of change of the solution, and is not reduced by implementation of the usual register transfers for suppression of third order truncation error.

It may be concluded that the accumulation of third order truncation error can be suppressed to a certain extent, but not as completely as can the second order error of nonlinear equations, or the third order error of linear equations.

6.2 When m×M-bit Multiplication is Inadequate

Some nonlinear DE functions contain multiplicative nonlinearities which cannot be calculated simply by a series of m×M-bit multiplications. As an example, consider a second order DE containing the term \((y_1^2 - y_2^2)^2\). m-bit versions of \(y_1^2\) and \(y_2^2\) can easily be generated through firstly obtaining incremental versions (ΔZs) of \(y_1\) and \(y_2\) using standard DDA integrators, then forming the products using m×M-bit DDA variable multipliers. An m-bit version of the difference between squares will then be readily available, however, to generate the square of this difference using DDA techniques an M-bit version is also required, hence M-bit versions of \(y_1^2\) and \(y_2^2\) must be generated.

Other DEs can be conceived for which the generation of M-bit intermediate results is not necessary, but would permit faster DE
solution. For example, computation of $y^0$ could be achieved through a series of seven $m\times M$-bit multiplications, but, if $M$-bit versions of $y^2$ and $y^4$ could be generated, only three multiplications would be necessary. The first would generate an $m$-bit version of $y^2$ in the normal DDA manner from $Y$ and $AZ$ registers. Then, assuming that an $M$-bit version is also available, another DDA variable multiplier could be used to generate an $m$-bit version of $y^4$, which could be similarly used in the calculation of the final product.

Three potential solutions to this problem (the absence of $M$-bit intermediate values) will be considered. The first is a departure from DDA philosophy, but is convenient because only the $m$-bit version of $y^2$ is available - simply perform an $m\times m$-bit multiplication. The second alternative is to "make up" an $M$-bit version of $y^2$ by borrowing $k$ bits from the most significant end of the $M$-bit MR register, and concatenating these to the least significant end of the $m$-bit $AZ$ register, thus permitting $m\times M$-bit multiplication in keeping with normal DDA philosophy. As will be shown, neither of these methods provides suppression of the accumulation of even second order truncation error, hence the third alternative should be chosen, that is, perform full $M\times M$-bit multiplications where $M$-bit intermediate products are required.

This is an important result for those DEs in which the need to generate an intermediate product with $M$-bit accuracy is unavoidable. A number of such DEs are included in the set used to test the VSSDDA integration methods developed in this Thesis.
6.2.1 \( m \times m \)-bit Multiplication

To demonstrate the inadequacy of this alternative, truncation error analysis will be applied to an attempt to produce an \( m \)-bit version of \( y^2 \) by squaring \( \Delta Z \) rather than multiplying \( \Delta Z \) by \( Y \). Register transfers to implement such an approach using Euler integration are:

\[
\begin{align*}
YE + IR & \to \Delta Z, IR \\
\Delta Z \times \Delta Z + h & \to MR + AYE, MR \\
YE + AYE & \to YE
\end{align*}
\]

Assuming that the simple nonlinear DE is to be solved.

Derivation of an equation for the error in \( y_{i+1} \) (compared to \( y_{i+1} \)) is carried out in the normal manner:

\[
\begin{align*}
\Delta z_i &= ye_i - \Delta ir_i2^{-m} \\
&= y_i - E_i2^{-M} - \Delta ir_i2^{-m} \\
\Delta y_{i+1} &= \Delta z_i \times \Delta z_i2^{-k} - \Delta mr_i2^{-M} \\
&= (y_i - E_i2^{-M} - \Delta ir_i2^{-m})2^{-k} - \Delta mr_i2^{-M} \\
&= (y_i - E_i2^{-M})2^{-k} - 2(y_i - E_i2^{-M})\Delta ir_i2^{-M} \\
&\quad - \Delta mr_i2^{-M} + (\Delta ir_i2^{-m})2^{-k} \\
&= y_i2^{-k} - 2y_iE_i2^{-M+k} + (E_i2^{-M})2^{-k} - 2y_i\Delta ir_i2^{-M} + 2E_i2^{-M}\Delta ir_i2^{-M} \\
&\quad - \Delta mr_i2^{-M} + (\Delta ir_i2^{-m})2^{-k} \\
&= \Delta y_i + \Delta E_i2^{-M}
\end{align*}
\]

therefore

\[
\Delta E_i = 2y_iE_i2^{-k} - E_i2^{-2(M+k)} + 2y_i\Delta ir_i - 2E_i2^{-M}\Delta ir_i + \Delta mr_i - \Delta ir_i2^{-2m}
\]

\[
= 2y_iE_i2^{-k} + 2y_i\Delta ir_i + \Delta mr_i - (E_i2^{-k} + 2E_i\Delta ir_i)2^{-M} - \Delta ir_i2^{-2m}
\]

\[
= E_{i+1} - E_i
\]

The term in \( 2^{-M} \) can again be discarded as a higher order term, so that
Chapter 6  
DDA Solution of Nonlinear DEs

\[ E_{i+1} = E_i + 2y_iE_i2^{-k} + 2y_i\Delta r_i + \Delta m_r - \Delta r_i2^{-m} \]

\[ = (1 + 2y_i2^{-k})E_i + 2y_i\Delta r_i + \Delta m_r - \Delta r_i2^{-m} \]

Now let

\[ X_i = 1 + 2y_i2^{-k} \]

\[ R_i = 2y_i\Delta r_i + \Delta m_r - \Delta r_i2^{-m} \]  \hspace{1cm} (6.2.1a)

and the result is the same error equation as for the previous register transfers (Section 6.1) except for the definition of \( R_i \), which, in comparison to equation (6.1a), has an extra factor of 2 in the first term, and an additional third term. This has no effect on the manipulation, so substitute equation (6.2.1a) into (6.1b):

\[
E_{i+1} = E_0 + \sum_{j=0}^{i} (2y_j\Delta r_j + \Delta m_r) - \sum_{j=0}^{i-1} (\Delta r_i2^{-m}) + 2\left[ \sum_{j=0}^{i} y_jE_0 + \sum_{j=0}^{i-1} ((2y_j\Delta r_j + \Delta m_r)(\sum_{n=j+1}^{i} y_n) \right. \\
- \left. \sum_{j=0}^{i-1} (\Delta r_i2^{-m})(\sum_{n=j+1}^{i} y_n) ]2^{-k} \right]
\]

Although some of the terms have twice the magnitude of the corresponding terms in equation (6.1c), the most significant difference here is the presence of new terms involving \( \Delta r_i^2 \), as is now shown by manipulation of the appropriate second order term:

\[
\frac{1}{2} (\Delta r_i2^{-m}) = \frac{1}{2} (ir_i - ir_{i-1})2^{-m} \\
= \frac{1}{2} (ir_j^2 - 2ir_jir_{j-1} + ir_{j-1})2^{-m} \\
= (ir_1^2 + ir_{i-1}^2 + 2\sum_{j=0}^{i-1} ir_j^2 - 2\sum_{j=0}^{i-1} (ir_jir_{j-1})2^{-m}
\]

which is clearly not a case of suppressed error.
As may be expected, this result proves that \( m \times m \)-bit multiplication is certainly inadequate, and an \( M \)-bit version of \( y^2 \) must be available to generate \( y^2 \) in only two multiplications without excessive loss of accuracy.

### 6.2.2 Misuse of Residues

The equivalent application of the second alternative is to multiply \( \Delta Z \) by the concatenation of \( \Delta Z \) and \( IR \) (instead of by \( Y \)) to produce \( y^2 \). (Although the original proposal was to borrow bits from the MR register to produce an \( M \)-bit operand for a succeeding multiplication, analysis is simplified by considering the related IR register case.) Implementation of Euler integration leads to the following register transfers and abbreviated error analysis:

\[
Y_E + IR + \Delta Z, IR
\]

\[
(\Delta Z, IR) \times AZ \cdot h + MR + \Delta Y_E, MR
\]

\[
Y_E + \Delta Y_E + Y_E
\]

(derivation of an equation for the error in \( ye_{i+1} \) need only be carried out for a few steps)

\[
\Delta z_i = ye_i - \Delta ir_i 2^{-m}
\]

\[
= y_i - E_i 2^{-M} - \Delta ir_i 2^{-m}
\]

\[
\Delta yE_i = (ye_i + ir_{i-1}) \times \Delta z_i - \Delta mr_i 2^{-m}
\]

\[
= (y_i - E_i 2^{-M} + ir_{i-1})(y_i - E_i 2^{-M} - \Delta ir_i 2^{-m}) - \Delta mr_i 2^{-m}
\]

which clearly leads to terms involving squares of residues which again frustrate error suppression.

It should be obvious that a similar error equation would result from using part of the MR register to extend \( \Delta Y_E \) values to \( M \) bits precision to permit the generation of \( y^2 \).
The underlying problem with this idea is that multiplying an 
m-bit operand by an M-bit operand will generate a product which cannot 
be guaranteed to be accurate to more than m bits, hence a reliable 
M-bit result cannot be so generated.

6.2.3 MxM-bit Multiplication

The one remaining means of obtaining an M-bit \( \Delta YE \) value is to 
perform an MxM-bit multiplication, as illustrated by the following 
register transfers for Euler integration:

\[
YE \times YE \cdot h + MR + \Delta YE,MR \\
YE + \Delta YE = YE
\]

The product of two M-bit quantities will obviously be reliably 
accurate to M bits, hence error analysis is not necessary.

Software simulation verifies that each of the first two 
alternatives lead to excessive loss of accuracy, but that this third 
alternative succeeds in the suppression of error accumulation. The 
unfortunate implication of this conclusion is that an extra type of 
DDA multiplier module should be added to the DDA module set, for use 
with those DEs which require the "double precision" intermediate 
results.

6.3 Second Order P-C Integration

Second order P-C integration is the method of interest for 
VSSDDAs, and, as stated in Chapter 4, the register transfers given in 
equations (4.3a-j) are suitable only for the solution of linear 
equations.
Equivalent register transfers which are also correct for nonlinear DEs may be derived directly from equation (4.2.2e-f), which, for the solution of the general DE: \( \dot{y} = f(y) \), may be rewritten:

\[
\begin{align*}
\text{yp}_{i+1} &= y_{c_i} + h(ax_{c_i} - bx_{c_{i-1}}) + 4 \\
&= y_{c_i} + h(ax(f(y_{c_i}) - bxf(y_{c_{i-1}})) + 4 \\
y_{c_{i+1}} &= y_{c_i} + h(y_{c_i} + y_{p_{i+1}}) + 2 \\
&= y_{c_i} + h(f(y_{c_i}) + f(y_{p_{i+1}})) + 2
\end{align*}
\]

Converting this form into register transfers is a straightforward step, provided that an extra register, \( \Delta YPL \), is used to save the value of \( f(y_{c_i}) \) for use in the next iteration:

\[
\begin{align*}
Y_C + PR &\rightarrow \Delta Z, PR \\
f(\Delta Z) &\rightarrow \Delta Y_P \\
Y_C + h(a\Delta Y_P - b\Delta YPL) + 4 &\rightarrow Y_P \\
\Delta Y_P &\rightarrow \Delta YPL \\
Y_P + CR &\rightarrow \Delta Z, CR \\
f(\Delta Z) &\rightarrow \Delta Y_C \\
Y_C + h(\Delta Y_P + \Delta Y_C) + 2 &\rightarrow Y_C
\end{align*}
\]

Appropriate values of \( a \) and \( b \) are those given in Table 4.1 for constant, halved and doubled step-size.

Proof that these register transfers provide correct suppression of the accumulation of truncation error is carried out in Appendix C using only a linear DE, and the results of software simulation relied on to verify that they are also correct for nonlinear DEs.
6.4 Summary

Truncation error analysis and simulation have been used to show that truncation error suppression is successful with nonlinear DEs, though not quite as complete as it is with linear DEs.

It has also been revealed that operands suitable for m×M-bit multiplication will be unavailable with some nonlinear DEs, unless one or more special M×M-bit multiplications are carried out beforehand. The implication of this discovery is that a second type of multiplier module, implementing M×M-bit multiplication, should be added to the DDA module set, to avoid any limitation on the range of nonlinear DEs which can be solved using a DDA.

Finally, register transfers appropriate for second order P-C solution of both linear and nonlinear equations are presented.
7. A NEW CLASS OF DEs FOR TESTING VSS INTEGRATION

This Chapter presents a new class of second order nonlinear DEs developed for the testing of the important features of VSS numerical integration algorithms. Accurate comparison of the performance of different algorithms can be made using a selection of these equations, generated by the variation of two parameters.

The material following is a slightly modified version of a paper entitled "A Class of Differential Equations for Testing Variable Step-size Integration" [52].

7.1 Test Problems for VSS Integration

Early research into the numerical solution of DEs led to the discovery of a wide range of integration methods, spanning a variety of algorithm classes as well as different orders of accuracy. In answer to a continuing growth in the demand for the accurate, high-speed solution of DEs, the past few decades have seen a similar rapid growth in the number and sophistication of software numerical integration subroutines, which now boast not only VSS integration but variable order of integration method as well.

This proliferation of integration packages has naturally generated wide interest in the comparison of both the subroutines and the variety of integration methods implemented in them. Publications by researchers such as Enright et al [50], Hull et al [48], Krogh [53], Shampine et al [54], and Gladwell et al [55] contain comprehensive sets of comparison criteria and present many useful test problems.
Chapter 7  

New DEs for Testing VSS Integration

The features considered important in a set of problems for testing VSS integration methods are:
1. a range of ratios of largest step-size to smallest step-size;
2. a range of rates of change of step-size;
3. a range of "duty cycles" to permit emphasis of the solution speed improvement of VSS integration over CSS integration.

Most of the test problems recommended in the literature require little variation in step-size. The simplest of them is the well known harmonic equation (problem 4 from Krogh), the most general form being:

\[ \begin{align*}
\dot{y}_1 &= Axy_2, & y_1(0) &= 0 \quad \{y_1 = \sin(x)\} \\
\dot{y}_2 &= -Axy_1, & y_2(0) &= 1 \quad \{y_2 = \cos(x)\}
\end{align*} \]

which requires less than a 10% variation in step-size during solution. The notable exceptions, providing more severe tests of a subroutine's step-size control, are problem 10 from Krogh (hereafter referred to as problem K10) and problem D5 from Hull et al. In order to emphasize the relative simplicity of the new equations, problem K10 is reproduced here:

\[ \begin{align*}
\dot{y}_1 &= y_3, & y_1(0) &= 1.2 \\
\dot{y}_2 &= y_4, & y_2(0) &= 0 \\
\dot{y}_3 &= 2xy_3 + y_1 - \mu_2(y_1 + \mu_1)/r_1^3 - \mu_1(y_1 - \mu_2)/r_2^3, & y_3(0) &= 0 \\
\dot{y}_4 &= -2xy_1 + y_2 - \mu_2xy_3/r_1^3 - \mu_1xy_2/r_2^3, & y_4(0) &= -1.04935750983031990726...
\end{align*} \quad (7.1a) \]

where

\[ \begin{align*}
\mu_1 &= 1/82.45, \quad \mu_2 = 1 - \mu_1, \\
r_1 &= \left((y_1 + \mu_1)^2 + y_2^2\right)^{1/2}, \\
r_2 &= \left((y_1 - \mu_2)^2 + y_2^2\right)^{1/2}.
\end{align*} \quad (7.1b) \]
Figure 7.1: VSS Equation solution with $B \neq 0$, $C = 0$

Figure 7.2: VSS Equation solution with $B \neq 0$, $C = 0$
The new equations, which will be referred to as the VSS Equations, are a nonlinear variant of the harmonic equation, and are produced by replacing the constant, $A$, in the harmonic equation by a nonlinear function:

\[ \begin{align*}
\dot{y}_1 &= f(y_1,y_2)y_2, & y_1(0) &= 0 \\
\dot{y}_2 &= -f(y_1,y_2)y_1, & y_2(0) &= 1
\end{align*} \tag{7.1c} \]

where the general form of the function is:

\[ f(y_1,y_2) = A + Bx(1 - (y_1^2 - y_2^2)^2a)^b + Cxy_1^2c \tag{7.1d} \]

\[ a,b,c = 0,1,2, \ldots \]

Clearly, when $B$ and $C$ equal zero, equation (7.1c) reduces to the harmonic equation, with sinusoidal frequency given by $A$. When $B$ and $C$ are nonzero, the basic sinusoidal solution shape can be tailored to produce test problems displaying a wide range of the above important features. The effect of the second and third terms in equation (7.1d) is to generate localised increases in the sinusoidal frequency at strategic intervals:

1. the term $Bx(1 - (y_1^2 - y_2^2)^2a)^b$ has most influence when $|y_1| - |y_2|$ is small, and hence modifies the basic sinusoidal solution to the shape indicated in figure 7.1;

2. the term $Cxy_1^2c$ dominates when $|y_1| - 1$ is small, consequently reducing the duty cycle of $y_1$ and increasing that of $y_2$ to produce the solution shapes shown in figure 7.2.

The resulting variable frequency harmonic equation requires VSS integration for accurate, high-speed solution.
Chapter 7  New DEs for Testing VSS Integration

Figure 7.3: Easy VSS Equation solution (problem VSS2)

Figure 7.4: More Difficult VSS Equation solution (problem VSS3)
7.2 VSS Equations: Coefficient and Superscript Selection

The minimum sinusoidal frequency, and hence the maximum step-size, is controlled by the coefficient $A$. Small values of $A$ reduce the sinusoidal frequency and allow larger steps to be taken along the "flat" sections of the solution (which are clearly visible in figure 7.3). The maximum sinusoidal frequency, and hence the minimum step-size, is determined by the other two coefficients, $B$ and $C$. Large values produce similarly large localised increases in sinusoidal frequency, which require smaller step-sizes for accurate integration. The ratio of largest to smallest step-size is thus closely related to the ratio of $B$ and $C$ to $A$. This ratio also has a secondary influence on the required rate of change of step-size.

The rate of change of sinusoidal frequency, which is apparent in the sharpness of the solution corners (compare figures 7.3 and 7.4), and which determines the required rate of change of step-size, is primarily dependent on the choice of superscripts: $a$, $b$ and $c$. Large values restrict the effect of the nonlinear terms in equation (7.1d) to short intervals of the solution, and hence produce higher rates of change of sinusoidal frequency, which will require higher rates of change of step-size to maintain accuracy. The choice of superscripts also has a secondary influence on the ratio of largest to smallest step-size.

A test problem which emphasises the solution speed advantage of VSS over CSS integration can be generated simply by combining a high ratio of $B$ and $C$ to $A$ with large superscript values.

Since the two coefficients, $B$ and $C$, are similar in their influence on test problem characteristics, the selection of
coefficients for a set of test problems based on the VSS Equations can be simplified by making the restriction that $B = C$. Similarly, the potential range of test problem characteristics is not limited by letting $a = 1$ and $b = 2c$, to simplify superscript selection. These restrictions lead to an alternate version of equation (7.1d) which is simpler to use:

$$f(y_1, y_2) = A + k x A \times \left[ (1 - (y_1^2 - y_2^2)^2)^{2c} + y_1^{2c} \right]$$

where $c = 0, 1, 2, \ldots$

The importance of the ratio of $B$ and $C$ to $A$ has been given added emphasis by replacing $B (= C)$ by $k x A$.

For a given choice of $k$ and $c$ in equation (7.2a), the overall solution period is determined solely by $A$. It should be noted that the choice of $A$ will have no effect on the characteristics of a VSS test problem. For example, a factor of 10 decrease in the value of $A$ used to produce figure 7.4 would change the horizontal scale of figure 7.4 by a factor of 10 but would not alter the shape. Any sophisticated subroutine would simply choose step-sizes 10 times larger at each point of the solution, producing no change in the ratios between step-sizes.

Hence the definition of a VSS test problem is reduced to the selection of only 2 parameters, $k$ and $c$.

### 7.3 Comparing Test Problems

To permit quantitative comparison of the VSS Equations to those recommended in the literature, the subroutine of Enright et al [56] was chosen for its well-developed step-size selection strategy and highly recommended integration method: 4th-5th order
Runge-Kutta-Fehlburg, commonly called RKF45. (Other subroutines would undoubtedly yield slightly different results for each equation, but would not significantly alter the results for one equation relative to another.) The subroutine was modified to print out, for each test problem, the following data:

1. the ratio of largest to smallest step-size chosen during integration;
2. the maximum relative change in step-size over intervals of 1, 5, 10, 20 and 50 iterations (to provide a measure of rate of change of step-size per iteration);
3. the total number of iterations required to achieve a given accuracy.

The third quantity is compared to the number of iterations required by a CSS version of the same integration method to achieve the same accuracy. Because of the unavailability of analytical solutions to some of the test problems (including the VSS Equations), another integration guaranteed to give greater accuracy was performed simultaneously and the difference between the two solutions used to indicate the error in RKF45. (The comparative solution speeds in the Tables below were obtained with solution error restricted to $10^{-4}$.)

It is important, when comparing rates of change of step-size, to solve each test problem using the same error tolerance. Consider a section of a solution over which the step-size should be halved, irrespective of the tolerance, and which the chosen tolerance allows to be integrated in 10 iterations. If a smaller tolerance were applied, smaller step-sizes would be necessary and more than 10 iterations would be taken, which would imply, incorrectly, that a smaller rate of change of step-size per iteration is required.
Similarly, choice of a larger tolerance would allow larger steps to be taken, leading to the implication that a larger rate of change of step-size is necessary. (The step-size ratios and rates of change in the Tables below were calculated using a tolerance of $10^{-8}$ per iteration, with which the overall solution error ranged from less than $10^{-5}$ for K4, VSS1 and VSS2, through $10^{-4}$ for D4 and VSS3, to $10^{-3}$ for D5, K10 and VSS4.)

7.4 Comparison Results

The step-size change characteristics of two easier problems, K4 (from Krogh) and D4 (from Hull et al), are included with those of D5 and K10 in Table 7.1, which summarises the range of characteristics available with the problems recommended in the literature.

Table 7.1. Range of Characteristics with Problems from Literature

<table>
<thead>
<tr>
<th>Problem</th>
<th>K4</th>
<th>D4</th>
<th>D5</th>
<th>K10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max:Min step-size ratio</td>
<td>1.1</td>
<td>15.0</td>
<td>108.8</td>
<td>280.8</td>
</tr>
<tr>
<td>N=1</td>
<td>1.0</td>
<td>1.2</td>
<td>1.3</td>
<td>1.4</td>
</tr>
<tr>
<td>Relative change in step-size over N steps</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N=5</td>
<td>1.1</td>
<td>1.7</td>
<td>1.9</td>
<td>2.3</td>
</tr>
<tr>
<td>N=10</td>
<td>1.1</td>
<td>2.5</td>
<td>3.0</td>
<td>2.6</td>
</tr>
<tr>
<td>N=20</td>
<td>1.0</td>
<td>4.5</td>
<td>6.9</td>
<td>6.5</td>
</tr>
<tr>
<td>N=50</td>
<td>1.1</td>
<td>15.0</td>
<td>54.2</td>
<td>59.3</td>
</tr>
<tr>
<td>VSS:CSS speed improvement</td>
<td>1.0</td>
<td>4.5</td>
<td>17.0</td>
<td>51.2</td>
</tr>
</tbody>
</table>
Chapter 7 New DEs for Testing VSS Integration

Figure 7.5: Solution Shape of problem VSS1

Figure 7.6: Solution Shape of problem VSS4
The range available with the VSS Equations, which extends beyond that indicated by Table 7.2 below, clearly compares favourably with that of the accepted test problems above.

Table 7.2. Range of Characteristics with VSS Equations

<table>
<thead>
<tr>
<th>Problem</th>
<th>VSS1</th>
<th>VSS2</th>
<th>VSS3</th>
<th>VSS4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max:Min step-size</td>
<td>1.1</td>
<td>62.2</td>
<td>1027.6</td>
<td>45871.7</td>
</tr>
<tr>
<td>ratio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N=1</td>
<td>1.0</td>
<td>1.7</td>
<td>2.1</td>
<td>2.1</td>
</tr>
<tr>
<td>N=5</td>
<td>1.1</td>
<td>3.4</td>
<td>4.7</td>
<td>6.5</td>
</tr>
<tr>
<td>N=10</td>
<td>1.1</td>
<td>7.1</td>
<td>11.9</td>
<td>18.5</td>
</tr>
<tr>
<td>N=20</td>
<td>1.0</td>
<td>24.9</td>
<td>71.7</td>
<td>164.7</td>
</tr>
<tr>
<td>N=50</td>
<td>1.1</td>
<td>37.0</td>
<td>968.6</td>
<td>25718.3</td>
</tr>
<tr>
<td>VSS:CSS speed</td>
<td>1.0</td>
<td>3.7</td>
<td>32.2</td>
<td>470.1</td>
</tr>
<tr>
<td>improvement</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The particular problems chosen for Table 7.2 are defined by:

VSS1: \( A = 0.157080 \), \( k = 0 \) (i.e., the harmonic equation);

VSS2: \( A = 0.012717 \), \( k = 100 \), \( c = 1 \);

VSS3: \( A = 0.018092 \), \( k = 1000 \), \( c = 2 \);

VSS4: \( A = 0.028915 \), \( k = 10000 \), \( c = 4 \).

The parameter values were selected so as to give a solution period of approximately 40 in each case, 40 being similar to the solution times of the problems from the literature, and convenient for checking peaks and zero-crossing points of VSS Equation solutions. The solution shapes of VSS2 and VSS3 have already been given, in figures 7.3 and 7.4 respectively, and those of VSS1 and VSS2 appear in figures 7.5 and 7.6.
The characteristics given in Table 7.1 for problem D5 can be improved by adjustment of a parameter, and similar improvement may also be possible with K10. Attempts were made to adjust the parameter in D5 such that results better than those for VSS4 could be produced, but these attempts failed because arithmetic requirements exceeded the limitations of the machine (a VAX 11/780).

Hence the quantitative comparison of test problem characteristics favours the VSS Equations even though they are relatively simple, a claim which may be verified by comparison of equation (7.1a-b) with (7.2a)).

7.5 Conclusions

The VSS Equations are seen to have the following advantages over the range of equations from the literature to which they were compared:

1. they possess only two states;
2. the only arithmetic functions used in function evaluation are addition and multiplication. Problems K10 and D5 require division and square root as well. Hence function evaluation is significantly more economical with the VSS Equations.
3. a far greater range of ratios and rates of change of step-size are available, by adjusting only two parameters.
8. TEST PROBLEMS AND RESULTS

Two comprehensive sets of numerical integration test problems appeared in the literature at about the same time, compiled independently by Hull et al [48] and Krogh [53]. These sets of problems have received acceptance from other researchers, such as Shampine et al [54] and Gladwell et al [55], who have employed them as the test problems for extensive comparison of numerical integration subroutines. As may be expected, some test problems are common to both sets, and neither set is a subset of the other, so the two sets have been combined, and some of the new VSS Equations added, to yield the set of test problems used to demonstrate the success (or otherwise) of the VSSDDA algorithms.

Not all of the test problems are suitable for DDA implementation in their published form, due to various complexities in their derivative functions. In some cases, only a conversion to state space representation is necessary, and in other cases, the inclusion of extra states to generate functions such as sine or cosine is sufficient to permit evaluation of the derivative. The more troublesome cases are those which require the evaluation of functions which are not easily generated by DDA techniques, such as division and square root. Unfortunately, most of the more difficult test problems from the literature contain such functions, and the test problem set could be sadly depleted by the exclusion of these. However, since the VSS Equations presented in Chapter 7 provide such a wide range of test problem characteristics, those less convenient test problems from the literature can reasonably be replaced by appropriate VSS Equations for the purpose of testing VSSDDAs.
8.1 Test Problems suitable for VSSDDAs

The test problem set compiled by Hull et al is divided into 5 different problem classes: single equations; small systems; moderate systems; orbit equations; and higher order equations. Each class consists of 5 problems, giving a total of 25 test problems, but only 15 of these do not require division or square root. 1 of these 15 is a 51 state version of another 10 state problem, and is excluded because the extra 41 states provide no extra test problem characteristics.

Gladwell added another class of problems, entitled "Systems of Equations arising when Solving Boundary Value Problems by Shooting Methods", but since these problems are complex, and do not provide test problem characteristics which would otherwise be unavailable, they are not included in the set for VSSDDA testing.

Of the 14 test problems recommended by Krogh, 5 are also recommended by Hull et al, 2 are stiff and therefore not relevant for testing VSSDDAs, 1 requires random number generation!, 2 others require division or square root, another is a single second order equation which is also included in state space form, and another is equivalent to the simplest VSS Equation. Hence only 2 extra problems are added from the set compiled by Krogh.

Finally, a group of 4 VSS Equations is included in the set to bring the total number of test problems up to 20.

This Section (8.1) is devoted to a presentation of these test problems, together with their characteristics - as generated by RKF45 in the manner described in Chapter 7. Graphs of the solutions to the equations are included to clarify the discussion.
Chapter 8

Test Problems and Results

Figure 8.1: Solution of problem A1

Figure 8.2: Solution of problem A2
8.1.1 Single Equations - from Hull et al

The problems in this class are first order DEs, each to be solved over the interval from \( x = 0 \) to 20.

**Problem A1**

\[
\dot{y} = -y, \quad y(0) = 1
\]

(solution: \( y = e^{-x} \))

The negative exponential requires a relatively small step-size initially, which can be progressively increased as solution proceeds.

Characteristics:

1. Max:min step-size ratio = 18.7;
2. Relative change in step-size over \( N \) steps:

<table>
<thead>
<tr>
<th>( N )</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.22</td>
<td>2.16</td>
<td>3.56</td>
<td>6.51</td>
<td>15.81</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

**Problem A2**

\[
\dot{y} = -\frac{1}{2}y^3, \quad y(0) = 1
\]

(solution: \( y = (x + 1)^{-\frac{1}{2}} \))

This problem similarly requires a relatively small initial step-size which can be progressively increased, as can be seen from figure 8.2.

Characteristics:

1. Max:min step-size ratio = 19.4;
2. Relative change in step-size over \( N \) steps:

<table>
<thead>
<tr>
<th>( N )</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.12</td>
<td>1.72</td>
<td>2.89</td>
<td>7.67</td>
<td>-</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>-</td>
</tr>
</tbody>
</table>

The last column of the above Table is empty because RKF45 solved this problem in less than 50 iterations.
Figure 8.3: Solution of problem A3

Figure 8.4: Solution of problem A4
Problem A3

\[ \dot{y} = y \cdot \cos(x), \quad y(0) = 1 \quad \text{[solution: } y = e^{\sin(x)} \text{]} \]

This problem is one of those which require the inclusion of extra states to permit function evaluation by DDA techniques. \( \cos(x) \) may be generated by simultaneous solution of the harmonic equation with a constant coefficient of 1:

\[ \begin{align*}
\dot{y}_1 &= y_2, \quad y_1(0) = 0 \quad \text{[solution: } y_1 = \sin(x) \text{]} \\
\dot{y}_2 &= -y_1, \quad y_2(0) = 1 \quad \text{[solution: } y_2 = \cos(x) \text{]} 
\end{align*} \]

Very little variation in step-size is necessary.

Characteristics:
1. Max:min step-size ratio = 1.1;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.06</td>
<td>1.11</td>
<td>1.07</td>
<td>1.13</td>
<td>1.06</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.04</td>
<td>1.06</td>
<td>1.07</td>
<td>1.07</td>
<td>1.13</td>
</tr>
</tbody>
</table>

Problem A4

\[ \dot{y} = \frac{1}{4} y(1 - \frac{1}{20} y), \quad y(0) = 1 \quad \text{[solution: } y = 20/(1 + 19e^{-1/4 x}) \text{]} \]

As suggested by the shape of figure 8.4, this problem requires little variation in step-size, both increasing and decreasing.

Characteristics:
1. Max:min step-size ratio = 2.2;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.16</td>
<td>1.59</td>
<td>1.80</td>
<td>1.86</td>
<td>-</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.31</td>
<td>1.54</td>
<td>1.74</td>
<td>1.44</td>
<td>-</td>
</tr>
</tbody>
</table>
Figure 8.5: Solution of problem B1
Problem A5
\[ \dot{y} = \frac{y - x}{y + x}, \quad y(0) = 4 \]
[solution: \( r = 4e^{\frac{\sqrt{2}}{4} \pi t} \) (polar coordinates)]
This is the first of the test problems excluded from the set for VSSDDAs, in this case due to the division required in the derivative function.

8.1.2 Small Systems - from Hull et al

The second class of problems from Hull et al consists of second and third order, linear and nonlinear DEs. As indicated on the x-axis of each graph, all of the problems from Hull et al are to be solved over the interval from \( x = 0 \) to 20.

Problem B1
\[ \dot{y}_1 = 2(y_1 - y_1 y_2), \quad y_1(0) = 1 \]
\[ \dot{y}_2 = -(y_2 - y_1 y_2), \quad y_2(0) = 3 \]
There is no analytical solution to this problem, which simulates the growth and decline of two conflicting populations. The solution is cyclic as shown in figure 8.5, with relatively "sharp" peaks demanding reduced step-size for accurate integration, hence a moderate variation in step-size is required.

Characteristics:
1. Max:min step-size ratio = 6.7;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.17</td>
<td>1.66</td>
<td>2.25</td>
<td>3.60</td>
<td>5.84</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.37</td>
<td>1.67</td>
<td>2.22</td>
<td>2.43</td>
<td>5.97</td>
</tr>
</tbody>
</table>
Figure 8.6: Solution of problem B2

Figure 8.7: Solution of problem B3
Problem B2

\[ \dot{y}_1 = -y_1 + y_2, \quad y_1(0) = 2 \]
\[ \dot{y}_2 = y_1 - 2y_2 + y_3, \quad y_2(0) = 0 \]
\[ \dot{y}_3 = y_2 - y_3, \quad y_3(0) = 1 \]

This DE simulates a linear chemical reaction, and requires a relatively very small initial step-size as suggested by the solution graph in figure 8.6.

Characteristics:
1. Max: min step-size ratio = 68.1;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relative increase</td>
<td>1.27</td>
<td>2.43</td>
<td>4.02</td>
<td>7.50</td>
</tr>
<tr>
<td></td>
<td>Relative decrease</td>
<td>1.01</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Problem B3

\[ \dot{y}_1 = -y_1, \quad y_1(0) = 1 \]
\[ \dot{y}_2 = y_1 - y_2^2, \quad y_2(0) = 0 \]
\[ \dot{y}_3 = y_2^2, \quad y_3(0) = 0 \]

This nonlinear chemical reaction similarly requires a relatively small step-size initially, with progressively larger step-sizes to follow.

Characteristics:
1. Max: min step-size ratio = 19.9;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relative increase</td>
<td>1.17</td>
<td>2.01</td>
<td>3.19</td>
<td>5.87</td>
</tr>
<tr>
<td></td>
<td>Relative decrease</td>
<td>1.01</td>
<td>1.02</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Figure 8.8: Solution of problem B5
Problem B4

\[
\begin{align*}
\dot{y}_1 &= -y_2 - \frac{y_1 y_3}{(y_1^2 + y_2^2)^{1/2}}, \quad y_1(0) = 3 \\
\dot{y}_2 &= y_1 - \frac{y_2 y_3}{(y_1^2 + y_2^2)^{1/2}}, \quad y_2(0) = 0 \\
\dot{y}_3 &= \frac{y_1}{(y_1^2 + y_2^2)^{1/2}}, \quad y_3(0) = 0
\end{align*}
\]

The presence of both division and square root in this problem justify its exclusion from the set for VSSDDAs.

Problem B5

\[
\begin{align*}
\dot{y}_1 &= y_2 y_3, \quad y_1(0) = 0 \\
\dot{y}_2 &= -y_1 y_3, \quad y_2(0) = 1 \\
\dot{y}_3 &= -0.51 y_1 y_2, \quad y_3(0) = 1
\end{align*}
\]

The Euler equations of motion for a rigid body without external forces have oscillatory solutions, as shown in figure 8.8, which require little variation in step-size.

Characteristics:

1. Max:min step-size ratio = 2.2;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relative increase</td>
<td>1.16</td>
<td>1.63</td>
<td>1.92</td>
<td>2.17</td>
</tr>
<tr>
<td></td>
<td>Relative decrease</td>
<td>1.26</td>
<td>1.55</td>
<td>1.95</td>
<td>2.19</td>
</tr>
</tbody>
</table>

8.1.3 Moderate Systems – from Hull et al

The third class of problems from Hull et al are DE systems of order 10 or more. The solutions of each of the three problems from this class to be used in testing VSSDDA algorithms are presented over multiple graphs, with a limit of five states plotted on each graph to enhance their clarity.
Figure 8.9a: Solution of problem C1 - first 5 states

Figure 8.9b: Solution of problem C1 - last 5 states
Chapter 8
Test Problems and Results

Problem C1

\[
\begin{align*}
\dot{y}_0 &= -y_0, & y_0(0) &= 1 \\
\dot{y}_1 &= y_0 - y_1, & y_1(0) &= 0 \\
\dot{y}_2 &= y_1 - y_2, & y_2(0) &= 0 \\
\dot{y}_3 &= y_2 - y_3, & y_3(0) &= 0 \\
\dot{y}_4 &= y_3 - y_4, & y_4(0) &= 0 \\
\dot{y}_5 &= y_4 - y_5, & y_5(0) &= 0 \\
\dot{y}_6 &= y_5 - y_6, & y_6(0) &= 0 \\
\dot{y}_7 &= y_6 - y_7, & y_7(0) &= 0 \\
\dot{y}_8 &= y_7 - y_8, & y_8(0) &= 0 \\
\dot{y}_9 &= y_8, & y_9(0) &= 0
\end{align*}
\]

The states in this radioactive decay chain follow the curves shown in figure 8.9a-b. The upper sections of \(y_0\) and \(y_9\) are truncated to enhance the clarity of the display of the shapes of the lower magnitude states. Each state peaks slightly after the previous one, and the magnitude of each peak progressively decreases. Hence this problem is another of those requiring a small initial step-size which then generally increases.

Characteristics:

1. Max:min step-size ratio = 9.4;

2. Relative change in step-size over \(N\) steps:

<table>
<thead>
<tr>
<th>(N)</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.09</td>
<td>1.32</td>
<td>1.65</td>
<td>1.88</td>
<td>3.63</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.06</td>
<td>1.07</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Figure 8.10a: Solution of problem C2 - first 5 states

Figure 8.10b: Solution of problem C2 - last 5 states
Problem C2

\begin{align*}
\dot{y}_0 &= -y_0, & y_0(0) &= 1 \\
\dot{y}_1 &= y_0 - 2y_1, & y_1(0) &= 0 \\
\dot{y}_2 &= 2y_1 - 3y_2, & y_2(0) &= 0 \\
\dot{y}_3 &= 3y_2 - 4y_3, & y_3(0) &= 0 \\
\dot{y}_4 &= 4y_3 - 5y_4, & y_4(0) &= 0 \\
\dot{y}_5 &= 5y_4 - 6y_5, & y_5(0) &= 0 \\
\dot{y}_6 &= 6y_5 - 7y_6, & y_6(0) &= 0 \\
\dot{y}_7 &= 7y_6 - 8y_7, & y_7(0) &= 0 \\
\dot{y}_8 &= 8y_7 - 9y_8, & y_8(0) &= 0 \\
\dot{y}_9 &= 9y_8, & y_9(0) &= 0
\end{align*}

Another radioactive decay chain, in which the states all peak at an earlier point than decay away more quickly than in problem C1. After an initial decrease in step-size to integrate over the peaks, step-size can then increase relatively quickly as the solution flattens out.

Characteristics:

1. Max:min step-size ratio = 30.8;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relative increase</td>
<td>1.16</td>
<td>1.71</td>
<td>2.40</td>
<td>3.77</td>
</tr>
<tr>
<td></td>
<td>Relative decrease</td>
<td>2.65</td>
<td>2.41</td>
<td>2.40</td>
<td>1.43</td>
</tr>
</tbody>
</table>

The scales of figures 8.10a-b are chosen (differently) to display the shapes of \( y_1 \) to \( y_9 \) as clearly as possible. As was the case for problem C1, \( y_9 \) is initially 1.0, and \( y_9 \) approaches 1.0 asymptotically.
Figure 8.11a: Solution of problem C3 - first 5 states

Figure 8.11b: Solution of problem C3 - last 5 states
Problem C3

\[ \dot{y}_0 = -2y_0 + y_1, \quad y_0(0) = 1 \]
\[ \dot{y}_1 = y_0 - 2y_1 + y_2, \quad y_1(0) = 0 \]
\[ \dot{y}_2 = y_1 - 2y_2 + y_3, \quad y_2(0) = 0 \]
\[ \dot{y}_3 = y_2 - 2y_3 + y_4, \quad y_3(0) = 0 \]
\[ \dot{y}_4 = y_3 - 2y_4 + y_5, \quad y_4(0) = 0 \]
\[ \dot{y}_5 = y_4 - 2y_5 + y_6, \quad y_5(0) = 0 \]
\[ \dot{y}_6 = y_5 - 2y_6 + y_7, \quad y_6(0) = 0 \]
\[ \dot{y}_7 = y_6 - 2y_7 + y_8, \quad y_7(0) = 0 \]
\[ \dot{y}_8 = y_7 - 2y_8 + y_9, \quad y_8(0) = 0 \]
\[ \dot{y}_9 = y_8 - 2y_9, \quad y_9(0) = 0 \]

This test problem is derived from a parabolic partial DE. The scales of figures 8.11a-b have once again been chosen to display the low magnitude states more clearly, especially \( y_5 \) to \( y_9 \). C3 is another problem for which a small initial step-size is necessary, with progressive increase in step-size possible as integration proceeds.

Characteristics:

1. Max:min step-size ratio = 38.0;
2. Relative change in step-size over \( N \) steps:

<table>
<thead>
<tr>
<th>( N )</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.08</td>
<td>1.46</td>
<td>2.08</td>
<td>4.04</td>
<td>18.24</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.01</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Problem C4

Problem C4 from Hull et al is the same as problem C3, but extended to 51 equations. By extrapolation from figure 8.11a-b, these extra states will make no contribution to the test problem characteristics, hence C4 is not included in the set for testing VSSDDAs.
Problem C5

Problem C5 is a 30 state problem with division required for the derivative function evaluation, hence this problem is also excluded from the VSSDDA set, and is not reproduced here.

8.1.4 Orbit Equations - from Hull et al

This class of test problem is generated simply by the variation of a parameter in the same fourth order DE.

Problem D1

\[
\begin{align*}
\dot{y}_1 &= y_3, & y_1(0) &= 1 - \varepsilon \\
\dot{y}_2 &= y_4, & y_2(0) &= 0 \\
\dot{y}_3 &= -\frac{y_1}{(y_1^2 + y_2^2)^{3/2}}, & y_3(0) &= 0 \\
\dot{y}_4 &= -\frac{y_2}{(y_1^2 + y_2^2)^{3/2}}, & y_4(0) &= \left(\frac{1 + \varepsilon}{1 - \varepsilon}\right)^{1/2}
\end{align*}
\]

where \( \varepsilon = 0.1 \) (\( \varepsilon \) is the eccentricity of the orbit).

Problem D2

As in D1 with \( \varepsilon = 0.3 \).

Problem D3

As in D1 with \( \varepsilon = 0.5 \).

Problem D4

As in D1 with \( \varepsilon = 0.7 \).

Problem D5

As in D1 with \( \varepsilon = 0.9 \).

This set of problems provides a useful range of test characteristics, but the need for division and square root excludes them from the set for testing VSSDDAs.
Figure 8.12: Solution of problem E2
8.1.5 Higher Order Equations - from Hull et al

The problems in this class are second order nonlinear DEs written as systems of first order DEs - as is convenient for DDA implementation. Problems E1 and E5 are both unsuitable for VSSDDA testing, since they require division and square root as shown below, but E2, E3 and E4 are included in the final set.

Problem E1
\[
\dot{y}_1 = y_2, \quad y_1(0) = 0.6713967071418030
\]
\[
\dot{y}_2 = -\left[\frac{y_2}{x+1} + y_1\left(1 - 0.25/(x+1)^2\right)\right], \quad y_2(0) = 0.09540051444747446
\]

This problem is derived from Bessel’s equation of order \(\frac{1}{2}\) with the origin shifted one unit to the left.

Problem E2
\[
\dot{y}_1 = y_2, \quad y_1(0) = 2
\]
\[
\dot{y}_2 = (1 - y_1^2)y_2 - y_1, \quad y_2(0) = 0
\]

This test problem is derived from Van der Pol’s equation, which has a cyclic solution as shown in figure 8.12, requiring relatively small step-size to integrate over the peaks of state 2. Moderate variation in step-size is called for.

Characteristics:
1. Max:min step-size ratio = 4.2;
2. Relative change in step-size over \(N\) steps:

<table>
<thead>
<tr>
<th>(N)</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.24</td>
<td>1.90</td>
<td>2.50</td>
<td>3.26</td>
<td>2.84</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.30</td>
<td>1.61</td>
<td>1.90</td>
<td>2.72</td>
<td>3.29</td>
</tr>
</tbody>
</table>
Figure 8.13: Solution of problem E3
Problem E3

\[ \dot{y}_1 = y_2 , \quad y_1(0) = 0 \]
\[ \dot{y}_2 = \frac{1}{2} y_1^2 - y_1 + 2 \sin(2.78535x) , \quad y_2(0) = 0 \]

The sinusoidal forcing function in this DE can be generated using two extra states, as was necessary for DDA solution of Problem A3. The two state DE required to generate a sinusoid of the appropriate frequency is:

\[ \dot{y}_3 = 2.78535 y_2 , \quad y_3(0) = 0 \quad \{ y_3 = \sin(2.78535x) \} \]
\[ \dot{y}_4 = -2.78535 y_1 , \quad y_4(0) = 1 \quad \{ y_4 = \cos(2.78535x) \} \]

Characteristics:

1. Max:min step-size ratio = 1.1;

2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relative increase</td>
<td>1.02</td>
<td>1.06</td>
<td>1.07</td>
<td>1.04</td>
</tr>
<tr>
<td></td>
<td>Relative decrease</td>
<td>1.03</td>
<td>1.07</td>
<td>1.07</td>
<td>1.04</td>
</tr>
</tbody>
</table>

Although the solution shape of this DE (Figure 8.13) looks substantially more complex than that of the harmonic equation, its characteristics turn out to be fairly similar, with little variation in step-size called for because of the constantly changing slopes.
Figure 8.14a: Solution of problem E4 - state 1

Figure 8.14b: Solution of problem E4 - state 2
Problem E4
\[ \dot{y}_1 = y_2 , \quad y_1(0) = 30 \]
\[ \dot{y}_2 = 0.032 - 0.4y_2^2 , \quad y_2(0) = 0 \]
Due to the large difference in the magnitudes of the two states of this DE, the shapes of the solutions can be adequately displayed only by resorting to separate graphs, using different vertical scales, as is done in figure 8.14a-b.

Characteristics :
1. Max:min step-size ratio = 2.0 ;
2. Relative change in step-size over N steps :

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.06</td>
<td>1.12</td>
<td>1.20</td>
<td>1.09</td>
<td>-</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.37</td>
<td>1.56</td>
<td>1.79</td>
<td>1.72</td>
<td>-</td>
</tr>
</tbody>
</table>

Problem E5
\[ \dot{y}_1 = y_2 , \quad y_1(0) = 0 \]
\[ \dot{y}_2 = (1 - y_2^2)^{1/2}/(25 - x) , \quad y_2(0) = 0 \]
This problem is derived from a linear pursuit equation. It is excluded by the need for both division and square root.

8.1.6 Problems from Krogh

The problems collected by Krogh are not divided into different classes as above, but are presented as a single set covering a range of test problems from simple linear DEs to relatively complex nonlinear and stiff DEs.

Problems K1 and K2

Both of these problems consist of the same DE as problem A1 above. The difference between them is that problem K1 calls for an absolute
Figure 8.15: Solution of problem K3
error test, whereas problem K2 specifies a relative error test.

**Problem K3**

\[ \dot{y} = y \quad y(0) = 1 \quad \text{[solution: } y = e^x] \]

The positive exponential requires progressive reduction of an initially larger step-size, as suggested by the shape of figure 8.15. Since the solution interval is restricted to only 0 to 5, the range of step-sizes required is much smaller than that of the negative exponential (problem A1).

**Characteristics:**

1. Max:min step-size ratio = 2.7;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.02</td>
<td>1.09</td>
<td>1.18</td>
<td>1.37</td>
<td>1.90</td>
</tr>
</tbody>
</table>

**Problem K4**

This problem is the harmonic equation, which was required above to permit DDA solution of problems A3 and E3, and which appears below as problem V1.

**Problem K5**

Problem K5 is the harmonic equation written as a single, second order equation. Since for DDA solution such problems are converted to a state space representation, this test problem is actually identical to problem K4.

**Problem K6**

The Euler equations of motion for a rigid body, which appear above as problem B5.
Figure 8.16: Solution of problem K7
Chapter 8  Test Problems and Results

Problem K7

\[ \dot{y} = x(1 - y) + (1 - x)e^{-x} , \quad y(0) = 1 \]

{solution : \( y = e^{-\frac{1}{2}x^2} - e^{-x} + 1 \)}

For DDA solution of this DE, problem A1 should be solved simultaneously to generate \( e^{-x} \), and access to the cumulative solution interval provided through another state variable. Thus two extra states are required:

\[ \dot{y}_1 = -y_1 , \quad y_1(0) = 1 \quad \text{[solution : } y_1 = e^{-x} \text{]} \]

\[ \dot{y}_2 = 1 , \quad y_2(0) = 0 \quad \text{[}\quad y_2 = x \quad \text{]} \]

The solution shape shown in figure 8.16 suggests that this problem is similar to problem C2, in that initial step-size should be small, then step-size should decrease further to cater for the early oscillations before increasing to maximise solution speed as the oscillations die out.

Characteristics:

1. Max:min step-size ratio = 11.5;

2. Relative change in step-size over \( N \) steps:

<table>
<thead>
<tr>
<th>( N )</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.22</td>
<td>1.86</td>
<td>2.49</td>
<td>4.85</td>
<td>8.79</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>2.07</td>
<td>2.12</td>
<td>2.59</td>
<td>2.76</td>
<td>4.19</td>
</tr>
</tbody>
</table>

Problems K8 and K9

These two problems are the orbit equations (Hull et al, Class D), with eccentricities of 0.0 and 0.6 respectively.
Chapter 8

Test Problems and Results

Problem K10

\[ \begin{align*}
\dot{y}_1 &= y_3, & y_1(0) &= 1.2 \\
\dot{y}_2 &= y_4, & y_2(0) &= 0 \\
\dot{y}_3 &= 2y_4 + y_1 - \mu_2(x(y_1 + \mu_1))/r_1^3 - \mu_1(x(y_1 - \mu_2))/r_2^3, & y_3(0) &= 0 \\
\dot{y}_4 &= -2y_3 + y_2 - \mu_2(x(y_2 + \mu_2))/r_1^3 - \mu_1(x(y_2 - \mu_2))/r_2^3, & y_4(0) &= -1.04935750983031990726...
\end{align*} \]

where

\[ \begin{align*}
\mu_1 &= 1/82.45, & \mu_2 &= 1 - \mu_1, \\
r_1 &= \left((y_1 + \mu_1)^2 + y_2^2\right)^{1/2}, \\
r_2 &= \left((y_1 - \mu_2)^2 + y_2^2\right)^{1/2}
\end{align*} \]

This test problem provides the most severe test of a subroutine’s step-size control of all the published test problems, but is unsuitable for testing VSSDDA algorithms because of the need for both division and square root.

Problems K11 to K14

The remaining 4 problems contain an integrable singularity (with square root), stiff components, and a requirement for random number generation. They are included by Krogh for the testing of particular features of integration algorithms beyond the expectations of a first attempt at hardware VSS integration.

8.1.7 VSS Equations

The parameters chosen to define the four VSS Equations included in the test problem set do not attempt to provide the range of test problem characteristics displayed in Chapter 7. However, as the results will show, the increasing complexity of the derivative functions from problem V1 through problem V4 provides a sufficient
Figure 8.17: Solution of problem VI
range to demonstrate the capabilities and limitations of the two VSSDDA algorithms proposed in the preceding Chapters.

Recalling the general form of the VSS Equation nonlinear function from Chapter 7:

\[ f(y_1, y_2) = A + k \times A \times [(1 - (y_1^2 - y_2^2)^2)^2 + y_1^2 c] \]

1. "c" values are chosen below to produce progressively increasing derivative function complexity over the four problems;
2. "k" values are chosen sufficiently small to avoid the solution accuracy problems which may result from large differences in constant and state values when GSP is employed;
3. "A" values are then chosen to produce solution periods of roughly 40, for the reasons given in Chapter 7.

**Problem V1**

Selecting \( k = 0 \) makes \( c \) irrelevant and produces the harmonic equation:

\[
\begin{align*}
\dot{y}_1 &= 0.01572y_2, & y_1(0) &= 0 & \{y_1 = \sin(0.01572x)\} \\
\dot{y}_2 &= -0.01572y_1, & y_2(0) &= 1 & \{y_2 = \cos(0.01572x)\}
\end{align*}
\]

**Characteristics:**

1. Max:min step-size ratio = 1.1;
2. Relative change in step-size over \( N \) steps:

<table>
<thead>
<tr>
<th>( N )</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.02</td>
<td>1.06</td>
<td>1.07</td>
<td>1.04</td>
<td>1.05</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.03</td>
<td>1.06</td>
<td>1.07</td>
<td>1.04</td>
<td>1.04</td>
</tr>
</tbody>
</table>
Chapter 8

Test Problems and Results

Figure 8.18: Solution of problem V2

Figure 8.19: Solution of problem V3
Problem V2

\[ \dot{y}_1 = 0.00283 \left[ 1 + 16 \left( (1 - (y_1^2 - y_2^2)^2 + y_1^2) \right) y_2 \right], \quad y_1(0) = 0 \]

\[ \dot{y}_2 = -0.00283 \left[ 1 + 16 \left( (1 - (y_1^2 - y_2^2)^2 + y_1^2) \right) y_1 \right], \quad y_2(0) = 1 \]

\( c = 1 \) provides derivative function complexity which apparently does not exceed the limitations imposed by first order error estimates in the first VSSDDA method.

Characteristics:

1. Max:min step-size ratio = 17.8;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relative increase</td>
<td>1.30</td>
<td>2.48</td>
<td>4.26</td>
<td>12.12</td>
</tr>
<tr>
<td></td>
<td>Relative decrease</td>
<td>1.52</td>
<td>3.13</td>
<td>5.84</td>
<td>14.32</td>
</tr>
</tbody>
</table>

Problem V3

\[ \dot{y}_1 = 0.00449 \left[ 1 + 16 \left( (1 - (y_1^2 - y_2^2)^2)^n + y_1^n \right) \right] y_2, \quad y_1(0) = 0 \]

\[ \dot{y}_2 = -0.00449 \left[ 1 + 16 \left( (1 - (y_1^2 - y_2^2)^2)^n + y_1^n \right) \right] y_1, \quad y_2(0) = 1 \]

\( c = 2 \) in this problem, and, as shown in the results in the following Section, the complexity of the derivative function begins to frustrate attempts at step-size selection on the basis of first order error estimates.

Characteristics:

1. Max:min step-size ratio = 37.7;
2. Relative change in step-size over N steps:

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relative increase</td>
<td>1.32</td>
<td>2.67</td>
<td>4.99</td>
<td>21.04</td>
</tr>
<tr>
<td></td>
<td>Relative decrease</td>
<td>1.64</td>
<td>3.55</td>
<td>6.69</td>
<td>29.39</td>
</tr>
</tbody>
</table>
Figure 8.20: Solution of problem V4
Chapter 8  Test Problems and Results

Problem V4

\[
\begin{align*}
\dot{y}_1 &= 0.00586 \left[ 1 + 16 \left( (1 - (y_1^2 - y_2^2)^2) + y_1^8 \right) \right] y_2 , \quad y_1(0) = 0 \\
\dot{y}_2 &= -0.00586 \left[ 1 + 16 \left( (1 - (y_1^2 - y_2^2)^2) + y_1^8 \right) \right] y_1 , \quad y_2(0) = 1
\end{align*}
\]

Choice of \( c = 4 \) produces a test problem on which the step-size selection strategy of VSSDDA method 1 falls further behind the second order strategies.

Characteristics:

1. Max:min step-size ratio = 94.9 ;
2. Relative change in step-size over \( N \) steps:

<table>
<thead>
<tr>
<th>( N )</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative increase</td>
<td>1.50</td>
<td>3.18</td>
<td>7.04</td>
<td>31.75</td>
<td>34.07</td>
</tr>
<tr>
<td>Relative decrease</td>
<td>1.89</td>
<td>3.76</td>
<td>8.41</td>
<td>51.62</td>
<td>40.66</td>
</tr>
</tbody>
</table>

8.2 Test Results

The most significant criterion for assessment of the success of the step-size selection strategy of a VSS integration algorithm is the solution speed improvement gained over a CSS integration algorithm based on the same integration formula. For most DEs, the VSS algorithm should require fewer iterations than the CSS algorithm to achieve a given solution accuracy. Hence the ratio of the number of iterations taken by the CSS algorithm to the number taken by the VSS algorithm will be used to measure the success of one step-size selection strategy in comparison to another.

For easy reference to the two VSSDDA step-size selection strategies proposed in Chapter 5, VSSDDA1 will be used to specify an implementation using first order estimates of local discretisation error, and VSSDDA2 to indicate that second order estimates are used.
Results are presented below for four integration algorithms. Along with VSSDDA1 and VSSDDA2, two floating point algorithms are tested to provide a basis for assessment of the success of the DDA algorithms. RKF45 is the first of the floating point algorithms included, because it is a widely tested and recommended algorithm with a well developed step-size selection strategy. The results will indicate what a highly sophisticated algorithm can achieve. The second floating point algorithm uses the same second order P-C integration formula as chosen for VSSDDAs, with changes in step-size restricted to either halving or doubling of the current step-size - as recommended for DDAs. Discretisation error estimates for step-size control are generated from the difference between predictor and corrector values as described in Chapter 4, and the results for this algorithm will provide an indication of the speed improvements the VSSDDA algorithms may reasonably be hoped to achieve.

The test results below are divided into three Sections based on the test problem characteristics and solution graphs of the 20 test problems discussed above. The three readily distinguishable types of test problem are:

1. those requiring little variation in step-size, namely problems A3, A4, B5, E3, E4, K3, and V1;
2. those which permit significant increase of an initially small step-size, but do not require step-size to be decreased at all during solution, such as problems A1, A2, B2, B3, C1, and C3;
3. those which require moderate or substantial increases and decreases in step-size during solution, namely problems B1, C2, E2, K7, V2, V3, and V4.
During generation of the test results following, machine word-length limitations dictated that the highest precision implementation of VSSDDA2 that could be simulated was one providing only 12 bits accuracy, due to the need for extended registers to contain the 10 extra bits for discretisation error estimates (hence \( M+10 = 22 \)), and the need to perform \((M+10) \times (M+10)\)-bit multiplications in some of the test problems. Consequently, 12 bits was chosen as the standard solution accuracy for all four integration methods, and all 20 test problems.

Thus the standard RKF45 results were generated by solving each problem using CSS and VSS versions of RKF45, with CSS step-size and VSS local error tolerance both chosen to achieve overall solution accuracies equivalent to 12 bits.

For the second order methods, CSS versions of the floating point and DDA algorithms were simulated, using step-sizes confined to negative powers of 2, and the number of iterations required to produce a suitably small discretisation error within the 12 bits recorded for each test problem. The solution error was also recorded for each DE, then VSS versions of the second order floating point method and both VSSDDA methods simulated with error tolerances chosen to achieve these same solution accuracies.

One consequence of the restriction of step-sizes to negative powers of 2 for the second order methods was that there was some variation in the solution accuracy achieved by the CSS integration algorithm for different DEs. However, this variation should not significantly effect the results since it was not in excess of one or two bits.
8.2.1 Easy Test Problems

The need for little (or no) variation in step-size is the distinguishing feature of the first of the three sets of test problems defined above. Because they possess this characteristic, the problems in this group are considered "easy" test problems for any VSS integration method. Provided that a reasonable initial step-size is chosen, very little will be required of any step-size control algorithm during solution of such equations.

Table 8.2.1a contains the numbers of integration iterations taken by CSS and VSS versions of each of the four integration algorithms chosen for testing.

As should be expected, the fourth order method is around an order of magnitude faster than the second order methods, between which there is little variation in performance.
The above iteration counts are expressed as ratios in Table 8.2.1b, to facilitate comparison of the different step-size selection algorithms.

Table 8.2.1b: CSS:VSS iteration count ratios for test problems which require little variation in step-size.

<table>
<thead>
<tr>
<th>Integration method</th>
<th>RKF45</th>
<th>Second order P-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td></td>
<td>floating point</td>
</tr>
<tr>
<td>A3</td>
<td>1.01</td>
<td>1.01</td>
</tr>
<tr>
<td>A4</td>
<td>1.08</td>
<td>1.00</td>
</tr>
<tr>
<td>B5</td>
<td>1.14</td>
<td>1.00</td>
</tr>
<tr>
<td>E3</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>E4</td>
<td>1.00</td>
<td>1.14</td>
</tr>
<tr>
<td>K3</td>
<td>0.87</td>
<td>1.03</td>
</tr>
<tr>
<td>V1</td>
<td>1.05</td>
<td>1.00</td>
</tr>
</tbody>
</table>

The more flexible RKF45 step-size selection algorithm can be seen to generally provide slightly better ratios than possible with the halving/doubling restriction placed on the second order algorithms. A surprise result is the ratio obtained by RKF45 for problem K3, where the VSS version actually requires more steps than the CSS version. An explanation of this behaviour may be found in Table 8.2.1c, which contains the step-sizes appropriate for CSS integration, and the maximum and minimum step-sizes chosen during VSS integration, for RKF45 and the second order algorithms.
Table 8.2.1c: Step-sizes chosen during solution of the "easy" test problems.

<table>
<thead>
<tr>
<th>Problem</th>
<th>RKF45</th>
<th>Second Order P-C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CSS</td>
<td>VSS min</td>
</tr>
<tr>
<td>A3</td>
<td>0.195</td>
<td>0.178</td>
</tr>
<tr>
<td>A4</td>
<td>1.541</td>
<td>1.537</td>
</tr>
<tr>
<td>B5</td>
<td>0.208</td>
<td>0.169</td>
</tr>
<tr>
<td>E3</td>
<td>0.063</td>
<td>0.063</td>
</tr>
<tr>
<td>E4</td>
<td>3.162</td>
<td>2.796</td>
</tr>
<tr>
<td>K3</td>
<td>0.097</td>
<td>0.054</td>
</tr>
<tr>
<td>V1</td>
<td>1.946</td>
<td>1.961</td>
</tr>
</tbody>
</table>

The contents of this Table reveal a characteristic of RKF45: the ranges of step-size chosen during VSS integration of each problem often extend below the step-sizes appropriate for the CSS version of RKF45. In general, the minimum variable step-size is slightly less than the constant step-size, with a relatively large difference apparent between them and the maximum variable step-size chosen. Problem K3 represents the extreme case of this behaviour, with the appropriate constant step-size falling close to the middle of the range of step-sizes chosen during VSS integration.

The negative result obtained for K3 suggests that the step-size selection strategy of RKF45 is probably not optimal. With the second order algorithms, similar behaviour was avoided simply by limiting the minimum step-size for VSS integration to the value appropriate for CSS integration. Although this does avoid potential degradation of the second order results, it is not claimed that they are consequently optimised.
8.2.2 Moderate Test Problems

The problems in the second group have in common the requirement that initial step-size be small, and each allows step-size thereafter to be progressively increased. Because a decrease in step-size is never required, these problems only test half of an algorithm's step-size control ability, so they are considered to be test problems of "moderate" difficulty.

The numbers of integration iterations taken by CSS and VSS versions of each of the four integration algorithms are presented in Table 8.2.2a.

<table>
<thead>
<tr>
<th>Problem</th>
<th>RKF45 CSS</th>
<th>VSS</th>
<th>Second Order P-C floating point CSS</th>
<th>VSS</th>
<th>VSSDDA1</th>
<th>VSSDDA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>40</td>
<td>14</td>
<td>160</td>
<td>59</td>
<td>58</td>
<td>59</td>
</tr>
<tr>
<td>A2</td>
<td>42</td>
<td>10</td>
<td>160</td>
<td>52</td>
<td>54</td>
<td>96</td>
</tr>
<tr>
<td>B2</td>
<td>120</td>
<td>28</td>
<td>640</td>
<td>83</td>
<td>86</td>
<td>386</td>
</tr>
<tr>
<td>B3</td>
<td>56</td>
<td>18</td>
<td>160</td>
<td>58</td>
<td>58</td>
<td>57</td>
</tr>
<tr>
<td>C1</td>
<td>64</td>
<td>23</td>
<td>320</td>
<td>112</td>
<td>125</td>
<td>118</td>
</tr>
<tr>
<td>C3</td>
<td>111</td>
<td>31</td>
<td>320</td>
<td>62</td>
<td>62</td>
<td>63</td>
</tr>
</tbody>
</table>

The most significant conclusion to be drawn from these results is that the performance of VSSDDA1 remains comparable to that of the standard second order method, which uses the more appropriate second order estimates of discretisation error for step-size control. The first order estimates employed by VSSDDA1 are adequate for the problems in this group, not because the derivative functions involved
are particularly uncomplicated, but because of the characteristic solution behaviour. All that is required is the increase of an initially small step-size, and, as the solution flattens out, both second and first order estimates of local discretisation error will decrease in magnitude, leading to the progressively increasing step-size required for high solution speed.

VSSDDA2 performs in the manner expected except on problems A2 and B2. The reason for these two anomalous results (out of a total of 80 results) unfortunately continues to elude even the imagination.

Expressing the above iteration counts as ratios simplifies comparison of the relative performances of RKF45 and the second order algorithms.

Table 8.2.2b: CSS:VSS iteration count ratios for test problems which allow substantial increase in step-size.

<table>
<thead>
<tr>
<th>Integration method Problem</th>
<th>RKF45</th>
<th>Second order P-C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>floating point</td>
</tr>
<tr>
<td>A1</td>
<td>2.86</td>
<td>2.71</td>
</tr>
<tr>
<td>A2</td>
<td>4.20</td>
<td>3.08</td>
</tr>
<tr>
<td>B2</td>
<td>4.29</td>
<td>7.71</td>
</tr>
<tr>
<td>B3</td>
<td>3.11</td>
<td>2.76</td>
</tr>
<tr>
<td>C1</td>
<td>2.78</td>
<td>2.86</td>
</tr>
<tr>
<td>C3</td>
<td>3.58</td>
<td>5.16</td>
</tr>
</tbody>
</table>

The apparent absence of any simple relationship between the solution speed improvements achieved by RKF45 and those possible with the second order algorithms requires some explanation.
In general, the ratio of CSS:VSS iterations for a given DE would be expected to be higher with a second order method than with any higher order method. Expressed simply, the reason for this is that higher order methods are better able to cope with those sections of a solution where small step-size is required, such as initial transients or sharp peaks, whereas any integration method can cope with the "flat" sections where large steps can be taken. Of course a higher order method will take larger steps along the flat sections than a second order method is able to, but the difference between second and fourth order step-sizes will generally be smaller along easy sections than around difficult corners. Because the appropriate step-size for CSS integration is determined by the most difficult section of the solution, it follows that the ratio of CSS:VSS iterations should be higher with a lower order method.

This expectation is supported by the results for problems B2 and C3, but the reverse is true with problems A2 and B3, and neither method stands out with problems A1 and C1.

This inconsistent behaviour is easily accounted for. The restriction of second order step-size to negative powers of 2 limits the maximum possible second order step-size to only \( \frac{1}{2} \). This restriction reduces the speed improvement achieved with problems such as A2 and B3, where much larger steps could otherwise have been taken. Because no such restriction was imposed on RKF45, maximum speed improvement was possible on all problems.

Thus two factors determine the relative speed improvements achieved, and, with different DEs, either (or neither) of these may be dominant.
Support for the above arguments may be found in Table 8.2.2c, which contains the constant step-sizes and variable step-size ranges used by RKF45 and the second order methods during solution of the test problems in this the second group.

Table 8.2.2c: Step-sizes chosen during solution of the "moderate" test problems.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Integration method</th>
<th>RKF45 CSS</th>
<th>RKF45 VSS</th>
<th>Second Order P-C CSS</th>
<th>Second Order P-C VSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td></td>
<td>0.509</td>
<td>0.491</td>
<td>3.71</td>
<td>0.1250 0.1250 0.5000</td>
</tr>
<tr>
<td>A2</td>
<td></td>
<td>0.487</td>
<td>0.488</td>
<td>3.892</td>
<td>0.1250 0.1250 0.5000</td>
</tr>
<tr>
<td>B2</td>
<td></td>
<td>0.168</td>
<td>0.158</td>
<td>1.379</td>
<td>0.0313 0.0313 0.5000</td>
</tr>
<tr>
<td>B3</td>
<td></td>
<td>0.359</td>
<td>0.346</td>
<td>4.512</td>
<td>0.1250 0.1250 0.5000</td>
</tr>
<tr>
<td>C1</td>
<td></td>
<td>0.316</td>
<td>0.268</td>
<td>1.929</td>
<td>0.0625 0.0625 0.5000</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td>0.181</td>
<td>0.176</td>
<td>1.545</td>
<td>0.0625 0.0625 0.5000</td>
</tr>
</tbody>
</table>

With all six problems, the second order maximum step-size limit is reached, but this does not necessarily mean that many larger steps would have been taken if allowed. The maximum step-sizes chosen by RKF45 are of greater interest, and these suggest that the restriction imposed on the second order methods would have had most effect with problems A1, A2 and B3. From Table 8.2.2b, these correspond to the problems with which RKF45 achieved a superior solution speed improvement, providing support for the reasoning above.

Comments made in Section 8.2.1 about the step-size selection strategy of RKF45 may be seen to apply also to these moderate test problems - the minimum step-size chosen is usually smaller than the step-size found to be appropriate for CSS integration.
8.2.3 Hard Test Problems

The final group of problems is considered "hard" in comparison to the first two groups, because the problems in this group test the ability of an algorithm to alter step-size in both senses, that is, increasing and decreasing.

The numbers of integration steps taken by the test algorithms to solve the problems in this group appear in Table 8.2.3a.

Table 8.2.3a: Numbers of iterations needed to solve test problems which require significant variation in step-size.

<table>
<thead>
<tr>
<th>Integration method</th>
<th>RKF45</th>
<th>Second Order P-C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CSS</td>
<td>VSS</td>
</tr>
<tr>
<td>Problem</td>
<td>CSS</td>
<td>VSS</td>
</tr>
<tr>
<td>B1</td>
<td>364</td>
<td>252</td>
</tr>
<tr>
<td>C2</td>
<td>136</td>
<td>63</td>
</tr>
<tr>
<td>E2</td>
<td>200</td>
<td>156</td>
</tr>
<tr>
<td>K7</td>
<td>642</td>
<td>421</td>
</tr>
<tr>
<td>V2</td>
<td>153</td>
<td>78</td>
</tr>
<tr>
<td>V3</td>
<td>272</td>
<td>87</td>
</tr>
<tr>
<td>V4</td>
<td>424</td>
<td>101</td>
</tr>
</tbody>
</table>

In all cases except problem V2, the results for VSSDDA1 are noticeably inferior to those achieved by the other two second order algorithms. This is not surprising, since most of the more complex derivative functions present in the set of 20 test problems are found within this final group. In particular, the results for problems V2, V3, and V4 clearly show a degradation in performance as derivative function complexity increases – just as predicted in Chapter 5.
Further comparison of the results is simplified by reference to the solution speed improvement ratios derived from these iteration counts, which are presented in Table 8.2.3b.

Table 8.2.3b: CSS:VSS iteration count ratios for test problems which require significant variation in step-size.

<table>
<thead>
<tr>
<th>Integration method</th>
<th>RKF45</th>
<th>Second order P-C floating point</th>
<th>VSSDDA1</th>
<th>VSSDDA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td>1.44</td>
<td>2.01</td>
<td>1.76</td>
<td>1.70</td>
</tr>
<tr>
<td>C2</td>
<td>2.16</td>
<td>2.34</td>
<td>1.72</td>
<td>1.82</td>
</tr>
<tr>
<td>E2</td>
<td>1.28</td>
<td>1.51</td>
<td>1.46</td>
<td>1.60</td>
</tr>
<tr>
<td>K7</td>
<td>1.52</td>
<td>1.55</td>
<td>1.00</td>
<td>1.19</td>
</tr>
<tr>
<td>V2</td>
<td>1.96</td>
<td>1.96</td>
<td>2.36</td>
<td>2.08</td>
</tr>
<tr>
<td>V3</td>
<td>3.13</td>
<td>2.49</td>
<td>2.11</td>
<td>2.37</td>
</tr>
<tr>
<td>V4</td>
<td>4.20</td>
<td>2.29</td>
<td>1.39</td>
<td>1.86</td>
</tr>
</tbody>
</table>

The relationship between the RKF45 ratios and the second order ratios again exhibits the conflicting effects of the differing orders of the two methods, and the limitation imposed on second order maximum step-size.

More worthy of comment are the minor differences in the results for VSSDDA2 and the equivalent floating point method, which are not as drastic as for problems A2 and B2, and which do not consistently favour one method or the other. Similar, but smaller, discrepancies are also evident in Tables 8.2.1b and 8.2.2b (ignoring problems A2 and B2). The source of these discrepancies is the differing precision of DDA and floating point second order error estimates, which naturally leads to step-size changes occurring at
slightly different points on the solution. Differences follow in the addition/cancellation of local error terms, and, for a particular DE, these could randomly favour either method.

The constant step-sizes and variable step-size ranges for this third group of problems are included below.

Table 8.2.3c: Step-sizes chosen during solution of the "hard" test problems.

<table>
<thead>
<tr>
<th>Integration method</th>
<th>RKF45</th>
<th>Second Order P-C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CSS</td>
<td>VSS</td>
</tr>
<tr>
<td></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>B1</td>
<td>0.055</td>
<td>0.206</td>
</tr>
<tr>
<td>C2</td>
<td>0.147</td>
<td>0.580</td>
</tr>
<tr>
<td>E2</td>
<td>0.100</td>
<td>0.331</td>
</tr>
<tr>
<td>K7</td>
<td>0.070</td>
<td>0.654</td>
</tr>
<tr>
<td>V2</td>
<td>0.262</td>
<td>3.698</td>
</tr>
<tr>
<td>V3</td>
<td>0.147</td>
<td>3.744</td>
</tr>
<tr>
<td>V4</td>
<td>0.094</td>
<td>4.682</td>
</tr>
</tbody>
</table>

No extra conclusions are drawn from these figures, but they do provide further support for the comments made in the last two Sections.
8.2.4 Second Order Solution Accuracies

As mentioned above, the solution accuracy achieved by the second order CSS algorithm was not constant across the 20 test problems, due to the restriction of step-sizes to negative powers of 2. However, as can be seen from Table 8.2.4, the variation in accuracy is not sufficiently large to significantly effect the results or discussion above.

Table 8.2.4: Second Order Solution Accuracies (bits)

<table>
<thead>
<tr>
<th>Easy problems</th>
<th>Solution accuracy</th>
<th>Moderate problems</th>
<th>Solution accuracy</th>
<th>Hard problems</th>
<th>Solution accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3</td>
<td>11</td>
<td>A1</td>
<td>11</td>
<td>B1</td>
<td>10</td>
</tr>
<tr>
<td>A4</td>
<td>11</td>
<td>A2</td>
<td>11</td>
<td>C2</td>
<td>11</td>
</tr>
<tr>
<td>B5</td>
<td>11</td>
<td>B2</td>
<td>10</td>
<td>E2</td>
<td>10</td>
</tr>
<tr>
<td>E3</td>
<td>11</td>
<td>B3</td>
<td>10</td>
<td>K7</td>
<td>11</td>
</tr>
<tr>
<td>E4</td>
<td>11</td>
<td>C1</td>
<td>11</td>
<td>V2</td>
<td>10</td>
</tr>
<tr>
<td>K3</td>
<td>10</td>
<td>C3</td>
<td>10</td>
<td>V3</td>
<td>10</td>
</tr>
<tr>
<td>V1</td>
<td>10</td>
<td></td>
<td></td>
<td>V4</td>
<td>10</td>
</tr>
</tbody>
</table>

The tolerances used to produce these results are not included, firstly because they were made internally dependant on step-size, and are thus difficult to define, and secondly because they do not aid interpretation of the results.
8.3 Summary and Comment

The test results indicate that autonomous control of VSSDDA step-size is a feasible proposition. For the majority of the test problems, first order estimates of local discretisation error are adequate for step-size control, with the more accurate second order estimates of no benefit except on those test problems listed in the "hard" group - primarily with cyclic solutions requiring significant variation in step-size.

However, from Tables 8.2.1c, 8.2.2c, and 8.2.3c, it is obvious that these results were obtained without the step-size restrictions necessary for effective Second Order Difference transmission. Neither the restriction on maximum step-size, given by \( \frac{1}{2} M \leq k \leq M \), where \( M = 12 \) for VSSDDA1 and \( M = 22 \) (effectively) for VSSDDA2, nor the restriction on initial step-size \( (k \geq M-2) \), were imposed during the software simulations with which the set of results were generated.

The effect of imposing such restrictions may now be seen to be quite detrimental to VSSDDA solution speed with most DEs, since step-sizes in excess of the maximum allowed with Second Order Difference transmission were chosen by the VSSDDA methods for all 20 test problems.

It may be concluded that, if VSS integration is to be worthwhile in DDAs, the higher order differences described in Chapter 5 are a necessity - unless wide intermodule data paths are acceptable.
9. **SOLUTION OF STIFF DEs**

Each of the DEs presented so far in this Thesis displays the following characteristic: the rates of decay/change present in its different states are all roughly the same. This feature may readily be observed in the solution graphs of the test problems in Chapter 8, and is typical of non-stiff DEs. The important implication of this solution behaviour, in the context of this Chapter, is that a step-size chosen to suit one of the states will be appropriate also for the other states.

The distinguishing feature of a stiff DE is the presence of time constants with greatly different values. If step-size is chosen to suit the non-stiff components, the solution accuracy of the stiff components (those with small time constants) will not only be very poor, but will suffer from numerical instability unless special integration techniques are employed.

The stiff components of a DE decay away rapidly to their final values during the initial part of the solution, after which the solution shape is dominated by the slowly varying components. Very small step-sizes are obviously required until the stiff components have decayed, after which it may be expected that the step-size could safely be increased to improve solution speed. This is not the case if conventional (explicit) integration methods are used. Even when the stiff components have decayed away to a very low magnitude, the step-size cannot be increased significantly without causing numerical instability. In fact, to avoid instability, the step-size must remain below a value not much greater than the smallest time constant present in the DE.
The quest for high speed solution of stiff DEs without numerical instability led to the discovery of implicit integration methods, which were developed to the point of general acceptance through the 1960s, principally by Gear [3]. In brief, the difference between implicit and explicit integration methods is this: whereas an explicit integration method depends on the derivative of $y_1$ to calculate $y_{1+1}$, implicit methods use the derivative of $y_{1+1}$ instead, even though $y_{1+1}$ itself has not yet been calculated.

To facilitate a more detailed description of the stability problems associated with explicit integration of stiff DEs, and the way in which implicit integration avoids these problems, the following simple stiff DE will be considered:

\[
\begin{align*}
\dot{y}_1 &= -y_1, \quad y_1(0) = 1 \quad \text{(9.a)} \\
\dot{y}_2 &= -1000y_2, \quad y_2(0) = 1 \quad \text{(9.b)} \\
\end{align*}
\]

{Solution: $y_1 = e^{-x}$, $y_2 = e^{-1000x}$}

$y_2$ is clearly the stiff component, with a time constant of $10^{-3}$: one thousandth that of $y_1$.

The presence of only two states, combined with the fact that they are uncoupled, make this stiff DE ideal for discussion purposes. It is not considered to be a serious candidate for inclusion in a general set of stiff test problems, but may easily be transformed to generate a coupled version: one in which both states contain stiff and non-stiff components, for that purpose. Nonlinearities could also be introduced readily if so desired.
Figure 9.1a: Solution with step-size of $0.8 \times 10^{-3}$

Figure 9.1b: Solution with step-size of $1.2 \times 10^{-3}$
9.1 Explicit Integration

Explicit solution of equation (9.a-b) will suffer from instability of \( y_2 \) if, at any time, a step-size appropriate for \( y_1 \) alone is chosen.

Selecting the simplest of explicit methods: Euler integration, as the vehicle for discussion, an appropriate step-size for reasonably accurate integration of \( y_1 \) would be of the order of \( 10^{-1} \), however, as far as the stability of \( y_2 \) is concerned, the critical step-size is only \( 2 \times 10^{-3} \). Step-sizes in excess of this value would lead to instability in the solution generated for \( y_2 \). This may be clearly illustrated by introducing numerical error into the solution for \( y_2 \) after the initial transient has died away, then examining the effect of proceeding from this point with a range of step-sizes.

For example, consider the point \( x = 0.05 \). The analytical value of \( y_2 \) there is less than \( 2 \times 10^{-22} \) - near enough to 0 as far as the following discussion is concerned. Assume, however, that the numerical value of \( y_2 \) has been erroneously calculated as 0.01, perhaps through inappropriate choice of step-size. Accurate integration (using a step-size much smaller than \( 10^{-3} \)) from this point onwards would follow the exponentially decaying transient back to 0 indicated in figures 9.1a and 9.1b in green. The blue traces in figures 9.1a-b show the effect of proceeding with a step-size close to the time constant of \( y_2 \) - rapid (though inaccurate) transition back to 0. The step-size used in figure 9.1a is slightly less than the time constant, and that used in figure 9.1b slightly greater.
Section 9.1

Figure 9.1c: Solution with step-size of $1.7 \times 10^{-3}$

Figure 9.1d: Solution with step-size of $2.3 \times 10^{-3}$
If step-sizes about twice as large as the time constant are chosen: close to the critical value mentioned above, the instability problem quickly becomes obvious. Figure 9.1c shows the slowly decaying oscillation resulting from a step-size a little less than $2 \times 10^{-3}$, and 9.1d the instability problem with a step-size slightly greater than this critical value.

At each step, the derivative of the numerical solution is used to generate the next point. Since the derivative is proportional to the deviation of the current point from the true solution, with constant of proportionality equal to the time constant, a step-size of twice the time constant would clearly lead to $100\%$ overshoot past the true solution at each step, and consequently to stable oscillation.

Thus, for Euler integration, a step-size equal to twice the minimum time constant provides maximum solution speed without instability. A similar result holds for other explicit methods as well [3]. Even the higher order algorithms suffer numerical instability if a step-size in excess of twice the stiff time constant is used - the only advantage of a higher order explicit algorithm under these circumstances is that the solution will diverge more slowly.

It should also be noted that, if a stiff transient is present and the immediate aim is to stabilise it, the optimum step-size is one equal to the stiff time constant. With Euler integration, this is clear from figures 9.1a-b, and only one step would be necessary to force the transient to die away. Higher order methods are better able to follow the exponential decay of the transient, and thus will take more than one step to achieve a similar degree of stabilisation, but the optimum step-size remains equal to the stiff time constant.
9.2 Matrix Representation of DEs

It has previously been stated (Chapter 2) that, if a DE is expressed in state space form, then an equivalent matrix representation of the system is readily produced. Matrices are particularly relevant to implicit integration algorithms, which rely on inverted DE matrices to improve the solution speed possible with stiff DEs.

If a DE is written in matrix form, the states, their initial values, and their derivatives all become vectors. For example, with any second order DE,

\[ y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad y(0) = \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix}, \text{ and so on.} \]

Equation (9.1-a-b) can thus be rewritten more simply as:

\[ \dot{y} = Ay, \quad y(0) = B \quad (9.2a) \]

where

\[ A = \begin{bmatrix} -1 & 0 \\ 0 & -1000 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \]

The A matrix describes the coupling between different states of the DE.

Equation (9.2a) bears a close resemblance to equation (2.1a). Conveniently, the subscripts necessary in equation (9.1-a-b) to differentiate between states are no longer present, which means that subscripts are free to be used to indicate the iteration count - as in Chapter 2. Thus Euler integration of equation (9.2a) can be described conveniently by:

\[ y_{e_{i+1}} = y_{e_i} + Ax_{e_i} \cdot h \]

\[ = [I + A \cdot h]y_{e_i} \quad (9.2b) \]
leading to a matrix appropriate for (explicit) Euler integration, which will be compared later to an equivalent matrix describing implicit integration:

\[
[I + A\times h] = \begin{bmatrix}
1-h & 0 \\
0 & 1-1000\times h
\end{bmatrix}
\]

Note that the condition which produces instability, namely \( h > 2\times10^{-3} \), corresponds to \( 1-1000\times h < -1 \).

9.3 Implicit Integration

To demonstrate the principle behind implicit integration, the implicit version of explicit Euler integration will be used.

As has been stated, implicit integration methods use the derivative of \( y_{i+1} \) instead of \( y_i \), even though \( y_{i+1} \) is not yet known. Modifying the equation for Euler integration: equation (9.2b), accordingly leads to:

\[
y_{i+1} = y_i + A\times y_{i+1} \times h
\]

i.e. \( y_{i+1} - A\times y_{i+1} \times h = y_i \)

i.e. \( [I - A\times h]y_{i+1} = y_i \)

hence the final equation for \( y_{i+1} \) is:

\[
y_{i+1} = [I - A\times h]^{-1}y_i
\]

and the need for matrix inversion is established. Obviously, \( y_{i+1} \) cannot be determined before \( y_{i+1} \) is known, but the inverted matrix implies that \( y_{i+1} \) is being used — hence the term "implicit" integration.

To solve equation (9.2a), the appropriate matrix for implicit (Euler) integration is:
Figure 9.2a: Implicit Solution with step-size of $0.8 \times 10^{-3}$

Figure 9.2b: Implicit Solution with step-size of $2.3 \times 10^{-3}$
\[ [I - A\times h]^{-1} = \begin{bmatrix}
1+h & 0 \\
0 & 1+1000h
\end{bmatrix}^{-1}
= \begin{bmatrix}
\frac{1}{1+h} & 0 \\
0 & \frac{1}{1+1000h}
\end{bmatrix} \]

Using this matrix, the solution shapes for \( y_2 \) for the step-sizes of \( 0.8\times10^{-3} \) and \( 2.3\times10^{-3} \) are as shown in figures 9.2a and 9.2b. These are the same step-sizes as used to produce figures 9.1a and 9.1d with explicit integration.

The reason for the stability of the implicit method is clear: at \( y_{i+1} \), the initial transient will have died away to some extent, thus the error in the derivative will be smaller than at \( y_i \). Selecting a larger step-size will merely allow the transient to decay away further before the derivative is used, hence will not lead to instability. Thus the key to the success of implicit methods is that they avoid using the slope at the beginning of the transient as the basis for calculation of the next point. In terms of matrix coefficients, those shown above for first order implicit integration of equation (9.2a) will obviously never fall below -1 regardless of step-size, so instability is impossible.

The trend evident in figures 9.2a-b reveals that if a step-size of the order of \( 10^{-1} \) is chosen, the relative accuracy of the solution generated for \( y_2 \) will not be very impressive, however, since the large step-sizes will only be employed after the stiff components have decayed away to a very low magnitude, the absolute accuracy will be quite acceptable. The aim of the implicit method thereafter is not accuracy but stability where the stiff components are concerned, allowing large step-sizes to be chosen according to the requirements.
of the non-stiff components.

For equation (9.a-b), the accuracy for the non-stiff component, \( y_1 \), will be similar to that possible with the explicit integration method provided that \( h \) is not too large. Comparing figures 9.1a and 9.2a, it may be concluded that, in general, the error in implicit and explicit solutions of non-stiff components will be similar in magnitude but opposite in sign, for step-sizes significantly less than the non-stiff time constants.

9.3.1 Hardware Implicit Integration

The need to invert matrices makes implicit integration generally unsuitable for hardware implementation. It could be argued that, since DDAs are special purpose machines, most useful when designed to solve a specific DE, the matrix inversion required for implicit integration of a particular stiff DE could be performed in software, then the inverted matrix wired into a DDA built specifically to solve that stiff DE. The problem is that the range of stiff DEs for which this approach would be successful is extremely limited. Firstly, since the step-size, \( h \), appears in nonlinear form in the inverted matrix above, it would generally need to be determined in advance, wired into the DDA along with the inverted matrix, and consequently remain constant. Secondly, nonlinear DEs would be unsolvable, because nonlinearities introduce variables into the matrix, which then must be re-evaluated and inverted each time it is to be used.
Chapter 9

Solution of Stiff DEs

Figure 9.3a: 2 Stabilised Euler Iterations

Figure 9.3b: Many Stabilised Euler Iterations
9.4 New Explicit Stiff Integration Technique

As noted earlier, it is clear from figures 9.1a-b that a stiff transient can be made to decay away very quickly if a few explicit steps are taken using a step-size close to the stiff time constant. This characteristic forms the basis of a new approach to the high speed solution of stiff DEs without numerical instability, based on existing explicit integration algorithms.

An intuitive explanation of this new stiff integration technique is facilitated by considering again explicit Euler solution of the second order, uncoupled, stiff DE defined by equation (9.a-b). Assume that solution has proceeded with step-size much less than $10^{-3}$ to the point $Y_1$ in figure 9.3a, where the stiff state ($y_2$) has decayed away to a negligible value. Now consider the effect of taking one explicit Euler step, using a step-size of $10^{-2}$ - much larger than the maximum permissible for numerical stability. In figure 9.3a, this step takes the solution to point $Y_E$. If the error in the solution at $Y_1$ was below a sufficiently small upper bound, say $\epsilon$, then the error magnitude after the large step (i.e., at point $Y_E$) will not be excessive. Instability will arise if further large steps are taken from $Y_E$, but if a number of small steps are taken until the stiff transient again dies away to less than $\epsilon$, another (one) large step can then safely be taken. In figure 9.3a, the "stabilised" point is labelled $Y_S$, and the result of one repetition of the above procedure is shown. Figure 9.3b displays the solution for over 20 iterations, demonstrating that the solution generated by this approach is indeed stable.
Stabilisation of stiff transients prior to taking a large step can be seen to have the same goal as does implicit integration, that is, to avoid using the slope at the beginning of a stiff transient as the basis for a large step. Whereas implicit methods employ an inverted matrix to effectively obtain the slope at $y_{i+1}$ (instead of $y_i$), the new method actually tracks the transient, using small steps, until it has decayed sufficiently for another large step to be safely taken. The results obtained with both these stiff integration techniques will be similar - stable solution, with low relative accuracy but reasonable absolute accuracy of the stiff components, and reasonable accuracy on the non-stiff components.

The software which produced the solution shown in figures 9.3a-b calculated the magnitude of the change in slope over each small step, and used this to detect sufficient decay of the stiff transients. Further software testing, using higher order explicit algorithms and a range of both coupled and uncoupled stiff DEs, indicates that this is a dependable criterion on which to cease stabilisation.

The advantage of the new method over implicit integration is that it does not require matrix inversion. The algorithms used are no more complex than required for non-stiff DEs, hence hardware implementation is feasible.

9.4.1 Multiple Step-size Integration

The discussion so far has described the application of a dual step-size (DSS) stiff integration technique to the simplest of stiff DEs. Obviously the method can be tailored to suit such a DE exactly, however, it should be clear that it will similarly be successful for
the restricted class of stiff DEs in which the stiff time constants fall within a fairly narrow band. In such a case, the large step-size would be chosen for suitable accuracy on the non-stiff components, and the small step-size for stabilising stiff transients should be roughly the average value of the stiff time constants, so that the minimum number of small steps is necessary.

In general, of course, stiff DEs will not contain such a convenient group of time constants. DSS integration is still successful with DEs containing a wider range of time constants, but large numbers of the small steps are required to stabilise the slower stiff transients. Preliminary simulation suggests that best results under these conditions are obtained with small step-size equal to 1.5 to 1.8 times the smallest time constant.

Alternatively, one or more intermediate step-sizes can be employed to speed the stabilisation of the slower stiff transients. The sequence of step-sizes necessary with the resulting multiple step-size (MSS) integration algorithm will naturally be more complex than that described above for DSS integration. For example, if three step-sizes are employed, the general strategy would be, after each large step, to firstly stabilise the faster stiff transients using a few small steps, then to repeat the combination of an intermediate step followed by a few more small steps (to restabilise the fast transients), until the slower stiff transients have also been stabilised. Preliminary simulations indicate that a set of step-sizes, starting from the minimum time constant and separated by factors of 8 or 10, with the largest not exceeding the largest stiff time constant, will generally give close to the fastest solution speed possible for MSS explicit integration of stiff DEs.
MSS integration is clearly a special case of VSS integration, even though the small and intermediate step-sizes (at least) will themselves be constant, being determined by the time constants present in the DE. The large step-size is assumed to be constant also, but of course could be made variable if so desired. In either case, the most significant changes in step-size are those between large, small and intermediate values, and the explicit integration method most appropriate for hardware implementation of such step-size changes should be chosen for hardware MSS integration.

9.4.2 Choice of Explicit Algorithm

Although most convenient for discussion purposes, Euler integration is not considered appropriate for practical MSS integration for two reasons. Firstly, being only a first order algorithm, Euler integration cannot compete with the large step-sizes possible with higher order methods on the non-stiff components of a DE. Secondly, for the purposes of stiff transient stabilisation, Euler integration is particularly sensitive to differences between small step-size and stiff time constants. Since the stiff DEs encountered in most practical applications may contain a range of time constants, it will generally be impossible to match each stiff component perfectly, and faster stabilisation will be possible if an algorithm less sensitive to these discrepancies is used.

As has already been argued (Chapter 4), second or third order methods provide sufficient speed and accuracy in hardware implementations, and second order is preferable to third order to avoid undesirable complexities and the need for division by 3. A second order algorithm will also naturally be less sensitive than
Euler integration to mismatches between the small step-size and the stiff time constants.

Second order P-C integration has been recommended above for hardware VSS integration, mainly due to the ease with which accurate local discretisation error estimates for step-size control can be generated. With MSS integration, two factors reduce the attraction of this algorithm. Firstly, such local error estimates are unnecessary, because the appropriate sequences of step-sizes are largely predefined. What may be variable is the number of stabilising steps (both small and intermediate) required after a large step, and, as noted above for DSS integration, a simple calculation of change in slope is adequate to detect sufficient transient decay. Secondly, the relative step-size changes necessary with MSS integration are far less convenient than simply halving or doubling, and it has been shown that the need for special predictor formulae can lead to excessive complication of DDA hardware. Hence a P-C algorithm is not necessarily the most appropriate for MSS integration.

To avoid the need for special formulae for use when changing between the large, small and intermediate step-sizes, a second order formula which does not require past points, such as Heun, is to be favoured. Heun integration is equivalent to an Euler prediction followed by a trapezoidal correction.

9.4.3 Dual Step-size Heun Integration

Assuming that integration has reached the point \( y_i \), standard Heun integration first makes an Euler prediction \( (y_{e_{i+1}}) \), then uses \( \frac{1}{2}(y_i + y_{e_{i+1}}) \) for the trapezoidal correction.
In solving a stiff DE, the effect of a large Euler step has already been shown, in figure 9.3a. YE obviously cannot be considered useful for trapezoidal correction because of the presence of such a large stiff transient. Instead, transient stabilisation must first be carried out to reach the point YS in figure 9.3a, from which the corrector can be calculated, using \( \frac{1}{2}(\dot{y}_1 + \dot{y}_{i+1}) \). Finally, a few more small steps should be taken to ensure that a suitable point is available for the next Euler predict - free of any large transients produced by the large correction step.

This strategy has so far been tested on both uncoupled and coupled versions of two fairly simple stiff DEs. The coupled versions, together with their analytical solutions, are:

**2nd order DE**

\[
\begin{align*}
\dot{y}_1 &= 998y_1 + 1998y_2, \quad y_1(0) = 1 \quad \{y_1 = 2e^{-x} - e^{-1000x}\} \\
\dot{y}_2 &= -999y_1 - 1999y_2, \quad y_2(0) = 0 \quad \{y_2 = -e^{-x} + e^{-1000x}\}
\end{align*}
\]

**4th order DE**

\[
\begin{align*}
\dot{y}_1 &= 2y_2 - y_4, \quad y_1(0) = -1 \quad \{y_1 = 2\sin(x) - (1+x)e^{-1000x}\} \\
\dot{y}_2 &= -y_1 - y_3, \quad y_2(0) = 1 \quad \{y_2 = \cos(x)\} \\
\dot{y}_3 &= -y_2 + y_4, \quad y_3(0) = 1 \quad \{y_3 = -\sin(x) + (1+x)e^{-1000x}\} \\
\dot{y}_4 &= -1000000y_1 - 2000000y_3 - 2000y_4, \quad y_4(0) = -999 \\
&\quad \{y_4 = -(999+1000x)e^{-1000x}\}
\end{align*}
\]

The results have shown that, in general, six or seven small Heun steps will be necessary to stabilise the large Euler prediction, with a further two or three required to stabilise the large trapezoidal correction. Hence each large Heun step is actually implemented as a combination of around 10 steps.

With a small step-size of \( 10^{-3} \), and a large step-size of around \( 10^{-1} \) giving three digit solution accuracy, DSS Heun integration...
is able to solve the above two stiff DEs using 1/10 as many steps as would be required by any normal explicit method.

9.5 Summary

The presence of wide ranges of time constants make stiff DEs difficult to solve quickly by normal explicit integration techniques. Unless step-size is limited to a very small value, numerical instability will result.

Implicit integration methods are better able to cope with stiff DEs, taking much larger steps without risk of instability. Stiff transients are produced by the large steps, but their destabilising effects are avoided through matrix inversion.

The need for matrix inversion makes implicit integration unsuitable for hardware implementation. Explicit integration is suitable, and new MSS explicit integration makes possible the high speed DDA solution of stiff DEs without numerical instability. Simulation has shown that the technique of stabilising stiff transients using a few small steps before each large step is taken permits solution speed to be improved by a factor of up to 10 over normal explicit methods.

9.6 Further Research

It has been made clear above that MSS integration is to be favoured over both implicit integration and normal explicit methods for the solution of stiff DEs in hardware. The potential worth of MSS integration for the software solution of stiff DEs is yet to be
determined. Two aspects of MSS integration require further investigation to ascertain the limits of its potential.

Firstly, extensive simulation is required to determine the optimum relationships between small step-size and time constants, and between small and intermediate step-sizes. This is not crucial for hardware implementations, since step-size would be restricted to negative powers of 2, but is necessary if software solution speed is to be maximised. The greater flexibility of software permits more accurate choice of step-size, enabling such optimum relationships to be fully implemented.

Secondly, before a general software MSS integration package can be produced, a mechanism for automatically determining the values of time constants present in a DE is necessary. Again, this is not of importance in hardware implementations, since they will most likely be designed with a specific DE in mind, and the time constants would then be known in advance.

When appropriate relationships and mechanisms have been determined, extensive comparison of MSS and implicit techniques may be carried out, using a wide range of stiff test problems, a variety of languages, and a variety of machines, to provide an indication of the relative merits of software MSS integration. Intuitively, implicit techniques should take fewer iterations, but MSS integration avoids matrix inversion, and the balance of these two factors may favour either method under different circumstances.
Digital Differential Analysers

CONCLUSIONS

The conclusions to be drawn from this Thesis will be presented in separate sections, corresponding to the four areas in which contributions have been made.

VSSDDAs

In the special purpose, dedicated applications where DDAs are of most benefit, the DE to be solved would be wired into the DDA for maximum solution speed, and the pattern of step-size change appropriate during solution of the DE may be able to be predetermined. In such cases, a VSSDDA with external step-size control can easily be constructed, since only minor modifications to Second Order Difference ERDDA modules are necessary to enable them to cope with VSS integration - provided that changes in step-size are restricted to either halving or doubling of the current step-size. The introduction of VSS integration to Second Order Difference ERDDAs is of particular benefit because, with VSS integration, the starting problem inherent in Second Order Difference transmission can be overcome simply by choice of a sufficiently small initial step-size.

However, for Second Order Difference transmission to be effective in limiting interconnection bit count, a restriction must be placed on the maximum step-size, and this restriction limits the solution speed improvement obtainable with VSS integration. The test results tabulated above suggest that DDA VSS integration will be useful with only a small minority of DEs, unless the step-size restriction associated with Second Order Difference transmission can
Conclusions

be avoided.

Fortunately, two means of improving the potential range of step-sizes are available. The first is simply to permit larger step-sizes, and broaden the intermodule data paths to cater for the higher precision increments which will inevitably be generated. If such increases in interconnection bit counts are intolerable, the preferable alternative is to generate and transmit higher order differences, with which it is possible to take larger steps without having to increase the intermodule path widths.

Assuming then that one of these measures (or a combination of the two) is implemented in order to avoid undesirable step-size limitations, significant improvements in solution speed will be made possible by the introduction of VSS integration to DDAs, as is indicated by the simulation results for the "moderate" and "hard" test problems. The test results also show that autonomous control of VSSDDA step-size is certainly feasible. For the majority of the test problems solved by the two VSSDDAs, an easily generated first order error estimate of local discretisation error proves adequate for step-size control. However, if the DE solution is cyclic and provides opportunity for significant variation in step-size, best results will be obtained by implementing registers of higher precision so that second order error estimates can be calculated.

Solution of Nonlinear DEs

Truncation error analysis is significantly more complicated when applied to a nonlinear DE, however, the analytical results for a simple nonlinear DE, and the simulation results for more complex
Conclusions

nonlinearities, indicate quite successful suppression of the accumulation of truncation error.

Second order truncation error accumulation is completely suppressed, for nonlinear as well as linear DEs, by the standard DDA register transfers - the difference is that the maximum magnitude of second order truncation error attainable is slightly greater if the DE is nonlinear. Modified register transfers, which permit complete suppression of third order truncation error accumulation with linear DEs, also successfully suppress most (but not all) of the third order truncation error components arising during solution of nonlinear DEs.

Hence the solution accuracy possible with a nonlinear DE will generally not quite match that possible with a linear DE, but, as shown by the results achieved above for both linear and nonlinear test problems, adequate accuracy will usually be attainable.

New Test Problems

The new second order nonlinear test problems are seen to be especially useful for testing DDAs, because they provide a wide range of test characteristics without the need for division, square root or more complex function evaluation. Most of the previously published test problems are relatively easy tests of step-size control unless they employ such functions, which are not only expensive to evaluate but are also inconvenient for DDA implementation.

A range of the new VSS Equations, generated simply by the variation of two parameters, can be combined with a subset of the published test problems to produce a set providing a similarly wide range of test characteristics through derivative functions more
Conclusions

convenient for testing VSSDDAs.

**Solution of Stiff DEs**

High speed DDA solution of a restricted class (at least) of stiff DEs, which is not possible with normal explicit integration, and not feasible with implicit integration, is made both possible and feasible with the introduction of Multiple Step-size explicit integration.

The numerical instability normally associated with large explicit steps is avoided by stabilising the stiff transients before each large step is taken, using a few small explicit steps.

For the price of relatively minor modification of DDA hardware, stiff DE solution speed can be improved by factors as large as 10, in comparison to the performance possible with normal explicit techniques.
REFERENCES


Digital Differential Analysers


Digital Differential Analysers


Digital Differential Analysers


Digital Differential Analysers


Digital Differential Analysers


A. DISCRETISATION ERROR ANALYSIS

The accuracy of an integration algorithm may be determined by comparing the estimate of $y_{i+1}$ produced by the algorithm with the exact value as determined by Taylor series expansions about $t_i$. In this Appendix, the accuracy of both second and third order P-C algorithms is derived for three distinct cases: constant step-size; halved step-size; and doubled step-size, assuming that the stored previous integration points are of second or third order accuracy as required.

The formulae analysed below are derived in Appendix B, where the need for separate formulae for constant, halved and doubled step-size is also explained. The variation, when step-size is changed, of the subscripts attached to stored previous integration points dictates that a range of Taylor series expansions are necessary for the following discretisation error analysis. The particular Taylor series expansions relevant are:

\begin{align*}
y_{i+1} &= y_i + h\dot{y}_i + \frac{1}{2}h^2\ddot{y}_i + \frac{1}{6}h^3\dddot{y}_i + \frac{1}{24}h^4\ddddot{y}_i + \text{HOTS} \quad (A.a) \\
y_{i-\frac{1}{2}} &= y_i - \frac{1}{2}h\dot{y}_i + \frac{1}{2}h^2\ddot{y}_i - \frac{1}{12}h^3\dddot{y}_i + \text{HOTS} \quad (A.b) \\
y_{i-1} &= y_i - h\dot{y}_i + \frac{1}{2}h^2\ddot{y}_i - \frac{1}{6}h^3\dddot{y}_i + \text{HOTS} \quad (A.c) \\
y_{i-\frac{3}{2}} &= y_i - \frac{3}{2}h\dot{y}_i + \frac{3}{2}h^2\ddot{y}_i - \frac{5}{8}h^3\dddot{y}_i + \text{HOTS} \quad (A.d) \\
y_{i-2} &= y_i - 2h\dot{y}_i + 2h^2\ddot{y}_i - \frac{7}{4}h^3\dddot{y}_i + \text{HOTS} \quad (A.e) \\
y_{i-3} &= y_i - 3h\dot{y}_i + \frac{9}{2}h^2\ddot{y}_i - \frac{25}{8}h^3\dddot{y}_i + \text{HOTS} \quad (A.f) \\
y_{i-4} &= y_i - 4h\dot{y}_i + 8h^2\ddot{y}_i - \frac{83}{8}h^3\dddot{y}_i + \text{HOTS} \quad (A.g) \\
\end{align*}

where HOTs = Higher Order Terms. Each of (A.b-g) is obtained from (A.a) by replacing $h$ by an appropriate multiple of $h$. 
A.1 Accuracy of Second Order P-C Algorithm

From Appendix B, equations (B.1a-b), the formula for a second order algorithm suitable for implementation in hardware is:

\[ y_{p_{i+1}} = y_{c_i} + \frac{h}{2} \dot{y}_{c_i} - \frac{h}{2} \dot{y}_{c_{i-1}} \]

\[ y_{c_{i+1}} = y_{c_i} + \frac{h}{2} \dot{y}_{c_i} + \frac{h}{2} \dot{y}_{p_{i+1}} \]

for constant step-size integration.

Replacing the terms on the right by Taylor series expansions about \( t_i \), the accuracy of each estimate is calculated assuming that \( y_{c_{i-1}} \) was computed to second order accuracy (a reasonable assumption with a second order integration method):

\[ y_{p_{i+1}} = y_i + \frac{h}{2} \dot{y}_{i} - \frac{h}{2} \dot{y}_{i} + \frac{h}{2} \ddot{y}_{i} + \frac{h}{2} \dddot{y}_{i} + \text{HOTs} \]

\[ y_{c_{i+1}} = y_i + \frac{h}{2} \ddot{y}_{i} + \frac{h}{2} \dddot{y}_{i} + \frac{h}{2} \ddot{y}_{i} + \frac{h}{2} \dddot{y}_{i} + \text{HOTs} \]  (A.1a)

Subtracting equation (A.1a) from (A.1c) yields the error in the predictor:

\[ y_{i+1} - y_{p_{i+1}} = \frac{h}{2} \dddot{y}_{i} + \text{HOTs} \]  (A.1c)

and similarly, from equations (A.1b) and (A.1a), the error in the corrector is:

\[ y_{i+1} - y_{c_{i+1}} = -\frac{h}{2} \dddot{y}_{i} + \text{HOTs} \]  (A.1d)

Comparing these two error equations, it is clear that a useful estimate of the error in the corrector is given by:

\[ y_{i+1} - y_{c_{i+1}} = \frac{1}{6} (y_{c_{i+1}} - y_{p_{i+1}}) \]  (A.1e)
A.1.1 Second Order Algorithm - Halved Step-size

From equations (B.1.1a) and (B.1b), the modified formula for the second order algorithm when step-size is halved is:

\[
\begin{align*}
\hat{y}_{p_{i+1}} &= y_{c_i} + \frac{5}{4}h \cdot \hat{y}_{c_i} - \frac{1}{4}h \cdot \hat{y}_{c_{i-2}} \\
\hat{y}_{c_{i+1}} &= y_{c_i} + \frac{1}{2} \cdot h \cdot \hat{y}_{c_i} + \frac{1}{2} \cdot h \cdot \hat{y}_{p_{i+1}}.
\end{align*}
\]

The equation for the corrector is unchanged, but the altered coefficients in the predictor equation necessitate recalculation of the error in the predictor estimate, again using Taylor series expansions about \( t_i \):

\[
\begin{align*}
\hat{y}_{p_{i+1}} &= \hat{y}_i + \frac{5}{4}h \cdot \hat{y}_i \\
&\quad - \frac{1}{4}h(\ddot{y}_i - 2h \cdot \ddot{y}_i + 2h^2 \dddot{y}_i + \text{HOTS}) \\
&\quad = \hat{y}_i + h \cdot \ddot{y}_i + \frac{1}{2}h^2 \dddot{y}_i - \frac{1}{2}h^3 \dddot{y}_i + \text{HOTS} . \quad (A.1.1a)
\end{align*}
\]

Subtracting equation (A.1.1a) from (A.a) yields the error in the predictor when step-size is halved:

\[
\begin{align*}
\hat{y}_{i+1} - \hat{y}_{p_{i+1}} &= \frac{5}{4}h \cdot \dddot{y}_i + \text{HOTS} \quad (A.1.1b)
\end{align*}
\]

and the modified estimate of the error in the corrector for halved step-size is given by comparing equations (A.1.1b) and (A.1d):

\[
\begin{align*}
\hat{y}_{i+1} - \hat{y}_{c_{i+1}} &= \frac{1}{4}(\hat{y}_{c_{i+1}} - \hat{y}_{p_{i+1}}) . \quad (A.1.1c)
\end{align*}
\]

A.1.2 Second Order Algorithm - Doubled Step-size

From equations (B.1.2a) and (B.1b), the formula for the second order algorithm when step-size is doubled is:

\[
\begin{align*}
\hat{y}_{p_{i+1}} &= y_{c_i} + 2h \cdot \hat{y}_{c_i} - h \cdot \hat{y}_{c_{i-1}} - \frac{1}{2} \\
\hat{y}_{c_{i+1}} &= y_{c_i} + \frac{1}{2}h \cdot \hat{y}_{c_i} + \frac{1}{2}h \cdot \hat{y}_{p_{i+1}} .
\end{align*}
\]

Again the equation for the corrector is unchanged, and new coefficients in the predictor equation necessitate recalculation of the error in the predictor estimate:
Appendix A  
Discretisation Error Analysis

\[ y_{p_1+1} = y_1 + 2h \dot{y}_1 \]
\[ - h(\ddot{y}_1 - \frac{1}{4}h\dot{y}_1^2 + \frac{1}{6}h^2\dddot{y}_1 + \text{HOTs}) \]
\[ = y_1 + h\dot{y}_1^2 + \frac{1}{2}h^2\dddot{y}_1 - \frac{1}{6}h^3\dddot{y}_1 + \text{HOTs}. \quad (A.1.2a) \]
Subtracting equation (A.1.2a) from (A.a) yields the error in the predictor:

\[ y_{i+1} - y_{p_{i+1}} = \frac{7}{9}h^3\dddot{y}_1 + \text{HOTs} \quad (A.1.2b) \]

which is compared to equation (A.1d) to give an estimate of the error in the corrector when step-size is doubled:

\[ y_{i+1} - y_{c_{i+1}} = \frac{7}{9}(y_{c_{i+1}} - y_{p_{i+1}}) \quad (A.1.2c) \]

A.2 Accuracy of Third Order P-C Algorithm

The analysis proceeds in similar fashion to that for the second order algorithm, with the formula for the third order algorithm given by equations (B.2a-b):

\[ y_{p_{i+1}} = yc_1 + \frac{1}{12}h(23y_{c_1} - 16y_{c_{1-1}} + 5y_{c_{1-2}}) \]
\[ y_{c_{i+1}} = yc_1 + \frac{1}{12}h(8y_{c_1} - y_{c_{1-1}} + 5y_{p_{i+1}}) \]

if step-size is constant.

\( yc_{i-1} \) and \( yc_{i-2} \) will be of third order accuracy, so an extra term is required in the Taylor series expansions of the predictor and corrector equations:

\[ y_{p_{i+1}} = y_1 + 2\frac{1}{12}h\dot{y}_1 \]
\[ - \frac{1}{4}h(\ddot{y}_1 - h\dot{y}_1^2 + \frac{1}{2}h^2\dddot{y}_1 - \frac{1}{6}h^3\dddot{y}_1 + \text{HOTs}) \]
\[ + \frac{1}{12}h(\ddot{y}_1 - 2h\dot{y}_1^2 + 2h^2\dddot{y}_1 - \frac{1}{4}h^3\dddot{y}_1 + \text{HOTs}) \]
\[ = y_1 + h\dot{y}_1^2 + \frac{1}{2}h^2\dddot{y}_1 + \frac{1}{6}h^3\dddot{y}_1 - \frac{1}{4}h^3\dddot{y}_1 + \text{HOTs} \quad (A.2a) \]
Appendix A Discretisation Error Analysis

\[ y_{c_{1+1}} = y_1 + \frac{1}{2} h \dot{y}_1 + \frac{1}{12} h^2 \ddot{y}_1\]

\[ - \frac{1}{12} h (\ddot{y}_1 - h \dot{y}_1 - \frac{1}{2} h^2 \dddot{y}_1 + \frac{1}{6} h^2 \dddot{y}_1 + \text{HOTs}) + \frac{5}{12} h (\ddot{y}_1 + h \dot{y}_1 + \frac{1}{2} h^2 \dddot{y}_1 + \frac{1}{6} h^2 \dddot{y}_1 + \text{HOTs})\]

\[ = y_1 + h \dot{y}_1 + \frac{1}{2} h^2 \dddot{y}_1 + \frac{1}{4} h^3 \ddddot{y}_1 + \frac{1}{12} h^4 \dddddot{y}_1 + \text{HOTs} . \quad (A.2b)\]

The difference between equations (A.2a) and (A.a) gives the error in the third order predictor:

\[ y_{i+1} - y_{P_{i+1}} = \frac{1}{2} h y_{c_{i+1}} + \text{HOTs} \quad (A.2c)\]

and similarly, from equations (A.2b) and (A.a), the error in the corrector is:

\[ y_{i+1} - y_{c_{i+1}} = -\frac{1}{2} h y_{c_{i+1}} + \text{HOTs} . \quad (A.2d)\]

Combination of these two yields a convenient estimate of the error in the third order corrector:

\[ y_{i+1} - y_{c_{i+1}} = \frac{1}{10} (y_{c_{i+1}} - y_{P_{i+1}}) . \quad (A.2e)\]

A.2.1 Third Order Algorithm - Halved Step-size

As explained in Appendix B, with third order P-C integration different formulae must be used for each of the first two steps after step-size is changed.

First Step.

From equations (B.2.1a-b),

\[ y_{P_{i+1}} = y_{c_{i}} + \frac{1}{2} a h (17 y_{c_{i}} - 7 y_{c_{i-2}} + 2 y_{c_{i-4}}) \]

\[ y_{c_{i+1}} = y_{c_{i}} + \frac{1}{3} h (21 y_{c_{i}} - y_{c_{i-2}} + 16 y_{P_{i+1}}) \]

for the first step with the third order algorithm after step-size is halved. Substituting in Taylor series expansions about \( t_{i} \),

- 195 -  Section A.2.1
Appendix A

Discretisation Error Analysis

\[ y_{p1+1} = y_1 + \frac{17}{12} h \dot{y}_1 \]
\[ - \gamma_{12} h (\dot{y}_1 - 2h \ddot{y}_1 + 2h^2 \dddot{y}_1 - \gamma h^3 \dddot{y}_1 + \text{HOTS}) \]
\[ + \gamma h (\dot{y}_1 - 4h \ddot{y}_1 + 8h^2 \dddot{y}_1 - \gamma h^3 \dddot{y}_1 + \text{HOTS}) \]
\[ = y_1 + h \dot{y}_1 + \frac{1}{3} h^2 \ddot{y}_1 + \frac{1}{3} h^3 \dddot{y}_1 - h^3 \dddot{y}_1 + \text{HOTS} \quad (A.2.1a) \]

\[ y_{c1+1} = y_1 + \frac{17}{12} h \dot{y}_1 \]
\[ - \gamma_{12} h (\dot{y}_1 - 2h \ddot{y}_1 + 2h^2 \dddot{y}_1 - \gamma h^3 \dddot{y}_1 + \text{HOTS}) \]
\[ + \gamma h (\dot{y}_1 - 4h \ddot{y}_1 + 8h^2 \dddot{y}_1 - \gamma h^3 \dddot{y}_1 + \text{HOTS}) \]
\[ = y_1 + h \dot{y}_1 + \frac{1}{3} h^2 \ddot{y}_1 + \frac{1}{3} h^3 \dddot{y}_1 - \frac{1}{3} h^3 \dddot{y}_1 + \text{HOTS} \quad (A.2.1b) \]

which can be combined with equation (A.a) to give the error in the predictor:

\[ y_{i+1} - y_{p1+1} = \frac{25}{18} h^3 \dddot{y}_1 + \text{HOTS} \quad (A.2.1c) \]

and in the corrector:

\[ y_{i+1} - y_{c1+1} = -\frac{7}{12} h^3 \dddot{y}_1 + \text{HOTS} . \quad (A.2.1d) \]

Hence an estimate of the error in the corrector for the first step after step-size is halved is:

\[ y_{i+1} - y_{c1+1} = \frac{1}{16} (y_{c1+1} - y_{p1+1}) . \quad (A.2.1e) \]

Second Step.

Equations (B.2.1c) and (B.2b) give the formula for the second step after step-size is halved:

\[ y_{p1+1} = y_{c1} + \frac{1}{36} h (64 \dot{y}_1 - 33 \ddot{y}_1 - 5 \dddot{y}_1 + 6 h \dddot{y}_1 + \text{HOTS}) \]
\[ y_{c1+1} = y_{c1} + \frac{1}{36} h (8 \dot{y}_1 - 5 \ddot{y}_1 - 5 \dddot{y}_1 + 5 \dddot{y}_1 + \text{HOTS}) . \]

Because the same corrector is used as for constant step-size, only the predictor error needs to be recalculated:

\[ y_{p1+1} = y_1 + \frac{17}{12} h \dot{y}_1 \]
\[ - \frac{11}{12} h (\dot{y}_1 - h \ddot{y}_1 + \frac{1}{2} h^2 \dddot{y}_1 - \gamma h^3 \dddot{y}_1 + \text{HOTS}) \]
\[ + \frac{9}{12} h (\dot{y}_1 - 3h \ddot{y}_1 + \frac{3}{2} h^2 \dddot{y}_1 - \gamma h^3 \dddot{y}_1 + \text{HOTS}) \]
\[ = y_1 + h \dot{y}_1 + \frac{1}{3} h^2 \ddot{y}_1 + \frac{1}{3} h^3 \dddot{y}_1 - \frac{17}{36} h \dddot{y}_1 + \text{HOTS} . \quad (A.2.1f) \]
Appendix A

Discretisation Error Analysis

From equation (A.a), the error in the predictor is:

\[ y_{1+1} - y_{p1+1} = \frac{y_i}{12} h^3 \gamma_1 + \text{HOTS} \]  \hspace{1cm} (A.2.1g)

and the estimate of the error in the corrector for the second step after step-size is halved is:

\[ y_{1+1} - y_{c1+1} = \frac{1}{360} (y_{c1+1} - y_{p1+1}) \]  \hspace{1cm} (A.2.1h)

A.2.2 Third Order Algorithm - Doubled Step-size

Local error equations can similarly be derived for the first two steps after step-size is doubled.

First Step.

Equations (B.2.2a-b) give the predictor and corrector equations for the first step after step-size is doubled:

\[ y_{p1+1} = y_i + \frac{1}{12} h (19y \dot{c}_i - 20y \ddot{c}_i - \frac{5}{2} + 7y \dddot{c}_i) \]
\[ y_{c1+1} = y_i + \frac{1}{12} h (30y \ddot{c}_i - 8y \dot{c}_i - \frac{5}{2} + 14y_p) \]

which are analysed using Taylor series expansions as above:

\[ y_{p1+1} = y_i + \frac{1}{12} h \gamma_i 
- \frac{1}{10} h (\dot{y}_i - \frac{1}{2} h \ddot{y}_i + \frac{1}{6} h^2 \gamma_i - \frac{1}{6} h^3 \gamma_i + \text{HOTS}) 
+ \frac{1}{10} h (\dot{y}_i - h \ddot{y}_i + \frac{1}{2} h \ddot{y}_i - \frac{1}{6} h \gamma_i + \text{HOTS}) 
= y_i + h \gamma_i + \frac{1}{2} h \ddot{y}_i + \frac{1}{6} h^2 \gamma_i - \frac{1}{6} h^3 \gamma_i + \text{HOTS} \] \hspace{1cm} (A.2.2a)

\[ y_{c1+1} = y_i + \frac{1}{12} h \gamma_i 
- \frac{1}{2} h (\dot{y}_i - \frac{1}{2} h \ddot{y}_i + \frac{1}{6} h^2 \gamma_i - \frac{1}{6} h^3 \gamma_i + \text{HOTS}) 
+ \frac{1}{12} h (\dot{y}_i + h \ddot{y}_i + \frac{1}{2} h \ddot{y}_i + \frac{1}{6} h \gamma_i + \text{HOTS}) 
= y_i + h \gamma_i + \frac{1}{2} h \ddot{y}_i + \frac{1}{6} h^2 \gamma_i + \frac{1}{2} h^3 \gamma_i + \text{HOTS} \] \hspace{1cm} (A.2.2b)

Subtracting (A.2.2a) from (A.a) gives the error in the predictor:

\[ y_{1+1} - y_{p1+1} = \frac{1}{12} h \gamma_i + \text{HOTS} \] \hspace{1cm} (A.2.2c)

and the error in the corrector, combining (A.2.2b) and (A.a), is:

\[ y_{1+1} - y_{c1+1} = -\frac{1}{12} h \gamma_i + \text{HOTS} \] \hspace{1cm} (A.2.2d)
Thus an estimate of the error in the corrector for the first step after step-size is doubled is:

\[ y_{i+1} - yc_{i+1} = \frac{1}{4}(yc_{i+1} - yp_{i+1}) \]  \hfill (A.2.2e)

Second Step.

Equations (B.2.2c) and (B.2b) give the formula for the second step after step-size is doubled:

\[ yp_{i+1} = yc_i + \frac{1}{4}h(74yc_i - 78yc_{i-1} + 40yc_{i-2}) \]
\[ yc_{i+1} = yo_i + \frac{1}{4}h(8yo_i - 5yo_{i-1} + 5yp_{i+1}) \]

where again the same corrector is used as for constant step-size.

Recalculating the predictor error:

\[ yp_{i+1} = y_1 + \frac{3}{4}h\hat{y}_1 \]
\[ = y_1 + \frac{13}{4}h(\hat{y}_1 - h\hat{y}_1 + \frac{1}{2}h^2\hat{y}_1 - \frac{1}{2}h^2\hat{y}_1 + \text{HOTS}) \]
\[ + \frac{19}{3}h(\hat{y}_1 - \frac{3}{4}h\hat{y}_1 + \frac{3}{8}h^2\hat{y}_1 - \frac{1}{4}h^2\hat{y}_1 + \text{HOTS}) \]
\[ = y_1 + h\hat{y}_1 + \frac{1}{2}h^2\hat{y}_1 + \frac{1}{2}h^2\hat{y}_1 - \frac{19}{3}h^2\hat{y}_1 + \text{HOTS} \]  \hfill (A.2.2f)

leads to:

\[ y_{i+1} - yp_{i+1} = \frac{11}{4}sh^2\hat{y}_1 + \text{HOTS} \]  \hfill (A.2.2g)

and to the following estimate of the error in the corrector for the second step after step-size is doubled:

\[ y_{i+1} - yc_{i+1} = \frac{3}{8}(yc_{i+1} - yp_{i+1}) \]  \hfill (A.2.2h)
B. DERIVATION OF PREDICTOR-CORRECTOR ALGORITHMS

Among the range of P-C algorithms which have been discovered, those based on Adams formulae have become the most popular, and the equations for constant step-size integration are well known. The subject of this Appendix is the derivation of similar equations for use when step-size is not constant: specifically, when step-size is halved or doubled. To introduce the method of derivation, the constant step-size case will be examined first.

The derivation presented here assumes that integration is with respect to time, and that it has proceeded to a point $t_i$, where the true value of the solution will be referred to as $y_i$. This should be distinguished from the approximation generated by the integration algorithm, which will be referred to as $y_c_i$ (c implies Corrector value). A simple representation of the solution points relevant to the derivation of the integration formulae is

\[
\begin{array}{c}
\times \\
\times \\
\times \\
y_{c_{i-2}} & y_{c_{i-1}} & y_{c_i}
\end{array}
\]

where the horizontal separation represents step-size, and the amplitude of the solution is not shown since it is unnecessary for the discussion. The task is now to compute $y_{c_{i+1}}$.

The second order P-C algorithm chosen, which is suitable for VSSDDA applications, utilises a second order Adams prediction followed by a Trapezoidal correction. The Adams predictor computes an approximation to the solution at $t_{i+1}$ using $y_{c_i}$ and the derivatives of $y_{c_{i-1}}$ and $y_{c_i}$. Referring to this approximation as $y_{p_{i+1}}$ (p implies
Predictor value), this intermediate stage in the calculations can be illustrated by

\[ \begin{array}{ccc}
   x & x & x \\
   y_{c_i-1} & y_{c_i} & y_{p_i+1} \\
\end{array} \]

The Trapezoidal corrector combines \( y_{c_i} \) with its derivative and with the derivative of \( y_{p_i+1} \) to compute \( y_{c_i+1} \): a more accurate approximation to \( y_{i+1} \). The differences between \( y_{i+1}, y_{c_i+1} \) and \( y_{p_i+1} \) are generally small, and being differences in amplitude are not of interest here.

The corresponding third order algorithm requires one extra previous solution value \( y_{c_i-2} \) to achieve its greater accuracy. Thus the third order Adams predictor computes \( y_{p_i+1} \) from \( y_{c_i} \) and the derivatives of \( y_{c_i-2}, y_{c_i-1}, \) and \( y_{i} \), then \( y_{c_i+1} \) is generated via third order correction from \( y_{c_i} \) and the derivatives of \( y_{c_i-1}, y_{c_i}, \) and \( y_{p_i+1} \).

The need for non-standard formulae when step-size is changed arises from the subscripts attached to the previous solution points, \( y_{c_i-1} \) and \( y_{c_i-2} \). Although there is no actual reference to \( h \), the subscripts are used to refer to the solution at a time measured in units of step-size. Hence if step-size is changed at \( t_i \), the subscripts should be changed accordingly. However, whether step-size is constant or changing, formulae for \( y_{p_i+1} \) and \( y_{c_i+1} \) can be derived from Taylor series expansions about \( t_i \), in particular the equalities given in equation (A.a-g).
B.1 Second Order Predictor-Corrector Algorithm

In order to derive the second order Adams formula for constant step-size integration, it is firstly written in the form
\[ y_{p_i+1} = y_{c_i} + a \cdot y_{c_i} + b \cdot y_{c_i-1} \]
then \( a \) and \( b \) are evaluated by equating Taylor series expansions of both sides, assuming that estimates are of second order accuracy:

L.H.S. = \( y_i + h \cdot \dot{y}_i + \frac{1}{2} h^2 \dot{y}^2_i + \text{HOTs} \)
R.H.S. = \( y_i + a \cdot \dot{y}_i 
+ b \cdot \dot{y}_1 - b \cdot h \cdot \dot{y}_i + \text{HOTs} \)
\[ \text{i.e. } a + b = h, -b \cdot h = \frac{1}{2} h^2 \]
\[ \text{i.e. } a = \frac{3}{2} h, b = -\frac{1}{2} h . \]
Substitution of these values for \( a \) and \( b \) leads to the well known second order Adams formula:
\[ y_{p_i+1} = y_{c_i} + \frac{3}{2} h \cdot \dot{y}_{c_i-1} - \frac{1}{2} h \cdot \dot{y}_{c_i-1} \]
\[ = y_{c_i} + h \cdot \dot{y}_{c_i} + \frac{1}{2} h (\dot{y}_{c_i} - \dot{y}_{c_i-1}) \]  \hspace{1cm} (B.1a)

Similarly the second order Trapezoidal correction formula can be derived, starting from the form:
\[ y_{c_i+1} = y_{c_i} + a \cdot \dot{y}_{c_i} + b \cdot \dot{y}_{p_i+1} \]
and applying Taylor series expansions to both sides:
L.H.S. = \( y_i + h \cdot \dot{y}_i + \frac{1}{2} h^2 \dot{y}^2_i + \text{HOTs} \)
R.H.S. = \( y_i + a \cdot \dot{y}_i 
+ b \cdot \dot{y}_1 + b \cdot h \cdot \dot{y}_i + \text{HOTs} \)
\[ \text{i.e. } a + b = h, b \cdot h = \frac{1}{2} h^2 \]
\[ \text{i.e. } a = \frac{1}{2} h, b = \frac{1}{4} h . \]
Substituting these values for \( a \) and \( b \) leads to the standard second order Trapezoidal correction formula:
\[ y_{c_i+1} = y_{c_i} + \frac{1}{2} h \cdot \dot{y}_{c_i} + \frac{1}{2} h \cdot \dot{y}_{p_i+1} . \]  \hspace{1cm} (B.1b)
B.1.1 Second Order - Halved Step-size

When step-size is halved, a simple representation of the situation after one more step of integration is

\[
\begin{align*}
\times & \quad \times & \quad \times \\
y_{c_{i-1}} & \quad y_{c_i} & \quad y_{c_{i+1}}
\end{align*}
\]

where the change in step-size is indicated by the decrease in horizontal separation of the points.

Because the subscripts attached to the solution points contain an implicit reference to \( h \), any change in \( h \) precipitates a corresponding change in the subscripts of the previous points used in calculating the next point. So when step-size is halved, \( y_{c_{i-1}} \) needs to be renamed \( y_{c_{i-2}} \) because, in terms of the new step-size, it represents the solution two steps previously. This means that \( y_{p_{i+1}} \) is to be calculated using \( y_{c_{i-2}} \) and \( y_{c_i} \), so a modified algorithm must be derived, starting with the formula

\[
y_{p_{i+1}} = y_{c_{i}} + a \cdot y_{c_{i-1}} + b \cdot y_{c_{i-2}}
\]

where \( a \) and \( b \) need to be re-evaluated by equating Taylor series expansions of both sides:

\[
\begin{align*}
\text{L.H.S.} &= y_i + h \cdot \dot{y}_i + \frac{1}{2} h^2 \ddot{y}_i + \text{HOTS} \\
\text{R.H.S.} &= y_i + a \cdot \dot{y}_i \\
&\quad + b \cdot \dot{y}_i - 2b \cdot h \cdot \ddot{y}_i + \text{HOTS}
\end{align*}
\]

i.e. \( a + b = h \), \(-2b \cdot h = \frac{1}{2} h^2 \)

i.e. \( a = \frac{5}{4} h \), \( b = -\frac{1}{4} h \).

This leads to a special second order Adams formula for use when step-size is halved:
Appendix B  

Derivation of P-C Algorithms

\[ y_{p_{i+1}} = y_{c_i} + \frac{3}{2} h x y_{c_i} - \frac{1}{2} h x y_{c_{i-2}} = y_{c_i} + h x y_{c_i} + \frac{1}{2} h (y_{c_i} - y_{c_{i-2}}) \quad (B.1.1a) \]

The two points available for calculation of the corrector are \( y_{c_i} \) and \( y_{p_{i+1}} \) as for constant step-size, so the corrector formula is unchanged for halved step-size - unlike the predictor formula.

B.1.2 Second Order - Doubled Step-size

Two options are open for computing the first step after doubling step-size. Including two past integration points, the picture after one step with doubled step-size is

\[
\begin{array}{cccc}
\times & \times & \times & \\
y_{c_{i-2}} & y_{c_{i-1}} & y_{c_i} & y_{c_{i+1}}
\end{array}
\]

from which it is clear that \( y_{c_{i-1}} \) must be renamed \( y_{c_{i-\frac{1}{2}}} \), and \( y_{c_{i-2}} \) is really \( y_{c_{i-1}} \), in terms of the new step-size. Obviously, \( y_{c_{i-2}} \) is not required for second order integration, but, if this extra previous point is stored, then, when step-size is doubled, a true \( y_{c_{i-1}} \) will be available, permitting the same formula to be applied as for constant step-size whenever step-size is doubled. Otherwise, a new formula can be derived as is necessary for halved step-size. If this alternative is chosen, the two points available will be \( y_{c_i} \) and \( y_{c_{i-\frac{1}{2}}} \):

\[ y_{p_{i+1}} = y_{c_i} + a x y_{c_i} + b x y_{c_{i-\frac{1}{2}}} \]

and \( a \) and \( b \) need to be re-evaluated again:
Appendix B

Derivation of P-C Algorithms

\[ L.H.S. = y_i + h\dot{y}_i + \frac{1}{2}h^2\ddot{y}_i + \text{HOTs} \]
\[ R.H.S. = y_i + a\dot{y}_i + b\dot{y}_i - \frac{1}{2}b\times h\times \dot{y}_i + \text{HOTs} \]

i.e. \( a + b = h, -\frac{1}{2}b = \frac{1}{2}h^2 \)

Thus when step-size is doubled, the modified second order Adams predictor is:

\[ y_{p_{i+1}} = yc_1 + 2hx\dot{c}_1 - h\times \ddot{c}_1 - \frac{1}{2} \]
\[ = yc_1 + h\times \dot{c}_1 + h(\ddot{c}_1 - \ddot{c}_1 - \frac{1}{2}) \]  \hspace{1cm} (B.1.2a)

Again the two points available for calculation of the corrector are \( yc_1 \) and \( yp_{i+1} \) as for constant step-size, so the corrector formula is also unchanged for doubled step-size.

\[ a + b + c = h \]

Taking out \( \frac{1}{12}h \) as a common factor, substitution of these values for

B.2 Third Order Predictor-Corrector Algorithm

Using the same approach as for the second order formula, the third order Adams formula for constant step-size integration can be derived, starting from the form:

\[ y_{p_{i+1}} = yc_1 + a\times y_{c_1} + b\times y_{c_1 - 1} + c\times y_{c_1 - 2} \]

and equating Taylor series expansions of both sides to evaluate \( a, b \) and \( c \):

\[ L.H.S. = y_i + h\dot{y}_i + \frac{1}{2}h^2\ddot{y}_i + \frac{1}{6}h^3\dddot{y}_i + \text{HOTs} \]
\[ R.H.S. = y_i + a\dot{y}_i + b\dot{y}_i + c\dot{y}_i - 2c\times h\times \ddot{y}_i + 2c\times h^2\dddot{y}_i + \text{HOTs} \]

i.e. \( a + b + c = h, b + 2c = -\frac{1}{2}h, \frac{1}{2}b + 2c = \frac{1}{6}h \)

Taking out \( \frac{1}{12}h \) as a common factor, substitution of these values for
Appendix B  Derivation of P-C Algorithms

a, b and c leads to the following third order Adams formula:

\[ y_{p_{i+1}} = y_{c_i} + \frac{1}{12} h (23y_{c_{i}}^2 - 16y_{c_{i-1}} + 5y_{c_{i-2}}) \]  

(B.2a)

Similarly the third order correction formula can be derived, starting with the form:

\[ y_{c_{i+1}} = y_{c_i} + a \cdot y_{c_{i}} + b \cdot y_{c_{i-1}} + c \cdot y_{p_{i+1}} \]

then applying Taylor series expansions to evaluate a, b and c:

L.H.S. = \[ y_i + \frac{1}{2} h \cdot y_i' + \frac{1}{6} h^2 \cdot y_i'' + \text{HOTS} \]
R.H.S. = \[ y_i + a \cdot y_i' + b \cdot y_i' - \frac{1}{2} b \cdot h \cdot y_i'' + \text{HOTS} \]
+ \[ c \cdot y_i' + c \cdot h \cdot y_i'' + \frac{1}{2} c \cdot h^2 \cdot y_i'' + \text{HOTS} \]

i.e. \( a + b + c = h \), \( b - c = -\frac{1}{2} h \), \( b + c = \frac{1}{4} h \),

i.e. \( a = \frac{1}{4} h \), \( b = -\frac{1}{12} h \), \( c = \frac{5}{12} h \).

The common factor is \( \frac{1}{12} h \) again, so the third order corrector can be written in the form:

\[ y_{c_{i+1}} = y_{c_i} + \frac{1}{12} h (8y_{c_{i}}^2 - y_{c_{i-1}} + 5y_{p_{i+1}}) \]  

(B.2b)

B.2.1 Third Order - Halved Step-size

Because the third order algorithm uses two previous points in computing the next one, a separate formula is required for each of the first two steps after step-size is halved. After one step with halved step-size, the situation before subscripts are corrected is

\[ x \ x \ x \ x \ x \ y_{c_{i-2}} \ y_{c_{i-1}} \ y_{c_{i}} \ y_{c_{i+1}} \]

Renaming \( y_{c_{i-2}} \) and \( y_{c_{i-1}} \) according to the new step-size, it can be seen that in order to compute \( y_{c_{i+1}} \), the points available are \( y_{c_{i-4}} \), \( y_{c_{i-2}} \) and \( y_{c_{i}} \) for the predictor, and \( y_{c_{i-2}} \), \( y_{c_{i}} \) and \( y_{p_{i+1}} \) for the
Appendix B  Derivation of P-C Algorithms

corrector. After another step at the new step-size, the true picture (with modified subscripts) is

\[
\begin{align*}
\times & \times \times \times \times \\
y_{c_{i-5}} & y_{c_{i-3}} & y_{c_{i-1}} & y_{c_{i}} & y_{c_{i+1}}
\end{align*}
\]

which indicates that the predictor can be computed from \(y_{c_{i-3}}, y_{c_{i-1}}, \) and \(y_{c_{i}},\) giving the corrector the same three points as for constant step-size.

**First Step.**

To calculate the Adams formula for the first step after step-size is halved, the appropriate subscripts are attached to the past integration points:

\[
y_{p_{i+1}} = y_{c_{i}} + a \cdot y_{c_{i-1}} + b \cdot y_{c_{i-2}} + c \cdot y_{c_{i-4}}
\]

then \(a, b,\) and \(c\) are evaluated, as usual, by equating Taylor series expansions of both sides:

L.H.S. = \(y_{i} + h \cdot \dot{y}_{i} + \frac{1}{2} h^{2} \ddot{y}_{i} + \frac{1}{6} h^{3} \dddot{y}_{i} + \text{HOTS}\)

R.H.S. = \(y_{i} + a \cdot \dot{y}_{i}\)
\[+ b \cdot \dot{y}_{i} - 2b \cdot h \cdot \ddot{y}_{i} + 2b \cdot h^{2} \dddot{y}_{i} + \text{HOTS}\]
\[+ c \cdot \ddot{y}_{i} - 4c \cdot h \cdot \dddot{y}_{i} + 8c \cdot h^{2} \dddddot{y}_{i} + \text{HOTS}\]

i.e. \(a + b + c = h, 2b + 4c = \frac{1}{2} h, 2b + 8c = \frac{1}{6} h\)

i.e. \(a = \frac{1}{12} h, b = -\frac{1}{12} h, c = \frac{1}{6} h.\)

Again, \(\frac{1}{12} h\) is a common factor, hence the predictor formula can be written:

\[
y_{p_{i+1}} = y_{c_{i}} + \frac{1}{12} h (17y_{c_{i}} - 7y_{c_{i-2}} + 2y_{c_{i-4}}).
\]  \(\text{(B.2.1a)}\)

Derivation of the associated third order corrector formula proceeds similarly, starting from:
Appendix B

Derivation of P-C Algorithms

\[ yc_{i+1} = yc_i + a \cdot yc_i + b \cdot yc_{i-2} + c \cdot yc_{i+1} \]

and then applying Taylor series expansions:

L.H.S. = \( y_i + h \cdot y_i' + \frac{1}{2}h^2y_i'' + \frac{1}{6}h^3y_i''' + \text{HOTS} \)

R.H.S. = \( y_i + a \cdot y_i' + \frac{1}{2}h \cdot y_i'' + \frac{1}{6}h^2y_i''' + \text{HOTS} \)

\[ i.e. a + b + c = h \]

\( b = \frac{1}{2}h \)

\( c = \frac{1}{6}h \)

Second Step.

To calculate the predictor formula for the second step after halving step-size, start with the correct subscripts:

\[ yp_{i+1} = yc_i + a \cdot y_i' + b \cdot y_{i-1} + c \cdot y_{i-3} \]

evaluate a, b and c:

L.H.S. = \( y_i + h \cdot y_i' + \frac{1}{2}h^2y_i'' + \frac{1}{6}h^3y_i''' + \text{HOTS} \)

R.H.S. = \( y_i + a \cdot y_i' + \frac{1}{2}h \cdot y_i'' + \frac{1}{6}h^2y_i''' + \text{HOTS} \)

\[ i.e. a + b + c = h \]

\( b + 3c = -\frac{1}{2}h \)

\( b + 9c = \frac{1}{2}h \)

\( i.e. a = \frac{16}{3}h \)

\( b = -\frac{11}{12}h \)

\( c = \frac{7}{36}h \)

and take out \( \frac{1}{36}h \) as the largest common factor:

\[ yp_{i+1} = yc_i + \frac{1}{36}h(64y_i' - 33y_{i-1} + 5y_{i-3}) \]  
(B.2.1c)

As noted above, the corrector formula for the second step after halving of the step-size is the same as for constant step-size.
B.2.2 Third Order - Doubled Step-size

As for the second order algorithm, storing extra previous points could eliminate the need for special formulae when step-size is doubled, but since the third order algorithm uses two previous points, two extra (i.e., \( y_{c_{i-4}} \) and \( y_{c_{i-3}} \)) would need to be stored. Alternatively, as was shown for halved step-size, a separate formula is required for each of the first two steps after step-size is doubled. Renaming subscripts as required, one step at doubled step-size aims to achieve

\[
\begin{array}{cccc}
\times & \times & \times & \times \\
y_{c_{i-1}} & y_{c_{i-\frac{1}{2}}} & y_{c_i} & y_{c_{i+1}}
\end{array}
\]

indicating that the points available are \( y_{c_{i-1}} \), \( y_{c_{i-\frac{1}{2}}} \) and \( y_{c_i} \) for the predictor, and \( y_{c_{i-\frac{1}{2}}} \), \( y_{c_i} \) and \( y_{p_{i+1}} \) for the corrector. The second step will lead to

\[
\begin{array}{cccccc}
\times & \times & \times & \times & \times \\
y_{c_{i-2}} & y_{c_{i-\frac{1}{2}}} & y_{c_{i-1}} & y_{c_i} & y_{c_{i+1}}
\end{array}
\]

using \( y_{c_{i-\frac{1}{2}}} \), \( y_{c_{i-1}} \) and \( y_{c_i} \) for the predictor, and, for the corrector, the same three points as for constant step-size.
Appendix B  Derivation of P-C Algorithms

First Step.

The initial form of the predictor equation, with appropriate subscripts, is:

\[ y_{p1+1} = yc_1 + ax'y_1 + bxy'c_1 - \frac{1}{2} + cxy'c_{1-1} \]

where a, b and c are evaluated by equating Taylor series expansions of both sides:

L.H.S. = \( y_1 + hxy_1 + \frac{1}{2}h^2y_1' + \frac{1}{6}h^3y_1'' + \text{HOTS} \)

R.H.S. = \( y_1 + ax'y_1 \\
+ bxy_1 - \frac{1}{2}bxy^2y_1 + \frac{1}{6}bxy^2y_1' + \text{HOTS} \\
+ cxy_1 - cxy^2y_1 + \frac{1}{2}cxy^2y_1' + \text{HOTS} \)

i.e. \( a + b + c = h, \frac{1}{2}b + c = \frac{-1}{2}h, \frac{1}{6}b + c = \frac{1}{3}h \)

i.e. \( a = \frac{19}{6}h, b = \frac{-19}{6}h, c = \frac{7}{6}h \).

This leads to an appropriate third order Adams formula for the first step after step-size is doubled:

\[ y_{p1+1} = yc_1 + \frac{1}{6}h(19yc_1 - 20y'c_1 - \frac{1}{2} + 7y''c_{1-1}) \]  \hspace{1cm} (B.2.2a)

The corresponding third order correction formula is similarly derived:

\[ yc_{i+1} = yc_1 + ax'y_1 + bxy'c_1 - \frac{1}{2} + cxy'p_{i+1} \]

using Taylor series expansions to determine a, b and c:

L.H.S. = \( y_1 + hxy_1 + \frac{1}{2}h^2y_1' + \frac{1}{6}h^3y_1'' + \text{HOTS} \)

R.H.S. = \( y_1 + ax'y_1 \\
+ bxy_1 - \frac{1}{2}bxy^2y_1 + \frac{1}{6}bxy^2y_1' + \text{HOTS} \\
+ cxy_1 - cxy^2y_1 + \frac{1}{2}cxy^2y_1' + \text{HOTS} \)

i.e. \( a + b + c = h, \frac{1}{2}b - c = \frac{-1}{2}h, \frac{1}{6}b + c = \frac{1}{3}h \)

i.e. \( a = \frac{1}{6}h, b = \frac{-1}{6}h, c = \frac{7}{6}h \)

then substituting these values in:

\[ yc_{i+1} = yc_1 + \frac{1}{6}h(30yc_1 - 8y'c_1 - \frac{1}{2} + 14y''p_{i+1}) \]  \hspace{1cm} (B.2.2b)
Second Step.

To calculate the formula for the second step after doubling step-size, start with:

\[ y_{p_{i+1}} = y_{c_{i}} + a \cdot y_{c_{i}} + b \cdot y_{c_{i-1}} + c \cdot y_{c_{i-1} - \frac{h}{2}} \]

evaluate \( a, b \) and \( c \) as usual:

\[
\begin{align*}
\text{L.H.S.} &= y_{i} + h \cdot y_{i} + \frac{1}{2} h \cdot \ddot{y}_{i} + \frac{1}{6} h \cdot \dddot{y}_{i} + \text{HOTS} \\
\text{R.H.S.} &= y_{i} + a \cdot \dot{y}_{i} \\
&+ b \cdot x_{1} - b \cdot x_{1} + \frac{1}{2} b \cdot x_{1} + \text{HOTS} \\
&+ c \cdot \dot{y}_{i} - \frac{1}{6} c \cdot x_{1} + \frac{9}{6} c \cdot x_{1} + \text{HOTS} \\
\text{i.e. } a + b + c &= h , b + \frac{3}{4} c = -\frac{1}{2} h , \frac{1}{2} b + \frac{9}{6} c = \frac{1}{6} h \\
i.e. a &= \frac{37}{6} h , b = -\frac{13}{6} h , c = \frac{10}{6} h
\end{align*}
\]

and substitute the values in:

\[ y_{p_{i+1}} = y_{c_{i}} + \frac{1}{6} h (74 y_{c_{i}} - 78 y_{c_{i-1}} + 40 y_{c_{i-1} - \frac{h}{2}}) \]  
(B.2.2c)

As noted above, the corrector formula for the second step after doubling of the step-size is the same as for constant step-size.
C. **TRUNCATION ERROR ANALYSIS**

The error analysis applied in Chapter 2 to the register transfers for Euler integration will now be applied to the two sets of register transfers prescribed in Chapters 4 and 6 for second order P-C integration. The simple DE: $\dot{y} = Ay$, employed in Chapter 2, will again be used to avoid unnecessary complication of the analysis.

C.1 **Register Transfers for Linear Equations**

Combining equation (4.3a-e) with the above DE yields a set of register transfers to implement constant step-size integration:

- $(YC + \frac{1}{2}h\Delta YC) + PIR + \Delta PZ, PIR$ (C.1a)
- $A*\Delta PZ + PMR + \Delta YP, PMR$ (C.1b)
- $(YC + \frac{1}{2}h\Delta YP) + CIR + \Delta CZ, CIR$ (C.1c)
- $A*\Delta CZ + CMR + \Delta YC, CMR$ (C.1d)
- $YC + h\Delta YC + YC$ (C.1e)

Only one $\Delta Z$ register is actually necessary, but since this register would contain two separate quantities during each integration iteration, two register names are used to facilitate reference to these quantities. The register names used in equation (C.1a-e) are chosen to indicate their purpose as follows:

- PIR = predictor integrator residue register;
- PMR = predictor multiplier residue register;
- CIR = corrector integrator residue register;
- CMR = corrector multiplier residue register;
- $\Delta PZ$ = truncated version of $YC$ for the predictor;
- $\Delta CZ$ = truncated version of $YC$ for the corrector.
Appendix C  Truncation Error Analysis

A set of equations defining the change in the contents of the four residue registers at each iteration is necessary for the error analysis:

\[ \Delta \text{pir}_i = \text{pir}_i - \text{pir}_{i-1} \quad (C.1f) \]
\[ \Delta \text{pmr}_i = \text{pmr}_i - \text{pmr}_{i-1} \quad (C.1g) \]
\[ \Delta \text{cir}_i = \text{cir}_i - \text{cir}_{i-1} \quad (C.1h) \]
\[ \Delta \text{cmr}_i = \text{cmr}_i - \text{cmr}_{i-1} \quad (C.1i) \]

as are definitions for the truncation error introduced by the finite lengths of the YC and ΔYC registers, similar to equation (2.2.1a-c):

\[ y_1 - y_{c1} = E_1 2^{-M} \quad (C.1j) \]
\[ \Delta y_{i-1} - \Delta y_{c1} = \Delta E_{i-1} 2^{-m} \quad (C.1k) \]
\[ E_{i+1} = E_i + \Delta E_i \quad (C.1l) \]

An expression for the truncation error after \( i \) iterations can now be derived by combining equations (C.1a-l) in a manner similar to that of Section 2.2.2:

\[ \Delta \text{pz}_i = y_{c1} + \frac{1}{2} \Delta y_{c1} 2^{-k} - \Delta \text{pir}_i 2^{-m} \]
\[ = y_1 - E_1 2^{-M} + \frac{1}{2} (\Delta y_{i-1} - \Delta E_{i-1} 2^{-m}) 2^{-k} - \Delta \text{pir}_i 2^{-m} \]
\[ = y_1 + \frac{1}{2} \Delta y_{i-1} 2^{-k} - (E_1 + \frac{1}{2} \Delta E_{i-1}) 2^{-M} - \Delta \text{pir}_i 2^{-m} \]

\[ \Delta \text{pp}_i = A \Delta \text{pz}_i - \Delta \text{pmr}_i 2^{-m} \]
\[ = A \times y_1 + \frac{1}{2} A \Delta y_{i-1} 2^{-k} - (A \times E_1 + \frac{1}{2} A \Delta E_{i-1}) 2^{-M} \\
- (A \Delta \text{pir}_i + \Delta \text{pmr}_i) 2^{-m} \]

\[ \Delta \text{cz}_i = y_{c1} + \frac{1}{2} \Delta \text{pp}_i 2^{-k} - \Delta \text{cir}_i 2^{-m} \]
\[ = y_1 + \frac{1}{2} (A \times y_1 + \frac{1}{2} A \Delta y_{i-1} 2^{-k}) 2^{-k} \\
- (E_1 + \frac{1}{2} (A \times E_1 + \frac{1}{2} A \Delta E_{i-1}) 2^{-k}) 2^{-M} \\
- (\frac{1}{2} (A \Delta \text{pir}_i + \Delta \text{pmr}_i) 2^{-k} + \Delta \text{cir}_i) 2^{-m} \]
At this point, a definition of the composite change in residue registers is useful:

\[ R_i = \frac{1}{2} A(\Delta \text{pir}_i + \Delta \text{pmr}_i)2^{-k} + A\Delta \text{cir}_i + \Delta \text{cmr}_i \]  

(C.1m)

which leads to:

\[ \Delta E_i = (A \times E_i + \frac{1}{2} A^2 E_i 2^{-k} + \frac{1}{4} A^2 \Delta E_{i-1} 2^{-k})2^{-k} + R_i \]

\[ = E_{i+1} - E_i \]

hence:

\[ E_{i+1} = E_i + (A \times E_i + \frac{1}{2} A^2 E_i 2^{-k} + \frac{1}{4} A^2 \Delta E_{i-1} 2^{-k})2^{-k} + R_i \]

\[ = E_i + (A \times E_i + \frac{1}{2} A^2 E_i 2^{-k} + \frac{1}{4} A^2 (E_i - E_{i-1}) 2^{-k} 2^{-k} + R_i \]

\[ = (1 + \frac{1}{4} A^2 2^{-2k})E_i - (\frac{1}{4} A^2 2^{-2k})E_{i-1} + R_i \]

This recursive equation for \( E_{i+1} \) will now be substituted into itself, but firstly, in order to simplify manipulation, let:

\[ V = 1 + A \times 2^{-k} + \frac{1}{4} A^2 2^{-2k} ; \quad \text{(C.1n)} \]

\[ W = -\frac{1}{4} A^2 2^{-2k} ; \quad \text{(C.1o)} \]

then

\[ E_{i+1} = V \times E_i + W \times E_{i-1} + R_i \]

\[ = V(V \times E_{i-1} + W \times E_{i-2} + R_{i-1}) + W \times E_{i-1} + R_i \]

\[ = (V^2 + W)E_{i-1} + (V \times W)E_{i-2} + R_i + V \times R_{i-1} \]
Appendix C  
Truncation Error Analysis

\[
\begin{align*}
(V_3 + 2VxW)E_{i-2} &+ (V^2W + W^2)E_{i-3} \\
+ R_1 + VxR_{i-1} &+ (V^2 + W)R_{i-2} \\
= (V'' + 3V^2W + W^2)E_{i-3} &+ (V^3W + 2VxW^2)E_{i-4} \\
+ R_1 + VxR_{i-1} &+ (V^2 + W)R_{i-2} + (V^3 + 2VxW)R_{i-3} \\
= &. \\
&. \\
= (V^{i+1} + iV^{i-1}W + HOTs)E_o &+ (V^iW + HOTs)E_{-1} \\
+ R_1 + VxR_{i-1} &+ (V^2 + W)R_{i-2} + (V^3 + 2VxW)R_{i-3} \\
+ .... &+ (V^i + (i-1)V^{i-2}W + HOTs)R_0
\end{align*}
\]

This form can be reorganised using

\[
E_{-1} = 0,
\]

\[
V^n = 1 + nA\times 2^{-k} + HOTs, \quad \text{[from (C.1n)]}
\]

\[
W = HOTs \quad \text{[from (C.1o)]}
\]

to give

\[
E_{i+1} = (1 + A(i+1)2^{-k})E_o + \frac{i}{j} R_j + A \sum_{j=0}^{i-1} \frac{1}{n=0} (\sum R_n) 2^{-k} + HOTs
\]

\[
= E_o + \sum_{j=0}^{i} R_j + A[(i+1)E_o + \sum_{j=0}^{i-1} \frac{1}{n=0} (\sum R_n) 2^{-k} + HOTs]
\]

which is the same as equation (2.2.2c).

C.1.1 Second Order Suppression

Substituting equation (C.1m) into equation (C.1p) and manipulating the result indicates that, with second order P-C integration, the accumulation of second order truncation error can be suppressed by the same initialisation of residue registers as has been shown to be appropriate for Euler integration. The result is:
Appendix C  Truncation Error Analysis

\[
E_{i+1} = E_0 + \sum_{j=0}^{i} \left( \frac{1}{2} A(\Delta \text{pir}_j + \Delta \text{pmr}_j)2^{-k} + A \Delta \text{circ}_j + \Delta \text{cmr}_j \right) \\
+ A((i+1)E_0 + \sum_{j=0}^{i-1} \left( \sum_{n=0}^{j} \left( \sum_{m=0}^{n} \left( \frac{1}{2} A(\Delta \text{pir}_n + \Delta \text{pmr}_n)2^{-k} + A \Delta \text{circ}_n + \Delta \text{cmr}_n \right) \right) \right)2^{-k} \\
+ \text{HOTS}
\]

\[
= E_0 + \sum_{j=0}^{i} (A \Delta \text{circ}_j + \Delta \text{cmr}_j) \\
+ A((i+1)E_0 + \frac{1}{2} \sum_{j=0}^{i-1} (A \Delta \text{pir}_j + \Delta \text{pmr}_j) + \sum_{j=0}^{i-1} \left( \sum_{n=0}^{j} \left( A \Delta \text{circ}_n + \Delta \text{cmr}_n \right) \right)2^{-k} \\
+ \text{HOTS}
\]

The first line represents the second order truncation error, and, by the similarity of the first line of equation (2.2.3a), it can be seen that if all residue registers are initialised to \( \frac{1}{2} \), suppression of the accumulation of second order error is achieved.

C.1.2 Third Order Suppression

Modified register transfers to implement suppression of third order truncation error lead to modified definitions of the change in residue register contents:

\[
\Delta \text{pir}_i = \text{pir}_i - 2 \times \text{pir}_{i-1} + \text{pir}_{i-2} \\
\Delta \text{pmr}_i = \text{pmr}_i - 2 \times \text{pmr}_{i-1} + \text{pmr}_{i-2} \\
\Delta \text{circ}_i = \text{circ}_i - 2 \times \text{circ}_{i-1} + \text{circ}_{i-2} \\
\Delta \text{cmr}_i = \text{cmr}_i - 2 \times \text{cmr}_{i-1} + \text{cmr}_{i-2}
\]

which, when combined with equations (C.1m) and (C.1p), leads to an
error equation indicating that third order truncation error accumulation can be suppressed:

\[ E_{i+1} = E_0 + \sum_{j=0}^{i} \left( \frac{1}{2} A (\Delta \text{pir}_j + \Delta \text{pmr}_j) 2^{-k} + A \Delta \text{cir}_j + A \Delta \text{cmr}_j \right) \]
\[ + A((i+1)E_0) \]
\[ + \sum_{j=0}^{i-1} \left( \sum_{n=0}^{j} \left( \frac{1}{2} A (\Delta \text{pir}_n + \Delta \text{pmr}_n) 2^{-k} + A \Delta \text{cir}_n + A \Delta \text{cmr}_n \right) \right) 2^{-k} \]
\[ + \text{HOTS} \]

\[ = E_0 + \sum_{j=0}^{i} \left( A \Delta \text{cir}_j + A \Delta \text{cmr}_j \right) \]
\[ + A((i+1)E_0) \]
\[ + \frac{1}{2} \sum_{j=0}^{i} \left( A (\Delta \text{pir}_j + \Delta \text{pmr}_j) + \sum_{n=0}^{j-1} \left( A \Delta \text{cir}_n + A \Delta \text{cmr}_n \right) \right) 2^{-k} \]
\[ + \text{HOTS} \]

\[ = E_0 + \sum_{j=0}^{i} \left( A (\text{cir}_j - 2 \times \text{cir}_{j-1} + \text{cir}_{j-2}) + (\text{cmr}_j - 2 \times \text{cmr}_{j-1} + \text{cmr}_{j-2}) \right) \]
\[ + A((i+1)E_0) \]
\[ + \frac{1}{2} \sum_{j=0}^{i} \left( A (\text{pir}_j - 2 \times \text{pir}_{j-1} + \text{pir}_{j-2}) + \text{pmr}_j - 2 \times \text{pmr}_{j-1} + \text{pmr}_{j-2} \right) \]
\[ + \sum_{j=0}^{i-1} \left( \sum_{n=0}^{j} \left( A (\text{cir}_n - 2 \times \text{cir}_{n-1} + \text{cir}_{n-2}) \right) \right. \]
\[ \left. \quad + \text{cmr}_n - 2 \times \text{cmr}_{n-1} + \text{cmr}_{n-2} \right) ) 2^{-k} \]
\[ + \text{HOTS} \]

\[ = E_0 + A(\text{cir}_i - \text{cir}_{i-1} + \text{cir}_{-1}) + (\text{cmr}_i - \text{cmr}_{i-1} + \text{cmr}_{-1}) \]
\[ + A((i+1)E_0) \]
\[ + \frac{1}{2} (A (\text{pir}_i - \text{pir}_{i-1} - \text{pir}_{-1}) + \text{pmr}_i - \text{pmr}_{i-1} - \text{pmr}_{-1}) \]
\[ + \sum_{j=0}^{i-1} \left( A (\text{cir}_j - \text{cir}_{j-1} + \text{cir}_{-1}) + \text{cmr}_j - \text{cmr}_{j-1} + \text{cmr}_{-1} \right) 2^{-k} \]
\[ + \text{HOTS} \]
Appendix C Truncation Error Analysis

\[ E_0 + A(\text{cir}_i - \text{cir}_{i-1}) + (\text{cmr}_i - \text{cmr}_{i-1}) - (A \times \text{cir}_i + \text{cmr}_i) \]
\[ + A((i+1)E_0 + \frac{1}{2}(A(\text{pir}_i - \text{pir}_{i-1}) + (\text{pmr}_i - \text{pmr}_{i-1}) - (A \times \text{pir}_i + \text{pmr}_i)) \]
\[ + \sum_{j=0}^{i-1} \frac{1}{2}(A(\text{cir}_j - \text{cir}_{j-1}) + (\text{cmr}_j - \text{cmr}_{j-1}) - (A \times \text{cir}_j + \text{cmr}_j)) \]
\[ + \text{HOTS} \]

As was found to be the case with Euler integration in Chapter 2, it can be seen that initialising all residue registers to 0 will minimise the third order error (the second, third and fourth lines), without jeopardising the suppression of second order error.

C.2 Register Transfers for Nonlinear Equations

The register transfers prescribed for nonlinear DEs in Chapter 6 are suitable for the solution of linear DEs as well. As shown in Chapter 6, error analysis is greatly complicated with nonlinear DEs, even if only Euler integration is used. Consequently, the analysis of second P-C integration is carried out with the simple linear DE: \( \dot{y} = Ay \), and the results of extrapolation and simulation relied on to verify that equivalent transfers are correct also for nonlinear DEs. For CSS second order P-C integration of this simple DE, the general register transfers given in equations (6.3a-g) become:

- 217 - Section C.2
Appendix C  Truncation Error Analysis

YC + PIR + ΔPZ, PIR

A × ΔPZ + PMR + ΔYP, PMR

YC + h(3ΔYP − ΔYP1)/2 + YP

ΔYP + ΔYP1

YP + CIR + ΔCZ, CIR

A × ΔCZ + CMR + ΔYC, CMR

YC + h(ΔYP + ΔYC)/2 + YC

An extra register, ΔYP1, is necessary with these register transfers to contain the previous ΔYP value.

Following the method of Section C.1, equation (C.2a-g) is combined with equation (C.1f-i) and equations similar to (C.1j-l), to perform the truncation error analysis as follows, leading to an equation for \( E_{i+1} \):

\[
\Delta p_z_i = y_{i} - \Delta p_{iri} 2^{-m}
\]

\[
= y_{i} - E_{i} 2^{-M} - \Delta p_{iri} 2^{-m}
\]

\[
\Delta p_y_i = A\Delta p_z_i - \Delta p_{mri} 2^{-m}
\]

\[
= Axy_{i} - AxE_{i} 2^{-M} - (A\Delta p_{iri} + \Delta p_{mri}) 2^{-m}
\]

\[
y_{i+1} = y_{i} + \frac{1}{2}(3\Delta y_{p_i} - \Delta y_{p_{i-1}}) 2^{-k}
\]

\[
= y_{i} - E_{i} 2^{-M} + \frac{1}{2}(Axy_{i} - AxE_{i} 2^{-M} - (A\Delta p_{iri} + \Delta p_{mri}) 2^{-m}) 2^{-k}
\]

\[
- \frac{1}{2}(Axy_{i-1} - AxE_{i-1} 2^{-M} - (A\Delta p_{iri-1} + \Delta p_{mri-1}) 2^{-m}) 2^{-k}
\]

\[
= y_{i} + \frac{1}{4}A(3y_{i} - y_{i-1}) 2^{-k}
\]

\[
- (E_{i} + \frac{1}{2}A(3E_{i} - E_{i-1}) 2^{-k}) 2^{-M}
\]

\[
- \frac{1}{2}(A(3\Delta p_{iri} - \Delta p_{iri-1}) 2^{-k} + (3\Delta p_{mri} - \Delta p_{mri-1}) 2^{-k}) 2^{-m}
\]
Appendix C

Truncation Error Analysis

\[ \Delta z_i = y_{p_i} - \Delta c_i r_{i-1} 2^{-m} \]
\[ = y_i + \frac{1}{2} A(3y_i - y_{i-1})2^{-k} \]
\[ - (E_i + \frac{1}{2} A(3E_i - E_{i-1})2^{-k})2^{-M} \]
\[ - \left( \frac{1}{2}(A(3\Delta p - \Delta p_{i-1}) + 3\Delta p - \Delta p_{i-1})2^{-k} + \Delta c_i r_{i-1} \right)2^{-m} \]

\[ \Delta y_i = A \times \Delta z_i - \Delta c_i m_{i-1} 2^{-m} \]
\[ = A \times y_i + \frac{1}{2} A^2 (3y_i - y_{i-1})2^{-k} \]
\[ - (A \times E_i + \frac{1}{2} A^2 (3E_i - E_{i-1})2^{-k})2^{-M} \]
\[ - \left( \frac{1}{2} A(A(3\Delta p - \Delta p_{i-1}) + 3\Delta p - \Delta p_{i-1})2^{-k} \right) \]
\[ + \frac{1}{2} (A\Delta p - \Delta p_{i-1} + A\Delta c_i r_{i-1} + \Delta c_i m_{i-1})2^{-m} \]

\[ \Delta Y_{P_i} = \frac{1}{2}(\Delta y_{p_i} + \Delta y_{c_i}) \quad \{ \Delta Y_{P_i} \text{ represents final estimate of } \Delta y_i \} \]
\[ = A \times y_i + \frac{1}{2} A^2 (3y_i - y_{i-1})2^{-k} \]
\[ - (A \times E_i + \frac{1}{2} A^2 (3E_i - E_{i-1})2^{-k})2^{-M} \]
\[ - \left( \frac{1}{2} A(A(3\Delta p - \Delta p_{i-1}) + 3\Delta p - \Delta p_{i-1})2^{-k} \right) \]
\[ + \frac{1}{2} (A\Delta p - \Delta p_{i-1} + A\Delta c_i r_{i-1} + \Delta c_i m_{i-1})2^{-m} \]
\[ = \Delta y_i - \Delta E_i 2^{-m} \]

In this case, the composite change in residue registers may be defined by:

\[ R_i = \frac{1}{4} A(A(3\Delta p - \Delta p_{i-1}) + 3\Delta p - \Delta p_{i-1})2^{-k} \]
\[ + \frac{1}{2} (A\Delta p - \Delta p_{i-1} + A\Delta c_i r_{i-1} + \Delta c_i m_{i-1}) \quad (C.2h) \]

which gives:

\[ \Delta E_i = (A \times E_i + \frac{1}{4} A^2 (3E_i - E_{i-1})2^{-k})2^{-k} + R_i \]
\[ = E_{i+1} - E_i \]

hence:

\[ E_{i+1} = E_i + (A \times E_i + \frac{1}{4} A^2 (3E_i - E_{i-1})2^{-k})2^{-k} + R_i \]
\[ = (1 + A \times 2^{-k} + \frac{1}{4} A^2 2^{-2k})E_i - (\frac{1}{4} A^2 2^{-2k})E_{i-1} + R_i \]

Apart from the different definition of \( R_i \), this is the same recursive equation as was found for the linear register transfers, so
Appendix C  

Truncation Error Analysis

equation (C.1p) applies:

\[ E_{i+1} = E_0 + \frac{1}{2} \sum_{j=0}^{i} R_j + A((i+1)E_0 + \frac{1}{2} \sum_{j=0}^{i-1} (\sum_{n=0}^{j} R_n))2^{-k} + \text{HOTS} \]

C.2.1 Second Order Suppression

Substitution of equation (C.2h) into the above error equation permits the accumulation of second order truncation error to be assessed:

\[ E_{i+1} = E_0 + \frac{1}{2} \sum_{j=0}^{i} \left( \Delta \text{pir}_j + \Delta \text{pmr}_j + \Delta \text{cir}_j + \Delta \text{cmr}_j \right) \]
\[ + \frac{1}{4} \sum_{j=0}^{i-1} A(3\Delta \text{pir}_j - \Delta \text{pir}_{j-1}) + 3\Delta \text{pmr}_j - \Delta \text{pmr}_{j-1})2^{-k} \]
\[ + A((i+1)E_0 + \frac{1}{2} \sum_{j=0}^{i-1} (\sum_{n=0}^{j} (A\Delta \text{pir}_n + \Delta \text{pmr}_n + A\Delta \text{cir}_n + \Delta \text{cmr}_n))2^{-k}) + \text{HOTS} \]

\[ = E_0 + \frac{1}{2} \sum_{j=0}^{i} \left( \Delta \text{pir}_j + \Delta \text{pmr}_j + \Delta \text{cir}_j + \Delta \text{cmr}_j \right) \]
\[ + A((i+1)E_0 + \frac{1}{4} \sum_{j=0}^{i-1} (A(3\Delta \text{pir}_j - \Delta \text{pir}_{j-1}) + 3\Delta \text{pmr}_j - \Delta \text{pmr}_{j-1}) \]
\[ + \frac{1}{2} \sum_{j=0}^{i-1} (\sum_{n=0}^{j} (A\Delta \text{pir}_n + \Delta \text{pmr}_n + A\Delta \text{cir}_n + \Delta \text{cmr}_n))2^{-k} \]
\[ + \text{HOTS} \]
Appendix C  

Truncation Error Analysis

\[ E_0 + \frac{1}{2} \sum_{j=0}^{i-1} (A\Delta \text{pir}_j + A\Delta \text{pmr}_j + A\Delta \text{cir}_j + A\Delta \text{cmr}_j) \]

\[ + A((i+1)E_0 \]

\[ + \frac{1}{4}(A(3\Delta \text{pir}_i - \Delta \text{pir}_{i-1}) + (3\Delta \text{pmr}_i - \Delta \text{pmr}_{i-1}) \]

\[ + \sum_{j=0}^{i-1} (A\Delta \text{pir}_j + 2A\Delta \text{pmr}_j)) \]

\[ + \frac{1}{2} \sum_{j=0}^{i-1} \sum_{n=0}^{j} (A\Delta \text{pir}_n + A\Delta \text{pmr}_n + A\Delta \text{cir}_n + A\Delta \text{cmr}_n))2^{-k} \]

\[ + \text{HOTS} \]

\[ = E_0 + \frac{1}{2}(A(\text{pir}_i - \text{pir}_{i-1})+(\text{pmr}_i - \text{pmr}_{i-1})+A(\text{cir}_i - \text{cir}_{i-1})+(\text{cmr}_i - \text{cmr}_{i-1})) \]

\[ + A((i+1)E_0 \]

\[ + \frac{1}{4}(A(3(\text{pir}_i - \text{pir}_{i-1})-\text{pir}_{i-1}) + (3(\text{pmr}_i - \text{pmr}_{i-1})-\text{pmr}_{i-1}) \]

\[ + 2(A(\text{pir}_{i-1} - \text{pir}_{i-1})+(\text{pmr}_{i-1} - \text{pmr}_{i-1})) \]

\[ + \frac{1}{2} \sum_{j=0}^{i-1} (A(\text{pir}_j - \text{pir}_{j-1}) + (\text{pmr}_j - \text{pmr}_{j-1}) \]

\[ + A(\text{cir}_j - \text{cir}_{j-1}) + (\text{cmr}_j - \text{cmr}_{j-1}))2^{-k} \]

\[ + \text{HOTS} \]

Again, it can be seen that initialising all residue registers to \( \frac{1}{2} \) will most effectively suppress the accumulation of second order truncation error, which in the above equation is represented by the first line.

C.2.2 Third Order Suppression

Applying the same analysis as in Section C.1.2 reveals that the accumulation of third order truncation error can similarly be suppressed by initialisation of all residue registers to 0.
D. FINE CONTROL OF SECOND ORDER P-C STEP-SIZE

Restriction of changes in step-size to only either doubling or halving of the current step-size is now generally considered rather primitive, and for some DEs is even implied, by Krogh [53], to be inadequate. This restriction has been retained in the discussion of VSSDDAs because, provided that h remains a negative power of 2, multiplication by h can be achieved by a simple hard-wired shift, rather than an actual multiplication. Restricting h to negative powers of 2 also avoids the need for the value of h to be retained in any physical storage device. If values of h other than $2^{-k}$ are to be used, an "h" register becomes necessary, the length of which would be determined by the maximum potential value of k.

To permit greater flexibility in the choice of step-size, consider values of h in which, at most, two bits of the h register are set. Multiplication by such a value requires one addition, which, though not as convenient as a simple shift operation, remains much simpler than a full multiplication. Clearly, if either one or two bits of h (rather than only one bit) are allowed to be set simultaneously, more refined selection of step-size will be possible.

To discover the effects of 2-bit h values on other aspects of DDA design, modified second order P-C algorithms will firstly be derived for the cases where, if two bits of h are set, they must be adjacent, that is, h must either be of the form $2^{-k}$ (1 bit set - as usual), or $2^{-k} + 2^{-(k+1)}$ (2 adjacent bits set). Secondly, the implications of allowing any two bits of h to be set simultaneously will be discussed, in particular the effect on Second Order Differences.
Appendix D Fine Step-size Control

D.1 Two adjacent bits of h set

Permitting two adjacent bits of h (or one bit of h) to be set, has the immediate complication that the magnitude of the smallest possible relative change in step-size depends on the initial bit pattern. If one bit of h is set, e.g. $h = 0.00100_2$ then the smallest possible decrease in step-size will lead to $h = 0.00011_2$, while the smallest increase would give $h = 0.00110_2$. If two bits of h are set initially, e.g. $h = 0.00110_2$ then a decrease leads to $h = 0.00100_2$, while an increase would produce $h = 0.01000_2$. That is to say, small decreases in step-size will be either to $\frac{3}{4}h$ or $\frac{3}{4}h$, and small increases to either $\frac{1}{4}h$ or $\frac{1}{2}h$, depending on initial bit patterns. If the second order P-C method previously recommended is to be used, these step-size changes require special register transfers, which can be implemented through extensive modification of equation (6.3c).

The process used in Appendix B to derive the second and third order P-C algorithms for constant, halved and doubled step-size will now be applied to derive special second order predictor formulae for use when the small changes in DDA step-size described above are required. As explained in Appendix B, the standard corrector formula for constant step-size (equation (B.1b)) remains appropriate also when step-size is changed.
Appendix D

Fine Step-size Control

D.1.1 Step-size Decreased by 1/3

When step-size is decreased to \( \frac{2}{3} \) of its previous value, the situation after one more step of integration can be represented by:

\[
x \quad x \quad x
\]

\[
y_{c_{i-\frac{1}{2}}} \quad y_{c_{i}} \quad y_{c_{i+1}}
\]

where the change in step-size is indicated by the decrease in horizontal separation of the points, and the subscript attached to the previous point is adjusted according to the new step-size. The intermediate predictor value, \( y_{p_{i+1}} \), is to be calculated using \( y_{c_{i-\frac{1}{2}}} \) and \( y_{c_{i}} \), hence derivation of the modified algorithm starts with the formula:

\[
y_{p_{i+1}} = y_{c_{i}} + a \cdot y_{c_{i}} + b \cdot y_{c_{i-\frac{1}{2}}}
\]

and is completed by evaluating \( a \) and \( b \) by equating Taylor series expansions of both sides:

- L.H.S. = \( y_{i} + h \cdot \dot{y}_{i} + \frac{1}{2} h^{2} \ddot{y}_{i} + \text{HOTS} \)
- R.H.S. = \( y_{i} + a \cdot \dot{y}_{i} + b \cdot \ddot{y}_{i} \)
  + \( bx_{i} + \frac{1}{2} bx_{i} + \text{HOTS} \)

i.e. \( a + b = h \), \( -\frac{1}{2} b = \frac{1}{2} h \)

i.e. \( a = \frac{1}{2} h \), \( b = -\frac{1}{4} h \).

Substitution of these values for \( a \) and \( b \) leads to a special second order Adams formula for use when integration step-size is decreased by a factor of \( \frac{1}{3} \):

\[
y_{p_{i+1}} = y_{c_{i}} + \frac{1}{2} h \cdot y_{\dot{c}_{i}} - \frac{1}{3} h \cdot y_{\dot{c}_{i-\frac{1}{2}}}
\]

which may be implemented by modifying equation (6.3c) to:

\[
Y_{C} + h(4\Delta Y_{P} - \Delta Y_{PL}) + 3 = Y_{P} \quad \text{(D.1.1a)}
\]
Appendix D

Fine Step-size Control

D.1.2 Step-size Decreased by 1/4

When step-size is decreased to \( \frac{3}{4} \) of its past value, a simple representation of the situation after one more integration step is:

\[
\begin{array}{ccc}
& & \\
y_{ci-\frac{1}{4}} & y_{ci} & y_{ci+1} \\
& & 
\end{array}
\]

indicating that the predictor value, \( y_{pi+1} \), should be calculated using \( y_{ci-\frac{1}{4}} \) and \( y_{ci} \). The formula from which \( a \) and \( b \) are to be derived is:

\[
y_{pi+1} = y_{ci} + ax_{ci} + bxy_{ci-\frac{1}{4}}
\]

hence:

\[
\begin{align*}
\text{L.H.S.} &= y_i + hx_y_i + \frac{1}{2}h^2\ddot{y}_i + \text{HOTs} \\
\text{R.H.S.} &= y_i + ax_y_i \\
&+ bxy_{ci} - \frac{3}{4}bxy_{ci} + \text{HOTs} \\
i.e. \quad a + b &= h, -\frac{3}{4}b = \frac{1}{2}h \\
i.e. \quad a &= \frac{11}{4}h, \quad b = -\frac{3}{4}h.
\end{align*}
\]

and the correct second order Adams formula for use when step-size is decreased by a factor of \( \frac{1}{4} \) is:

\[
y_{pi+1} = y_{ci} + \frac{11}{4}hx_{ci} + \frac{3}{4}h(\ddot{y}_{ci} - \ddot{y}_{ci-\frac{1}{4}})
\]

To implement this formula, the appropriate modification of equation (6.3c) gives:

\[
\begin{align*}
& y_c + h(11\Delta Y_P - 3\Delta Y_P) + b \rightarrow Y_P
\end{align*}
\]
Appendix D  Fine Step-size Control

D.1.3 Step-size Increased by 1/3

When step-size is increased to $1\frac{1}{3}$ times its previous value, the usual representation of the situation after one more iteration is:

\[
\begin{array}{ccc}
  
  x & x & x \\
  \text{yc}_{i-\frac{3}{4}} & \text{yc}_i & \text{yc}_{i+1}
\end{array}
\]

The predictor value, $\text{yp}_{i+1}$, is to be calculated using $\text{yc}_{i-\frac{3}{4}}$ and $\text{yc}_i$, so the appropriate modified algorithm is:

\[
\text{yp}_{i+1} = \text{yc}_i + a \text{yc}_{i-\frac{3}{4}} + b \text{yc}_{i-\frac{3}{4}}
\]

where $a$ and $b$ are again evaluated by equating Taylor series expansions of both sides:

L.H.S. = $y_i + hx\dot{y}_i + \frac{1}{4}h^2\ddot{y}_i + \text{HOTS}$

R.H.S. = $y_i + a\dot{y}_i$

\[+ b\dot{y}_i - \frac{7}{3}bhx\dot{y}_i + \text{HOTS}\]

i.e. $a + b = h$, $-\frac{7}{3}b = \frac{1}{2}h$

i.e. $a = \frac{5}{3}h$, $b = -\frac{7}{3}h$

Thus the second order Adams formula for use when step-size is increased by a factor of $\frac{1}{3}$ is:

\[
\text{yp}_{i+1} = \text{yc}_i + \frac{5}{3}h\text{yc}_{i-\frac{3}{4}} - \frac{7}{3}h\text{yc}_{i-\frac{3}{4}}
\]

and the correct version of equation (6.3c) becomes:

\[
\text{yc} + h(5\Delta\text{yp} - 2\Delta\text{yp}_i + 3) = \text{yp} \tag{D.1.3a}
\]
D.1.4 Step-size Increased by 1/2

When step-size is increased to 1 1/2 times its previous value, the situation after one more step of integration is:

\[ \begin{array}{ccc}
  & x & \\
y_{c_{i-3/4}} & x & y_{c_{i+1}} \\
\end{array} \]

with change in step-size indicated as usual by the increase in horizontal separation of the points. The predictor value, \( y_{p_{i+1}} \), is to be calculated using \( y_{c_{i-3/4}} \) and \( y_{c_{i}} \), using the formula

\[ y_{p_{i+1}} = y_{c_{i}} + a \cdot y_{c_{i}} + b \cdot y_{c_{i-3/4}} \]

and evaluating \( a \) and \( b \) in the usual manner:

L.H.S. = \( y_{i} \cdot h \cdot \dot{y}_{i} + \frac{1}{2} h \cdot \ddot{y}_{i} + \) HOTs

R.H.S. = \( y_{i} + a \cdot \dot{y}_{i} + b \cdot \dot{y}_{i} - \frac{3}{4} b \cdot h \cdot \ddot{y}_{i} + \) HOTs

i.e. \( a + b = h \), \(-\frac{3}{4} b = \frac{1}{2} h \)

i.e. \( a = \frac{7}{4} h \), \( b = \frac{9}{4} h \)

To give:

\[ y_{p_{i+1}} = y_{c_{i}} + \frac{7}{4} h \cdot \dot{y}_{c_{i}} - \frac{3}{4} h \cdot \ddot{y}_{c_{i-3/4}} \]

\[ = y_{c_{i}} + h \cdot \dot{y}_{c_{i}} + \frac{3}{4} h (y_{c_{i}} - y_{c_{i-3/4}}) \]

when step-size is increased by a factor of \( \frac{3}{2} \). In this case the modification of equation (6.3c) necessary is:

\[ YC + h(7 \Delta YP - 3 \Delta YPL) + 4 \rightarrow YP \quad (D.1.4a) \]
In summary of these four results, the need to multiply by 11 (in D.1.2a) or by 7 (in D.1.4a) implies that an extra addition is required compared to equation (6.3c), but the division by 3 (in D.1.1a and D.1.3a) is a much more significant complication. Division by 3 cannot be achieved by the simple shift needed by equation (6.3c), and cannot be performed without loss of accuracy, for reasons explained in Chapter 1.

It may thus be concluded that, if 2-bit values of h are desired, hardware Adams prediction immediately after a change of step-size is not feasible, suggesting that whenever step-size is changed a single-step prediction is a necessity. Since error estimates are not made immediately after a change in step-size (see Chapter 4) the use of a different formula for those steps will not influence step-size control, however, unless a simple Euler prediction is employed, the increase in hardware complexity will be prohibitive.

D.2 Any two bits of h set

Since Adams prediction is not feasible immediately after step-size is changed, even if only adjacent bits of h are set, it is reasonable to consider allowing any two bits of h to be set simultaneously. No further complication is introduced into multiplication by h by this extra flexibility in step-size, since still only an addition is necessary. The advantage of such h values is that more precise control of step-size is made possible: step-sizes based on the relationship between the error estimate and the local error tolerance could be selected, as is popular in software subroutines, provided that the calculated step-size is rounded such
Appendix D Fine Step-size Control

that only two bits are set.

The extra hardware required for the calculation of the appropriate step-size would not be significant in comparison to the overall hardware cost, and could operate in parallel to the integration process itself. Of much greater significance is the difficulty which would be experienced in the generation of Second Order Differences, which occurs in series with other operations and thus contributes directly to the speed of the DDA. From equation (3.1.1a), the relationship between $h_{i+1}$ and $h_i$ should be simply a power of 2 for easy generation of $\Delta^2z_i$. This will not be the case with 2-bit $h$ values, hence actual divisions and multiplications will be necessary instead of shifts, which implies excessive complication of DDA modules as well as a greatly reduced iteration rate.

In conclusion, the improvement in solution speed gained by the more sophisticated step-size control mechanisms would be more than lost through the reduced iteration rate resulting from the increased hardware complexity.