COMPUTER AIDED ANALYSIS OF PERIODICALLY SWITCHED LINEAR NETWORKS

A Thesis submitted to the

Faculty of Engineering

of the University of Glasgow

for the degree of

Doctor of Philosophy

by

LIONEL BEREL WOLOVITZ

August 1987

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

I dedicate this thesis to my parents, Maurice and Eve Wolovitz, for whom my education has been of utmost importance and have always encouraged me to take the opportunities they did not have.

SUMMARY

Interest in analysing periodically switched linear networks has developed in response to the rapid development of sampled data communications systems. In particular, integrated circuit switched capacitor networks play an important part in modern analogue signal processing systems.

This thesis addresses the problem of developing techniques for analysing periodically switched linear networks in the time and frequency domains that are suited to computer implementation and therefore facilitate the development of efficient computer aided analysis tools for these networks.

Systems of large sparse complex linear equations arise in many network analysis problems and efficient techniques for solving these systems are crucial to the analysis methods developed in this thesis. By extending the concept of sparsity to include the type of the nonzero elements, very efficient solution and optimal ordering algorithms are developed.

A new method for computing the time domain response of linear networks is presented. The method is based on numerical inversion of the Laplace transform and polynomial approximation of the excitations. This high accuracy method is well suited to solving large stiff systems and is extremely efficient. The method is extended to periodically switched linear networks and provides the basis for frequency domain analysis.

A new frequency domain analysis method is presented that is orders of magnitude faster than existing techniques. This efficiency is achieved by developing a formulation such that AC analysis is not required, which allows the system to be solved as a discrete system. A special system compression reduces the solution of this discrete system to the solution of the network in one phase only. This solution step, which ordinarily requires $O(N^3)$ operations, is made more efficient by reducing the system to upper Hessenberg form in a preprocessing step, which then reduces the solution cost to $O(N^2)$ operations.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Professor J.I. Sewell, for the opportunity to undertake this research and for his support and guidance during the course of this work. I am especially grateful for his help in registering for this degree. Many thanks are due to the Department of Electronics and Electrical Engineering for the facilities provided.

I wish to thank Robert Henderson and Li Ping for many useful discussions and assistance during this work. A special thanks is due to Alistair Meakin for his help in validating part of this work and many helpful discussions. In particular I wish to thank Oliver Pun who first introduced me to some of the problems addressed in this work and for many early discussions.

I wish to acknowledge the financial support from the Science and Engineering Research Council and also acknowledge financial support for part of this work by the Ministry of Defence – Procurement executive.

Finally I would like to specially thank Jane for her love and patience during the course of this work; for being there if not always here.

TABLE OF CONTENTS

CHAPTER 3: TIME DOMAIN SOLUTION OF LINEAR NETWORKS

CHAPTER 4: FREQUENCY DOMAIN ANALYSIS OF PERIODICALLY SWITCHED LINEAR NETWORKS

CHAPTER ONE

INTRODUCTION

1.1) INTRODUCTION

Interest in analysing periodically switched linear networks developed in response to the rapid development of sampled data communications systems. These discrete time systems used switched linear filters in time division multiplex systems that used pulse amplitude modulation (PAM) or pulse code modulation (PCM) techniques. Before these developments the most widespread application of periodically switched networks was in modulators and demodulators for frequency translation in frequency division multiplex systems [1].

Today by far the most common application of periodically switched linear networks are switched capacitor (SC) networks. The use of SC networks has become widespread in recent years. The primary reasons for this popularity are that they can be fully integrated using MOS technology and are VLSI compatible. Recently interest has grown in gallium–arsenide (GaAs) implementations of SC circuits for high frequency applications. Other attractive properties of these networks are small chip area requirements, low power consumption and ease of achieving high precision ($< 0.5\%$) filter specifications. As the state of the art has progressed, larger and more complex networks have been realised and the frequency of operation of the filters has been pushed into the megahertz range. Consequently applications have grown from audio frequency filtering to high frequency communications systems.

This thesis addresses the problem of developing techniques for analysing periodically switched linear systems in the time and frequency domains that are suited to computer implementation and therefore facilitate the development of efficient computer aided analysis tools for these networks. The analysis which is developed is applicable to arbitrary periodically switched linear networks although the techniques are intended primarily for the analysis of nonideal SC networks.

The foundation for the exact analysis of periodically switched networks was given by Bennett [2]. His method, though exact, cannot be easily applied to general switched networks because it requires analytic time domain responses of the network variables and is therefore not easily implemented on a computer. A more efficient method was developed by Desoer [3] which used the successive approximation scheme for circuits with a very small ratio of switch closure time to switching period. This condition severely restricts the applicability of this approximate approach. Desoer developed an exact analysis [4] using the state space formulation similar to Bennett's approach and consequently also not suited to computer implementation. An entirely different approach based on a pole–zero description, together with the Fourier analysis, is given by Fettweiss [5]. This novel approach only considers networks with a single switch and therefore is of limited practical use. Sandberg [6] presented an approach to solve a more general class of time varying circuits which is similar to the approach of Desoer [4]. Sun and Frisch [7] extended these approaches to include an arbitrary number of switches based on the state space formulation.

The first major step in developing techniques suited to computer implementation was taken by Liou [8]. This state space based formulation gave explicit closed form solutions for both time and frequency domain solutions. The method is applicable to arbitrary circuit configurations with an exponentially modulated cisoidal input and can handle cases in which discontinuities in state variables occur at the switching instants. The method is however limited to systems with only two different periodic states, called phases. This method was generalised to include an arbitrary number of phases and arbitrary deterministic or stochastic inputs [1]. Unfortunately both these methods, though implemented in computer programs, are not efficient and are therefore limited to analysing small networks.

Many different methods of analysing SC filters have been proposed and a number actually implemented in computer programs [9]. The need for these CAD tools becomes more essential as the networks being designed grow in size and complexity. The majority of these programs are designed for the analysis of ideal SC networks, that is

all switches are assumed to have infinite off−resistances and zero on−resistances and the amplifiers are assumed to have infinite bandwidth (modelled as ideal controlled sources). Because these ideal networks do not have any resistive elements, these networks can be very efficiently modelled using difference equations [9]. A number of very efficient programs that use a compaction process to reduce the size of these systems have been developed. These programs are capable of analysing very large networks with an arbitrary number of phases and circuit configuration [10], [11], [12]. These programs provide a useful design facility for verifying and evaluating alternative design techniques. The facilities provided are usually a subset of time domain analysis, frequency domain analysis, sensitivity analysis, optimisation and symbolic analysis. However the overiding disadvantage of all these programs is that they fail to model real integrated networks accurately due to the assumption of ideal elements and therefore are used only as initial design aids.

When SC networks are implemented in integrated circuits (usually MOS technology) various imperfections occur. These imperfections are usually characterised as linear, nonlinear and statistical. Different analysis techniques are needed for these different imperfections.

The most common linear imperfections of SC networks are the parasitic capacitances associated with the switches, finite amplifier gain, switch resistances and finite amplifier gain bandwidth product [9]. Design techniques have been developed that can overcome the effects of the parasitic capacitances, which are however dependent on high gain amplifiers, and in some cases, matching of the parasitic capacitances [13]. The effects of parasitic capacitances and finite amplifier gain can be modelled by the ideal analysis techniques.

The nonlinear imperfections of SC networks are the nonlinear characteristics of the switches and amplifiers. The most important effect in the amplifiers are finite slew−rate and amplifier limiting which can introduce distortion and limit the frequency of operation of the networks. The switches introduce signal dependent nonlinear distortion and also clock feedthrough distortion due to nonlinear coupling of the clock signals into the main signal path. To model

these effects a nonlinear time domain analysis program and nonlinear models of the devices are required. Techniques for nonlinear time domain analysis are well developed and distortion products can be obtained using the FFT algorithm [14]. By taking into account the characteristics of SC networks, an efficient program for the time domain analysis of SC networks was developed, which includes a distortion analysis capability [15], [16]. However these programs require very large amounts of computation time and therefore are of use only in final design stages.

The statistical imperfections of SC networks are the noise sources associated with the switches and amplifiers. To simulate the noise response of a SC network a noise analysis program is needed that takes into account the noise sources, the resistive time-constants (transient effects) and the noise spectrum foldover due to the sampling of the switches [9]. A number of approximate techniques have been proposed that are of varying applicability [9]. Two programs of general applicability have been developed, both based on nonideal frequency domain analysis techniques [17], [18], requiring a substantial amount of computation.

To accurately model the effects of arbitrary on and off switch resistances, finite amplifier gain bandwidth, amplifier input and output impedances, analysis methods are required that take into account the transient effects caused by these imperfections. These methods must be suited to computer implementation to provide designers with efficient tools for evaluating these effects and to help in the design of networks that circumvent these imperfections. The most useful tools are time and frequency domain analyses and multiparameter sensitivity analysis. Other tools may be needed depending on the application. In particular high frequency and low power consumption applications require special attention to finite amplifier gain bandwidth and switch on resistance values. These parameters need to be carefully calculated to optimise overall performance, which requires the use of optimisers in conjunction with the analysis programs. Recently attempts have been made to include some of these imperfections into the design process [19] and it is anticipated that future design methods will incorporate these resistive imperfections and other effects such as

sinx/x droop into the SC networks. However the resistive time-constants are not designable parameters as variations from chip to chip are very large, so Monte Carlo analysis and design centering tools will be required to provide a practical overall design procedure.

For a non-ideal analysis method to be usable within the context of Monte Carlo analysis, design centering or optimisation, the cost in terms of computation time must be low for both calculating the network response and the preprocessing required for the method. With the increasing use of large and complex networks, the analysis methods must be applicable to large networks, that is the method must be accurate and numerically stable even for large networks and the computation cost should increase modestly with increasing network size.

The non-ideal SC analysis methods that have been proposed to date do not meet these requirements. Even the formulations that are best suited to efficient computer implementation are inherently slow due to the need for AC analysis at each frequency point, in addition to the Z– domain analysis. The objective of this work was to overcome these limitations and develop a very fast method for analysing non-ideal SC networks that could provide the basis for very efficient sensitivity and noise analysis and be efficiently used with an optimiser. This thesis presents the new approaches to time and frequency domain analysis of non-ideal SC networks, formulated as general periodically switched linear networks. Central to these analyses is a new method for computing the time domain response of linear networks. This new method requires the development of efficient algorithms for computing the extended state transition matrix and excitation response vectors. The efficiency of these algorithms derives from a new approach developed for solving large sparse sets of complex linear equations.

All the above analyses are implemented in a computer program and extensive results on the performance of the algorithms are given.

1.2) NOTES

The term 'flop' (floating point operation) is used extensively throughout this thesis. The definition of a flop was defined more precisely [20] to be the time required for a particular computer system to execute the FORTRAN statement

```
A(I,J) = A(I,J) + T * A(I,K)                    (1.1)
```

which involves one floating point multiplication, one floating point addition, a few subscript and index calculations and a few storage references. This definition of a flop is used throughout this thesis.

The network equations used in this work are formulated using the MNA approach [21] and an efficient row swapping algorithm that ensures a nonzero diagonal [22]. This formulation has many advantages [22], notably that it preserves the inherent sparsity in the network and allows the inclusion of voltage sources using topological values.

No typographical distinction is made between matrices, vectors and scalars. The usual conventions of capital and lowercase letters are used. Where a new quantity is introduced for the first time, it is explicitly stated whether the quantity is vector or scalar. Other notation that is used is introduced where needed.

All the algorithms discussed in this thesis were implemented in a FORTRAN 77 program called FOOLSCAP. This name was chosen as a play on the words Full Switched Capacitor Analysis Program. This program was implemented on a μVAX II under VMS. All the results presented in this thesis were obtained from this implementation.

1.3) OUTLINE OF THE THESIS

Chapter One introduces the subject of periodically switched linear networks and discusses the computer aided analysis tools that are required for analysing them. General notation and terms that are used throughout the thesis are briefly discussed. The outline of the thesis is presented and areas of the work which are novel and original are highlighted.

Chapter Two addresses the problem of solving sparse sets of complex linear equations and introduces the concept of domain types which is key to the efficient techniques developed to solve these equations. The interpretable code generation scheme and optimal ordering algorithms developed are discussed in detail and extensive practical results are used to compare the various algorithms and indicate the massive gains in efficiency made possible by these techniques.

Chapter Three is concerned with the efficient time domain solution of linear networks. The problem is formulated mathematically and the characteristics of the equations that make the solution difficult are discussed. The numerous methods that have been proposed to solve this problem are discussed in light of these difficulties and their suitability evaluated. A new approach to this problem is developed that makes use of the extended state transition matrix and polynomial approximations of the network excitations. Methods of calculating these matrices and approximations are presented and efficient computer implementation strategies are discussed. Finally the accuracy and stability of the new method is evaluated and the approach compared with other methods in terms of accuracy and efficiency.

Chapter Four considers the frequency domain analysis of periodically switched linear networks. Many different methods have been proposed and a number actually implemented in computer programs. The most important and successful approaches are discussed and compared on the basis of their limitations of generality and computational efficiency. A new method is developed to overcome the drawbacks of these methods, which requires the development of a number of different techniques. The time domain solution technique presented in Chapter Three is generalised to periodically switched linear networks. This solution is transformed to the Z− domain and a discrete system constructed. An efficient method for solving this discrete system is presented. The frequency analysis algorithm is presented, based on the solution of this discrete system. Finally the verification of the theory and its implementation in a computer program is discussed and the performance of the implemetation is compared with other methods.

Chapter Five is concerned with methods that can efficiently solve a particular form of complex linear equations that arises in the frequency domain analysis method presented in Chapter Four. Three approaches, the direct approach, the iterative approach and reduction to simpler forms are considered and discussed in detail. The performance of all three approaches is compared on the basis of results obtained from the implementation of the approaches in the frequency analysis program. Detailed comparisons of the three most effective methods are given and clearly show the substantial savings afforded by these techniques.

Finally the conclusions and suggestions for further work are presented in Chapter Six.

1.4) STATEMENT OF ORIGINALITY

The following most significant results of the research work presented in this thesis are, to the best of our knowledge, original:

– In Chapter 2, the introduction of the concept of domain types which extends the concept of sparsity from simply a zero/nonzero structure to include the type of the nonzero elements. The development of methods for taking advantage of the domain type structure using interpretable code generation and an associated numeric interpretor. The three optimal ordering algorithms designed to take into account the domain type structure. The modification of the numeric interpretor to accumulate operation statistics, which is a very useful symbolic tool for comparing different solution algorithms.

– In Chapter 3, the development of a time domain solution technique based on polynomial approximation of the excitations and numerical Laplace transform inversion. The derivation of polynomial approximation formulas which calculate the coefficients of the polynomial explicitly. The technique used to calculate the extended state transition matrix and excitation response vectors, which makes optimal use of the sparsity of the matrices.

– In Chapter 4, the development of the overall frequency domain analysis method. The use of polynomial approximation of the excitations, which avoids the need for AC analysis at each frequency point. The derivation of the method for calculating the time domain response of periodically switched linear networks with arbitrary input.

– In Chapter 5, the extension of the Hessenberg approach to multiphase SC networks and incorporation into the block Gauss elimination solution method used to efficiently solve the discrete system in Chapter 4. The development of the tridiagonal approach and its application to multiphase SC networks.

REFERENCES FOR CHAPTER ONE

[1] T. Ström and S. Signell, "Analysis of periodically switched linear networks", IEEE Trans. CAS, Vol. CAS−24, Oct. 1977, pp.531−540.

[2] W.R. Bennett, "Steady state transmission through networks containing periodically operated switches", IRE Trans. Circuit Theory, Vol. CT−2, Mar. 1955, pp.17−21.

[3] C.A. Desoer, "A network containing a periodically operated switch solved by successive approximations", Bell Syst. Tech. J., Vol. 36, pp.1403−1428, Nov. 1957.

[4] C.A. Desoer, "Transmission through a linear network containing a periodically operated switch", 1958 Wescon Conv. Rec., pt. 2, pp.34−41.

[5] A. Fettweis, "Steady state analysis of circuits containing a periodically operated switch", IRE Trans. Circuit Theory, Vol. CT−6, pp.252−260, Sep. 1959.

[6] I.W. Sandberg, "The analysis of networks containing periodically variable piecewise constant elements", Proc. 1961 Nat. Electron. Conf., Vol. 17, pp.81−97.

[7] Y. Sun and I.T. Frisch, "Transfer functions and stability for networks with periodically varying switches", Proc. 1st Asilomar Conf. Circuits and Systems, Nov. 1967, pp.130−142.

[8] M.L. Liou, "Exact analysis of linear circuits containing periodically operated switches with applications", IEEE Trans. CT, Vol. CT−19, Mar. 1972, pp.146−154.

[9] M.L. Liou, Y.L. Kuo and C.F. Lee, " A tutorial on computer aided analysis of switched capacitor circuits", Proc. IEEE, Vol. 71, No. 8, Aug. 1983, pp.987−1005.

[10] J. Vanderwalle, H. De Man and J. Rabaey, "Time, frequency and z−domain modified nodal analysis of switched capacitor networks", IEEE Trans. CAS, Vol. CAS−28, No. 3, pp.186−195, Mar. 1981.

[11] A.D. Meakin, J.I. Sewell and L.B. Wolovitz, "Techniques for improving the efficiency of analysis software for large switched−capacitor networks", Proc. 28th Midwest Symposium on Circuits and Systems, pp.390−393, Aug. 1985.

[12] J. Vlach, K. Singhal, and M. Vlach, "Computer oriented formulation of equations and analysis of switched−capacitor networks", IEEE Trans. CAS, Vol. CAS−31, Sept. 1984, pp.753−765.

[13] G.S. Moschytz, ed., "MOS switched capacitor filters: analysis and design", IEEE Press Selected Reprint Series, 1984.

[14] L.W. Nagel, "SPICE2 : A computer program to simulate semiconductor circuits", ERL Memo ERL−M520, University of California, Berkeley, 1975.

[15] L.B. Wolovitz and J.I. Sewell, "Advanced switched capacitor analysis software", Final Report, Dept. Electronics and Electrical Engineering, University of Glasgow, 1986.

[16] L.B. Wolovitz and J.I. Sewell, "Efficient computer techniques for the exact analysis of all nonideal effects of switched−capacitor networks in the time−domain", Proc. IEEE Int. Symp. on Circuits and Systems, pp.373−376, May 1986.

[17] J. Rabaey, J. Vanderwalle and H. De Man, "On the frequency domain analysis of switched capacitor networks including all parasitics", Proc. IEEE Int. Symp. on Circuits and Systems, April 1981, pp.868−871.

[18] C.K. Pun, A.G. Hall and J.I. Sewell, "Noise analysis of switched capacitor networks in symbolic form", Proc. 29th Midwest Symposium on Circuits and Systems, Lincoln, 1986.

[19] M.A. Tan, C. Acar and M.S. Ghausi, "Design of switched capacitor filters using nonideal op amps", Journal of the Franklin Institute, Vol. 323, No. 1, pp.55–72, 1987.

[20] C.B. Moler and C. Van Loan, "Nineteen dubious ways to compute the exponential of a matrix", SIAM Review, Vol. 20, No. 4, pp.801–836, Oct. 1978.

[21] C.W. Ho, A.E. Ruehli and P.A. Brennan, "The modified nodal approach to network analysis", IEEE Trans. CAS, Vol. CAS–22, No. 6, June 1975, pp.504–509.

[22] L.B. Wolovitz, "Improved techniques for time domain analysis of switched capacitor networks", M.Sc. Thesis, University of Hull, 1986.

CHAPTER TWO

SOLVING SPARSE SETS OF COMPLEX LINEAR EQUATIONS

2.1) INTRODUCTION

Systems of large sparse linear equations arise in many numerical problems, notably network analysis and design. Sparse matrix techniques have enabled the solution of large problems and provided the efficiency to make optimisation and Monte Carlo analysis of large networks feasible [1].

This chapter addresses the problem of developing techniques that can efficiently solve large sparse sets of complex linear equations. These equations arise in network analysis when analysing networks in the complex frequency domain. Here the definition of a sparse matrix is one in which advantage can be taken of the percentage, distribution and type of nonzero elements [1].

The problem then is the solution of the N $\times$ N system of linear equations

$$Ax = b \tag{2.1}$$

where matrix A and vectors x and b are in the complex domain.
The matrix A has an arbitrary zero/nonzero structure, where the nonzero elements are real, complex or imaginary as arises in network analysis. Furthermore, with the widespread use of modern tableau equation formulation methods [2], the topological elements $+1$ and -1 are also considered. The 'type' of an element refers to whether the element is a topological, real, imaginary or complex.

The methods discussed here are all based on the direct LU decomposition method of solving equation (2.1). This approach consists of factoring A into the product

$$A = LU \tag{2.2}$$

of a lower triangular matrix L and an upper triangular matrix U, then solving the triangular system

$$Ly = b \tag{2.3}$$

by forward elimination, and the triangular system

$$Ux = y \tag{2.4}$$

by backsubstitution.

Sparse matrix methods gain their efficiency by avoiding redundant arithmetic operations in computing (2.2), (2.3) and (2.4). This is achieved by organising the computations such that access to and multiplication by zero valued elements are avoided. Similarly multiplication by a topological is replaced by an addition/subtraction. Although computational speed is the primary consideration and motivation, sparse methods also have an obvious benefit in storage requirements, in that only the nonzero elements (plus indexing information to access these elements) need be stored. These savings can be of importance for very large problems, though today computers with large main store and virtual memory operating systems are common and therefore storage is generally of secondary concern.

The solution of the triangular systems (2.3) and (2.4) by forward elimination and backsubstitution respectively, does not alter the sparsity structure of the L and U matrices and therefore the number of arithmetic operations involved is only dependent on the nonzeros in L and U. For this reason sparse methods generally concentrate primarily on the first stage of factoring A into LU. This problem involves three basic steps: reordering the equations based on sparsity and solution efficiency considerations, a symbolic stage that prepares data structures and information for the last step, the actual numeric factorisation.

In the applications mentioned above, the solution of equation (2.1) must be repeated many times, where the numerical values of A change (due to frequency variance or parameter changes), but the sparsity and type structure remain fixed. Steps one and two are done only once, while the third step (the solution process) is repeated many times.

2.1.1) DOMAIN TYPES

Standard sparse methods only consider the zero/nonzero structure of the problem. However as mentioned previously by taking into account topological elements, further computational savings can be achieved [3]. The category of different elements was broadened by Hachtel [4], who pioneered the concept of 'variab ility' type. In this case the application was time domain optimisation of nonlinear networks, which has various levels of looping and the system (2.1) is solved at the innermost loop. The system thus has entries that are constant, topologicals, vary only with each optimisation step, each time–step or each nonlinear iteration. By taking advantage of these different variability types the scope of avoiding redundant arithmetic operations was extended to the various levels and achieved impressive savings [4].

In this work we consider complex equations which vary with frequency and therefore the matrix A has entries that are topologicals, constants or frequency dependent. To efficiently solve these kind of systems a new 'domain' type is introduced. The motivation for this domain type is best explained by means of example.

Consider the following fragment of code

$$\text{Sum} = \text{Sum} + A(47) \times A(93) \tag{2.5}$$

which is the most frequent operation in the numeric factorisation stage. Writing the complex values explicitly i.e.

$$\begin{aligned} A(47) &= Re(47) + jIm(47) \\ \text{Sum} &= \text{SumR} + j\text{SumI} \end{aligned} \tag{2.6}$$

code fragment (2.5) becomes

$$\begin{aligned} \text{SumR} &= \text{SumR} + Re(47) \times Re(93) - Im(47) \times Im(93) \\ \text{SumI} &= \text{SumI} + Re(47) \times Im(93) + Im(47) \times Re(93) \end{aligned} \tag{2.7}$$

which requires 4 additions/subtractions, 4 multiplications and 4 array accesses. Now consider the case where A(47) is real and A(93) is imaginary, i.e.

$$A(47) = Re(47)$$
$$A(93) = jIm(93) \qquad (2.8)$$

Code fragment (2.5) now reduces to

$$SumI = SumI + Re(47) \times Im(93) \qquad (2.9)$$

which only requires 1 addition, 1 multiplication and 2 array accesses. Finally consider the case where A(47) is a topological of value +1, which then reduces (2.5) to

$$SumI = SumI + Im(93) \qquad (2.10)$$

Quite clearly dramatic savings can be achieved by taking advantage of the domain types, which are listed in Table 2.1.

Type 1	elements which are +1
Type 2	elements which are -1
Type 3	elements which are real i.e. x = a
Type 4	elements which are imaginary i.e. x = jb
Type 5	elements which are complex i.e. x = a + jb

Table 2.1 Description of domain types

2.1.2) SYMBOLIC AND NUMERIC FACTORISATION

The three stages outlined above for factorising matrix A are discussed with an emphasis placed on methods that take advantage of the matrix type structure.

The first stage, reordering the equations, is independent of the other two stages and is therefore considered separately and discussed after the solution stages. The symbolic phase and the numeric factorisation phase, though two distinct steps, are intimately related and thus treated as one when discussing various approaches to the LU factorisation. Three standard approaches are identified **[3]**, though there are many variants and hybrids of the methods. These approaches are: generated machine code (MC), looping index (LI) and generated interpretable code (IC).

The first approach, generated machine code, analyses the matrix

structure in the symbolic phase and generates a loop free code that implements the factorisation [6]. The third phase, numeric factorisation, then simply involves executing this code, which has the advantage of being extremely fast since no testing or branching is performed and every variable is accessed directly. Another major advantage of the MC approach is the ease with which it can handle typed problems. The MC approach has been effectively used to solve variability typed problems [4], and could be easily adapted for domain type problems. However the MC approach has a number of important disadvantages. The first is that the compiled code can be very long [1], growing rapidly for large systems. The problem is even worse for complex equations [3]. Because the code is executed sequentially it does allow the code to be held in secondary store but this slows execution substantially [1]. The second disadvantage is that for efficiency the code is often generated in machine code, which makes implementations machine dependent. The third disadvantage is that this code has to be translated into machine code (in the case of high level language or assembler implementations) and then linked to the rest of the application program, which can be quite slow and is again machine dependent. For these reasons this approach was rejected as a viable method of handling domain types.

The second approach, looping index, analyses both the initial and decomposed matrix structures in the symbolic phase and generates a data structure for holding the nonzero entries plus indexing information for efficiently accessing rows and columns of the matrix [7]. The third phase then performs the numeric factorisation by using this indexing information to access elements involved in the elimination steps. The advantages of this approach are that it is readily implemented in a high level language (and is therfore machine independent), the symbolic phase is significantly quicker than the MC approach and the information generated requires much less storage. It has the disadvantage of being much slower than the MC approach because of the looping and two levels of indirect addressing for accessing entries. However its main disadvantage is the difficulty of taking advantage of typed problems. These difficulties, discussed in [3], of handling variablilty types are even worse for the domain types and therefore this method was also rejected.

The third approach, generated interpretable code, is similar to the MC approach in that it also generates a loop free code that implements the factorisation. It differs from MC in that this code is not a machine or high level code but is designed to be interpreted in the third phase by an interpretor that interprets the sequence of instructions and performs the numeric factorisation [8]. It shares with MC the advantage of being very fast, though it is slower because of the overheads of the interpretor and the indirect addressing of entries, and the ease of handling types problems. Although, as for MC, the code can grow quite large, this is less of a problem as the instructions are much shorter. In the case of complex equations this is an important advantage as the instructions are exactly the same length as for real instructions, whereas they are at least four times longer for MC [3]. Compared to the LI approach, IC still requires more storage but is significantly faster. Perhaps the only disadvantage of the IC approach is the need for an interpretor. Though it is relatively straightforward to implement, great care is needed to ensure that it executes efficiently. For utmost efficiency it is desirable to implement the interpretor in assembler, which has the disadvantage of being machine dependent, time-consuming and error prone. However well coded high level language implementations are perfectly adequate and have the advantage of being portable. Based on these considerations, the IC approach was selected to be used for solving the domain type problem.

This solution is presented in section 2.2, where the design of the interpretable instruction set and the efficient implementation of the code generation and interpretor are discussed. Results of this implementation are presented in section 2.4.

2.1.3) RE-ORDERING THE EQUATIONS

As mentioned previously the first step in solving the equations (2.1) is to re-order the equations. The objective of an ordering algorithm is to try and minimise the number of fills produced during the factorisation or to minimise the number of arithmetic operations for this factorisation. In general an algorithm that minimises the fill minimises the amount of computation for the factorisation, though the reverse is

not necessarily true, as is shown in the results presented at the end of this chapter. Therefore the objective in this work is to specifically minimise the amount of computation, which is proportional to the execution speed of the solution process, which after all is what one wants to minimise.

The algorithms for reordering the equations are generally called optimal ordering algorithms, which is misleading because no known practical algorithm can ensure a global optimum ordering. This is because of the combinatorial explosion of the problem [1]. All practical algorithms are local methods in that they select at each step the pivot which has a minimum cost for the next pivot step. It has been shown [9], that no ordering based solely on local criteria can be guaranteed to produce a globally best ordering. The various algorithms differ essentially in the way they calculate these costs. The most widely used algorithms are the Markowitz [10] which attempts to minimise the number of multiplications and the minimal fill [11], [12] which attempts to minimise the amount of fill, though many others have been suggested [1]. Most algorithms tacitly assume that numerical stability is not adversely affected by the reordering. In many problems this assumption is justified and in fact diagonally dominant matrices allow the algorithms to restrict pivot selection to the diagonal, which makes the algorithms far more efficient than if a full pivot search were used. In the context of network analysis, the formulation of the equations is such that diagonal pivoting may be used and therefore the ordering algorithms presented here are restricted to diagonal pivot selection, though they are applicable to the more general case.

Even though many different methods have been proposed, only one has been specifically developed to handle typed matrices. This algorithm, OPTORD [4], is based on the Markowitz algorithm and is applicable to variability type problems. The algorithm which uses weighted costs dependent on variability type attempts to re-order the equations to minimise the number of multiplications at each of the different variability levels. Though effective, this algorithm is not applicable to domain type problems. Therefore three new algorithms have been developed that take account of the domain type structure of the matrices and attempt to minimise the arithmetic operations for

the factorisation stage, assuming that this stage takes full advantage of the domain type structure. The algorithms are based on the Markowitz, minimal fill and a hybrid of the two algorithms and are presented in section 2.3. The efficient practical implementation of these algorithms is discussed and extensive comparisons with other approaches are presented in the results of section 2.4.

2.2) DOMAIN INTERPRETABLE CODE APPROACH

There are two algorithms that may be used to LU decompose the matrix A. The algorithms are equivalent in that they require the same number of arithmetic operations and produce the same factorisation. They differ only in the organisation of the factorisation steps. The first is the Gauss elimination method with row normalisation,

$$\begin{aligned} a_{kj} &= a_{kj} \,/\, a_{kk} && j = k+1,..,N \\ a_{ij} &= a_{ij} - a_{ik} \times a_{kj} && i,j = k+1,..,N \end{aligned} \qquad (2.11)$$

and the second is the Crout method

$$\ell_{ij} = a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj} \qquad i \geq j \qquad (2.12)$$

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj}}{a_{ii}} \qquad i<j \qquad (2.13)$$

Because the two methods produce the same LU factors, the same algorithms may be used to solve the triangular systems (2.3) and (2.4). The system (2.3) is solved by forward elimination,

$$y_i = \frac{b_i - \sum_{j=1}^{i-1} \ell_{ij} y_j}{\ell_{ii}} \qquad (2.14)$$

and the system (2.4) by backsubstitution,

$$x_i = y_i - \sum_{j=i+1}^{N} u_{ij} x_j \qquad (2.15)$$

The Gauss elimination method is most widely used with the looping index approach whilst the Crout method is most commonly used with the generated code approaches. There are a number of reasons why the Crout method is more suitable for code generation. They all derive from the fact that although the two solution methods require the identical number of arithmetic operations, the Crout method requires fewer accesses to the matrix as element values are only updated once, whereas the Gauss elimination method requires multiple updates. Therefore the amount of code generated by the Crout method is less than the Gauss elimination method. This fact also makes the updating of the type structure of the matrix much simpler for the Crout algorithm and it is therefore potentially faster. Another advantage of the Crout algorithm is that the inner loop form allows the accumulation to be implemented in higher precision arithmetic which can reduce roundoff errors. However in this application this higher precision was not required and therefore this factor was not considered in deciding between the two methods. The Crout algorithm was selected primarily because of its ease of implementation and because it facilitates the generation of very efficient code.

2.2.1) INTERPRETABLE CODE INSTRUCTION SET

A set of instructions must be developed that can implement the LU decomposition formulae (2.12) and (2.13) and the solution formulae (2.14) and (2.15). There are a number of requirements of such an instruction set. The first is that it must be able to handle all the domain types given in Table 2.1. The second is that the organisation of the instruction set should be regular to make implementation of the code generation algorithms simple and efficient. The third requirement is that the instructions be simple so that they can be efficiently implemented in an interpretor with minimal overhead (for example no looping or multiple operations which require any testing or branching). A desirable feature, though not a strict requirement, is that the instruction set should be small, which makes the interpretor shorter and simpler.

Initialise

Sr = 0	Si = 0	1 a
Sr = 1	Si = 0	2 a
Sr = -1	Si = 0	3 a
Sr = Re(a)	Si = 0	4 a
Sr = 0	Si = Im(a)	5 a
Sr = Re(a)	Si = Im(a)	6 a

Inner product

Sr = Sr - 1		7
Sr = Sr + 1		8
Sr = Sr - Re(a)		9 a
Si = Si - Im(a)		10 a
Sr = Sr - Re(a)	Si = Si - Im(a)	11 a
Sr = Sr + Re(a)		12 a
Si = Si + Im(a)		13 a
Sr = Sr + Re(a)	Si = Si + Im(a)	14 a
Sr = Sr - Re(a) × Re(b)		15 a b
Si = Si - Re(a) × Im(b)		16 a b
Sr = Sr - Re(a) × Re(b)		
Si = Si - Re(a) × Im(b)		17 a b
Sr = Sr + Im(a) × Im(b)		18 a b
Sr = Sr + Im(a) × Im(b)		
Si = Si - Im(a) × Re(b)		19 a b
Sr = Sr - Re(a) × Re(b) + Im(a) × Im(b)		
Si = Si - Re(a) × Im(b) - Im(a) × Re(b)		20 a b

Normalise and store

Re(a) = -Sr		21
Im(a) = -Si		22
Re(a) = -Sr	Im(a) = -Si	23
Re(a) = Pr		24
Re(a) = -Pr		25
Re(a) = Sr × Pr		26
Im(a) = Si × Pr		27
Re(a) = Sr × Pr	Im(a) = Si × Pr	28
Im(a) = Pi		29
Im(a) = -Pi		30
Im(a) = Sr × Pi		31
Re(a) = -Si × Pi		32
Re(a) = -Si × Pi	Im(a) = Sr × Pi	33
Re(a) = Pr	Im(a) = Pi	34
Re(a) = -Pr	Im(a) = -Pi	35
Re(a) = Sr × Pr	Im(a) = Sr × Pi	36
Re(a) = -Si × Pi	Im(a) = Si × Pr	37
Re(a) = Sr × Pr - Si × Pi		
Im(a) = Sr × Pi + Si × Pr		38

more...

Table 2.2 Interpretable code instruction set

Store

```
Re(a) = Sr                                          39
Im(a) = Si                                          40
Re(a) = Sr                  Im(a) = Si              41
```

Pivot

```
Pr =  1 / Sr                Re(a) = Pr              42
Pi = -1 / Si                Im(a) = Pi              43
Pr =  Sr / (Sr  ×  Sr + Si  ×  Si)
Pi = -Si / (Sr  ×  Sr + Si  ×  Si)
Re(a) = Pr                  Im(a) = Pi              44
```

Load

```
Sr = Br(a)     Si = Bi(a)                           45 a
Sr = Xr(a)     Si = Xi(a)                           46 a
```

Cross product

```
Sr = Sr - Xr(a)             Si = Si - Xi(a)         47 a
Sr = Sr + Xr(a)             Si = Si + Xi(a)         48 a
Sr = Sr - Re(a) × Xr(b)
Si = Si - Re(a) × Xi(b)                             49 a b
Sr = Sr + Im(a) × Xi(b)
Si = Si - Im(a) × Xr(b)                             50 a b
Sr = Sr - Re(a) × Xr(b) + Im(a) × Xi(b)
Si = Si - Re(a) × Xi(b) - Im(a) × Xr(b)             51 a b
```

Multiply and store

```
Xr(a) =  Sr                 Xi(a) = Si              52
Xr(a) = -Sr                 Xi(a) = -Si             53
Xr(a) =  Sr  ×  Re(b)
Xi(a) =  Si  ×  Re(b)                               54 b
Xr(a) = -Si  ×  Im(b)
Xi(a) =  Sr  ×  Im(b)                               55 b
Xr(a) =  Sr  ×  Re(b) - Si  ×  Im(b)
Xi(a) =  Sr  ×  Im(b) + Si  ×  Re(b)                56 b

Return                                              57
```

Table 2.2 ***(continued)***

An instruction set that meets these requirements was developed and is given in Table 2.2. The instruction set consists of 57 operations which includes the RETURN instruction. This set is therefore not small, though it is believed to be close to a minimal set that implements formulae (2.12) through (2.15) taking into account domain types.

The method by which this instruction set was constructed is now described. By analysing the Crout formulae (2.12) and (2.13) four different operations can be identified. They are:

1)	initialisation	$sum = a_{ij}$
2)	inner product	$sum = sum - a_{ik} \times a_{kj}$
3)	normalise	$sum = sum / a_{ii}$
4)	store	$a_{ij} = sum$

Considering the 5 domain types the initialisation operation is then implemented by instructions 1 through 6. Note that instruction 1 is introduced for the case where a fill is introduced and therefore the variables that accumulate the inner product are initialised to zero.

Next consider the inner product operation. There are 25 poss ible combinations of the domain types for a_{ik} and a_{kj}, though about half of these are symmetric and therefore can use the same instruction. To meet the requirement of regularity these instructions are grouped systematically as is shown in Table 2.3. The negative signs in the table are for the symmetric operations and denote that the operands are reversed.

Looking at the normalise and store operations it is seen that the two operations can be combined into one step, which simplifies and reduces the size of the instruction set. However looking at formula (2.12) it is seen that a seperate store operation is still needed. Fortunately the instructions needed for this operation are a subset of the instructions for the combined normalise and store operation. This subset corresponds to the case where $a_{ii} = 1$. Applying the domain types to these operations, 21 instructions are needed which are regularly grouped as is shown in Table 2.4. The instructions are described in Table 2.2 where the store operation is shown seperately

for reasons of clarity. There are four null operations for the topologicals as no instructions are actually needed, only the type structure of the matrix need be updated. This updating procedure is discussed in the next section.

From the instruction set in Table 2.2 it is seen that there is a group of instructions for pivoting. This group was introduced as it is more efficient to hold the diagonal (the pivots) inverted for complex numbers, as then division by the pivot is implemented with at most 4 multiplications (compared to 6 multiplications and 2 divisions).

A similar procedure to the above is used for the forward elimination (2.14) and the backsubstitution (2.15) formulae. Again 4 different operations can be identified, which are:

1)	load	$sum = x_i$
2)	cross product	$sum = sum - a_{ij} \times x_j$
3)	normalise	$sum = sum / a_{ii}$
4)	store	$x_i = sum$

A compromise between ultimate efficiency and instruction set size was made when implementing these operations. This compromise concerns the allowable domain types for the vector x. If all domain types were allowed then 40 new instructions would be needed to implement the solution formulae. Now in practice it is found that generally the RHS vector (b) is complex and even if it contains a range of domain types, the solution vector x rapidly becomes overwhelmingly compex during the solution steps. Therefore it was decided to restrict the RHS vector to being type 5 (complex). Although the implementation of the formulae is not optimal, the difference is insignificant and in return a massive reduction in the number of instructions is obtained. Applying the techniques described above, 12 instructions are required to implement the 4 operations. These are instructions 45 through 56 in Table 2.2.

		OPERAND B				
		1	2	3	4	5
OPERAND A	1	7	8	-9	-10	-11
	2	8	7	-12	-13	-14
	3	9	12	15	16	17
	4	10	13	-16	18	19
	5	11	14	-17	-19	20

Table 2.3 Inner product instruction lookup table

		OPERAND B				
		1	2	3	4	5
OPERAND A	1	-	-	24	29	34
	2	-	-	25	30	35
	3	39	21	26	31	36
	4	40	22	27	32	37
	5	41	23	28	33	38

Table 2.4 Normalise and store instruction lookup table

2.2.2) INTERPRETABLE CODE GENERATION

Having constructed the interpretable code instruction set, algorithms are required that will generate the code for a given sparse domain matrix. Two sets of information are required for this code generation process. The first is the type of each nonzero element and the second is the address (or location) of these elements. Exactly how this information is organised or what data structures are used to hold the information is not important to the functioning of the algorithm, though it obviously has an impact on the efficiency of the algorithm as regards storage and speed. Usually the nonzero elements of the matrix are stored contiguously in a vector (or two vectors in the case of complex matrices) and the addresses then correspond to the position of the elements within these vectors.

The algorithm proceeds in the same manner that a numerical implementation of the Crout algorithm does, except that instead of performing the numeric operations, instructions are generated that implement these operations. Therefore efficient techniques for implementing the Crout algorithm [13] may be used to reduce searching and testing thereby speeding this part of the algorithm up.

The procedure for generating the appropriate instructions is simplified if lookup tables are used. This has the added benefits that the instruction set may then be easily modified and it is a very efficient method. The lookup table for the normalise and store operations has already been presented in Table 2.4. Given the two operands A and B, the appropriate instruction is found from the table and the addresses of A and B then complete the instruction. In the case of a negative instruction the addresses are reversed and obviously the positive valued instruction is generated.

As well as generating the instructions at each step of the Crout algorithm, the type structure of the matrix must be updated. For this two lookup tables are used. The first gives the resultant type of a multiplication operation and is shown in Table 2.5. The second, given in Table 2.6 is the resultant type after an additive operation.

		OPERAND B				
		1	2	3	4	5
OPERAND A	1	1	2	3	4	5
	2	2	1	3	4	5
	3	3	3	3	4	5
	4	4	4	4	3	5
	5	5	5	5	5	5

Table 2.5 Resultant type after multiplication

		OPERAND B					
		0	1	2	3	4	5
OPERAND A	1	1	3	3	3	5	5
	2	2	3	3	3	5	5
	3	3	3	3	3	5	5
	4	4	5	5	5	5	5
	5	5	5	5	5	5	5

Table 2.6 Resultant type after addition/subtraction

At the initialisation of an inner product, the type of the summation is set to that of the element a_{ij}. As each inner product is formed, the type of the multiplication (from Table 2.5) is used to update the type of the summation (from Table 2.6). Similarly the type is updated when multiplied by the pivot and finally the matrix is updated with this new type when the store operation is generated.

Various other bookkeeping tasks are required for allocating fill and avoiding redundant loads and subsequent stores in the cases where elements are not altered by the factorisation. The code generation algorithms for the forward elimnation and backsubstitution formulae are developed using the above techniques, except they can be simplified because of the fixed type of the RHS.

The above algorithms have been implemented in the program FOOLSCAP using FORTRAN 77. The algorithms are designed in a modular fashion and the total length of code (including comments) is 700 lines, which is a very modest size considering the complexity of the problem which they solve.

2.3) OPTIMAL ORDERING

The general aims of an optimal ordering algorithm have been discussed in the introduction. As mentioned in the introduction three new algorithms have been developed to take into account the domain type structure of the equations. All three algorithms assume that the equations will be solved using a method that takes advantage of the domain structure and therefore attempt to minimise the operations required by such methods. The performance of the algorithms when such methods are not used are discussed in the results section 2.4.

A common feature of all the algorithms is that the pivots are restricted to the diagonal which greatly simplifies the algorithms and therefore speeds up their execution. The search for a pivot along the diagonal may be done in either direction, forwards or in reverse. In this particular application it is known that the equation formulation method produces topologicals along the diagonal, which are ordered towards the bottom of the matrices [13]. Therefore better results are

expected if the orderings are done in reverse order. This result was confirmed by numerous experiments and again the results are discussed in section 2.4.

2.3.1) DOMAIN MINIMAL MULTIPLICATION ALGORITHM

The aim of the domain minimal multiplication (MM) algorithm is to minimise the number of multiplications needed to implement the formulae (2.12) and (2.13) using domain type techniques. To take into account the domain types the multiplication costs are weighted according to the actual number of multiplications needed to perform the multiplication of the two domain type operands. These multiplication costs are given in Table 2.7. The divisions required in the pivot step are considered as multiplications in forming the cost of the step.

This cost could be extended to include the number of additions/subtractions, but because multiplication operations require more time to execute they generally dominate the computation time. However a more accurate cost could be obtained by using a weighted cost of multiplications and additions, which is weighted according to their actual execution speeds. This extra sophistication was not tried and indeed it is suspected that it would not produce better orderings because by minimising multiplications the number of additions are also minimised, which is confirmed by experimental results.

If the matrix does not have different domain types, then this algorithm is identical to the Markowitz algorithm.

2.3.2) DOMAIN MINIMAL FILL ALGORITHM

The aim of this algorithm is to minimise the amount of fill produced when the formulae (2.12) and (2.13) are implemented using domain type techniques. This algorithm was developed primarily out of academic interest in comparing its performance with the ususal minimal fill (MF) algorithms. However, as in the case of the MF algorithm for untyped matrices, a good MF ordering generally produces a good ordering in terms of the number of multiplications.

To take into account the domain types, the concept of a fill has to be extended. Normally a fill is used to denote a zero element that becomes nonzero during the factorisation process. In the context of domain types, the two parts of a complex number (real and imaginary) are treated as seperate entities and therefore count as two locations which can be filled independently. Thus for example a real element has an imaginary part of zero and if it becomes complex then the imaginary part is treated as a fill.

So taking the above into consideration, the fill cost is then weighted accoring to the number of fills that are produced when the type of an element (operand B) is updated to a new value (operand A). These fill costs are given in Table 2.8. From the table it is evident that the topologicals are treated (from a fill point of view) as equivalent to a type 3 element. This is because they are in fact equivalent in terms of fill as defined above. Although this will have no impact on the performance of the algorithm in terms of fill, it is expected that in terms of multiplications this is a drawback, because of the failure to distinguish between topologicals and type 3 elements. This feature of the algorithm is clearly demonstrated in the results of section 2.4.

In the case where the matrix does not have different domain types, this algorithm is identical to the usual minimal fill algorithm.

2.3.3) DOMAIN HYBRID ALGORITHM

The aim of this algorithm is to minimise the number of multiplications needed to implement the formulae (2.12) and (2.13) using domain type techniques. This algorithm combines the domain minimal multiplication and minimal fill algorithms to try and achieve an even better algorithm.

		OPERAND B				
		1	2	3	4	5
OPERAND A	1	0	0	0	0	0
	2	0	0	0	0	0
	3	0	0	1	1	2
	4	0	0	1	1	2
	5	0	0	2	2	4

Table 2.7 Multiplication cost table

		OPERAND B					
		0	1	2	3	4	5
OPERAND A	1	1	0	0	0	1	0
	2	1	0	0	0	1	0
	3	1	0	0	0	1	0
	4	1	1	1	1	0	0
	5	2	1	1	1	1	0

Table 2.8 Fill cost table

The motivation for this combination is provided by two features found in each of the algorithms respectively. It is found that even though the MM algorithm performs very well in terms of minimising multiplications, it produces more fill than one would expect. This property is discussed further in the results section. By combining the two algorithms it is hoped that the multiplication cost would still be kept low due to the MM algorithm, but the amount of fill would be reduced by the MF algorithm, thereby reducing the overall number of multiplications even further.

Similarly it is found that even though the MF algorithm performs very well in terms of minimising fill, it produces orderings which are clearly not optimal in terms of multiplications. One of the reasons for this has already been discussed. By combining the algorithms it is hoped that the MM algorithm would overcome this drawback.

To compute the cost of a pivot step, a weighted sum of the costs of the individual algorithms as outlined above is used. Experimental results conclusively showed that the best results were obtained when equal weights were used. The performance of this hybrid algorithm is discussed in the results of section 2.4.

2.3.4) EFFICIENT IMPLEMENTATION STRATEGIES

It is crucial that an optimal ordering algorithm be efficiently implemented because if it is poorly implemented then the time taken to execute the algorithm can quite easily wipe out any benefit gained from using the algorithm.

All three algorithms are based on essentially the same Gauss elimination form of algorithm and therefore share many common features. The first technique used is to take out the pivot row and put it into a compact form that can then be efficiently processed [13]. At the same time the cost of the row normalisation is calculated. This technique can lead to substantial improvements in performance.

The second shortcut is to put a check inside the loop that accumulates the costs of the pivot step and to exit if the partial cost

exceeds the current minimal cost found so far. This technique then saves subsequent calculations which are redundant because it is already known that this pivot will be rejected, and can obviously save a lot of computation. To avoid processing elements that have already been processed and to speed up the search for candidate pivots, a list of remaining pivots is kept and only rows and columns corresponding to these pivots are considered. Although this technique necessitates an extra level of addressing, it avoids having to do any testing and therefore quickly recoups any loss of speed.

To update the type structure of the matrix, the lookup Table 2.6 is used. This updating step is time consuming and there does not appear to be any way of speeding it up. If a pivot of zero cost is found, the pivot is selected immmediately and therefore any subsequent processing is saved. Since such pivots will not alter the matrix in any way, no updating is necessary and therefore this stage is bypassed.

To calculate the cost of a pivot step for the minimal multiplication algorithm the cost of each domain multiplication is evaluated using Table 2.7. However this is very time consuming as $n_r \times n_c$ such evaluations are required for each pivot, where n_r and n_c are the number of nonzero elements in the row and column respectively. Inspecting Table 2.7 it is seen that the costs for a multiplication involving a type 5 element is double that for a type 3 or 4 element. This then suggests that individual evaluations are not needed, as the cost of multiplying the pivot row by a type 3 element can be evaluated once and thus the costs for the other rows is simply equal to this cost for type 3 and 4 elements, or double this cost for type 5 elements. This technique then only requires n_r cost evaluations, which dramatically improves the efficiency of the algorithm.

Unfortunately this technique is not applicable to the minimal fill algorithm and therefore individual fill cost evaluations are stil needed. A very clever technique for the untyped minimal fill algorithm was presented in [14]. This technique avoids having to re-evaluate all the individual fill costs for each pivot by retaining these costs and updating the costs after a particular pivot has been selected. However this technique is not applicable to this algorithm because the

domain fill costs cannot be updated in this way. The algorithm is therefore relatively slow. Because the hybrid algorithm is dependent on the minimal fill algorithm, it too suffers from this drawback and its execution speed is therefore dominated by the speed of this algorithm.

2.4) RESULTS

To be able to quantitatively compare the various ordering algorithms and the effectiveness of the domain code generation method a means of calculating the various operation costs is required. The interpretable code method provides a very attractive and simple solution to this problem. All that is required is to write a special interpretor program that instead of performing numeric operations accumulates statistics for the operations. This provides a very powerful tool for evaluating different ordering and code generation algorithms. The interpretor accumulates seperate totals for the number of multiplications, divisions, additions, subtractions, loads, stores and the various categories of fills.

In the following two sections, results are presented for a particular example matrix which is derived from the 9th order filter given in Fig. 2.1 **[2, p.142]**. This example is typical of the intended application of the methods described here. This particular example was chosen as it is of medium size (35 $\times$ 35) and complexity and therefore achieves a balance between a trivial and overwhelmingly complex example. Although the results are presented using this one example, the general performance of the algorithms is also discussed as one example can often be misleading.

2.4.1) OPTIMAL ORDERING

In the following comparisons, the statistics are accumulated assuming that full advantage is taken of the domain structure of the ordered matrices. For each algorithm a lot of detailed information is provided, so this information is briefly discussed. The type structure of the matrix is given after it has been ordered, but before LU decomposition. The dots indicate zero elements and the nonzero elements are indicated by their actual type numbers. Below this the statistics for the LU decomposition are given. The totals at the right

are for the total number of multiplicative operations, additive operations and array accesses respectively. Below this the total statistics are given which include the LU decomposition, forward elimination and backsubstitution statistics. Therefore these totals are the total number of operations required for solving the equations and therefore are perhaps the best criteria for comparing the algorithms. Finally the fill statistics are presented in a table giving the number of fills for each of the fill categories. The first total corresponds to the usual nonzero fill that is used for comparative purposes [14].

The type structure of the orginal matrix corresponding to the network in Fig. 2.1 is given in Fig. 2.2 as well as the various other information. This provides a basis for comparing to what extent the ordering algorithms improve on these results. Looking at the type structure of the matrix the grouping of topological 1's along the diagonal is clearly seen. This is due to the equation formulation method used [13].

The results of the Markowitz algorithm are given in Fig. 2.3. From these results it is clear that an ordering algorithm can provide dramatic savings. Note how this algorithm orders all the singletons first and then the pivots with the lowest number of off diagonals next. Because this algorithm does not take advantage of the domain type structure, a number of topologicals are ordered way down in the matrix. Although the total number of zero fills is very low, 8 type 1 and 19 type 3 fills occur because domain types are not taken into account.

The results of the minimal fill algorithm are given in Fig. 2.4. The number of fills are lower than the Markowitz (just), but the other operations show a slight increase. Generally one would expect the MF algorithm to outperform the Markowitz algorithm but exceptions are not uncommon. Again one sees the effect on the fills of not taking type into account (particularly type 3).

The results of the domain minimal multiplication algorithm are given in Fig. 2.5. The first thing to note is how this algorithm orders all the toplogical 1's first and thereby eliminates the possibility of type 1

fills. Looking at the operation statistics, one sees a significant improvement over the two previous algorithms, particularly in the LU decomposition. Note the big reduction in the number of divisions, again due to the ordering of the topologicals. Looking at the fill information a surprising result is obtained. The zero fill produced by this algorithm is over double that of the other two, yet the number of operations is still significantly lower. This appears to be a general property of this algorithm, that is it orders to minim ise multiplications at the expense of fills. These extra fills do not seem to imply an increase in the number of operations. This feature only applies if domain types are taken into account for these operations. The case where domain types are not taken into account is discussed in the next section.

The results of the domain minimal fill algorithm are given in Fig. 2.6. From the operation statistics one sees that this algorithm performs on a par with the Markowitz algorithm. However quite surprisingly it produces more zero fill than the untyped algorithms, but does reduce the type 3 fill. Compared to the domain MM algorithm this algorithm halves the fill, but in terms of operations the former algorithm is significantly better. It has been found that the two algorithms generally perform similarly, neither consistently outperforming the other, though as expected the MF always produces less fill. Inspecting the type structure one sees that the topologicals are not all ordered first, even though a domain algorithm is used. It is this feature that motivated the hybrid algorithm.

The results of the domain hybrid algorithm are given in Fig. 2.7. The results are virtually identical to the domain MM ordering, except that slightly fewer operations are needed for the solve stages. The influence of the domain MM algorithm is clearly seen by the ordering of the topologicals. In fact for this example the domain MF algorithm does not seem to have much influence. In general it has been found that the performance of the hybrid algorithm lies between the two algorithms, sometimes one algorithm influencing the ordering more than the other dependent on the domain structure of the matrix.

The previous algorithms all search for pivots in the reverse direction for reasons discussed previously. To evaluate the effect on the algorithms of searching in the forward direction, the above evaluations were repeated with the algorithms operating in the forward mode. These results are presented in Figs. 2.8 to 2.12. From these results it is seen that this change does not have much influence on the domain or minimal fill algorithms, but has a strong influence on the Markowitz algorithm. Similar results for other examples seem to imply that the domain algorithms are more stable than the Markowitz algorithm (that is given the same matrix with a different ordering, the algorithms produce orderings that do not differ much from each other). This has important implications for applications in network analysis as it is desirable to have a method that is independant of the order in which the circuit nodes are numbered.

2.4.2) COMPARISON OF OVERALL METHOD

To evaluate the effectiveness of the domain interpretable code approach and its effectiveness together with the domain hybrid ordering, a number of tests were run. These evaluated particular orderings taking no types, topologicals or full domain types into account. For comparative purposes the statistics for the full matrix solution which does not take type into account are given in Fig. 2.13. Taking into account the sparsity structure of the matrix (but not the type structure) leads to a massive reduction in operations as shown in Fig. 2.14. The number of operations are even further reduced by the Markowitz algorithms, the results of which are given in Figs. 2.15 and 2.16.

Comparing these results with the results obtained with full domain types taken into account (Figs. 2.2, 2.3 and 2.8) it is clear that the domain type approach leads to very large computational savings (in this case virtually halving the number of operations). Therefore the extra effort required in handling the domain types is more than justified by the significant savings that can be achieved.

The performance of the hybrid algorithm when no types are taken into account is given in Fig. 2.17. As expected this algorithm

performs poorly (compared to the Markowitz algortithms) when domain types are not taken advantage of. This is because the domain ordering algorithms are specifically designed to work in conjunction with domain solution algorithms. The effect of taking topologicals into account is shown in Fig. 2.18. From these results it is seen that this step virtually halves the computation. However looking back to Fig. 2.7 where full domain types are taken into account, one sees that these results are further reduced by a factor of 2. These results conclusively show the need for using domain solution methods with the domain ordering algorithms. In fact it is clear that if domain types are not taken into account in the solution operations, the domain ordering algorithms are actually not to be recommended. Note that this is different to the case when the matrix itself is untyped, because then the domain algorithms are equivalent to their untyped counterparts and the solution phase is identical to the usual sparse matrix solution methods.

To compare the overall improvement using the domain techniques, Figs. 2.7 and 2.16 should be compared. The forward Markowitz ordering with no types taken into account is a fair representation of the typical performance of a general sparse matrix code. Comparing the statistics one sees that using the overall domain technique leads to about a factor of 3 saving in computation. This factor has been found to vary from 1 (where the problem is untyped) to as high as 10. Therefore it can be concluded that the domain type approach is a very effective method for problems that consist of domain types, but works equally well for untyped problems.

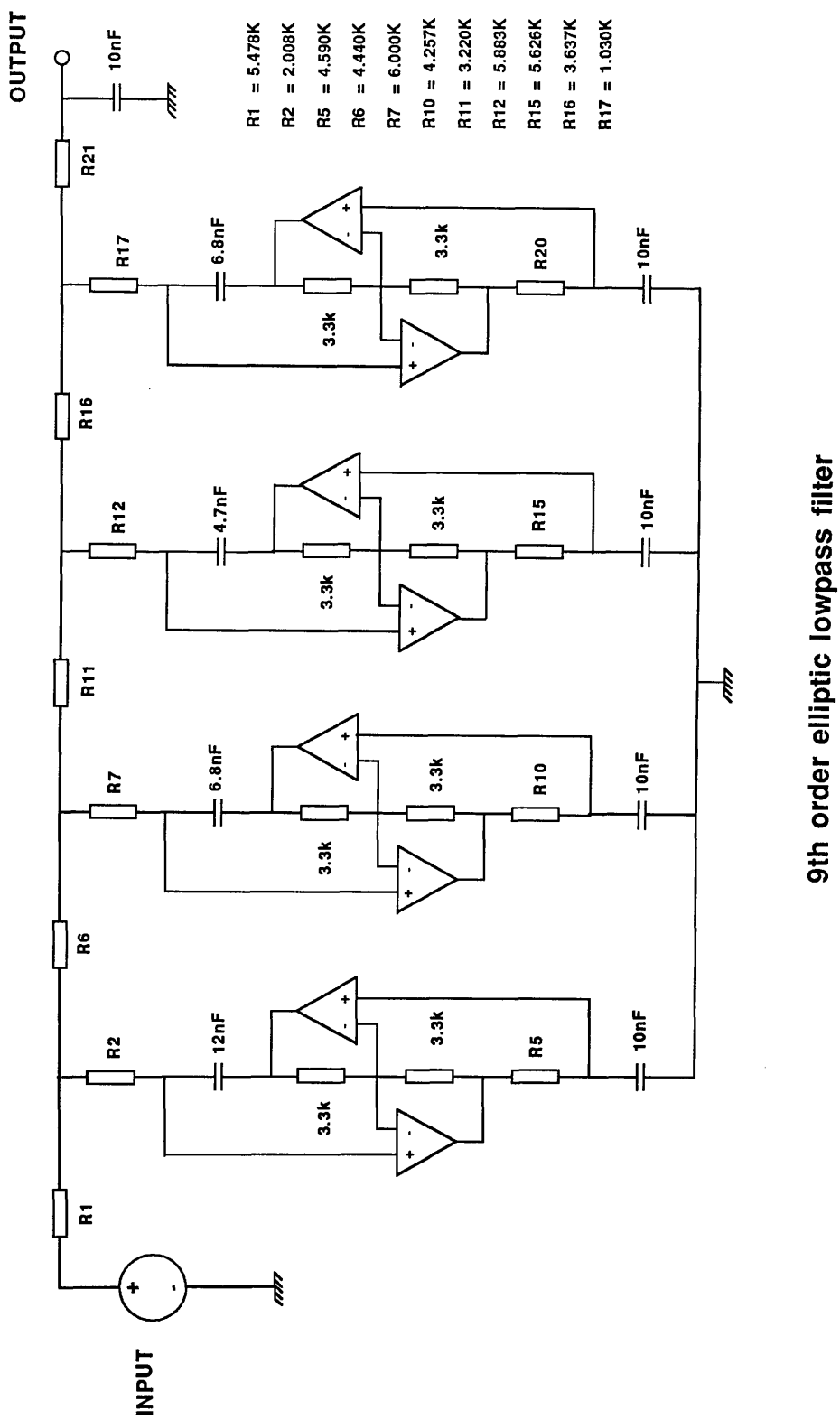
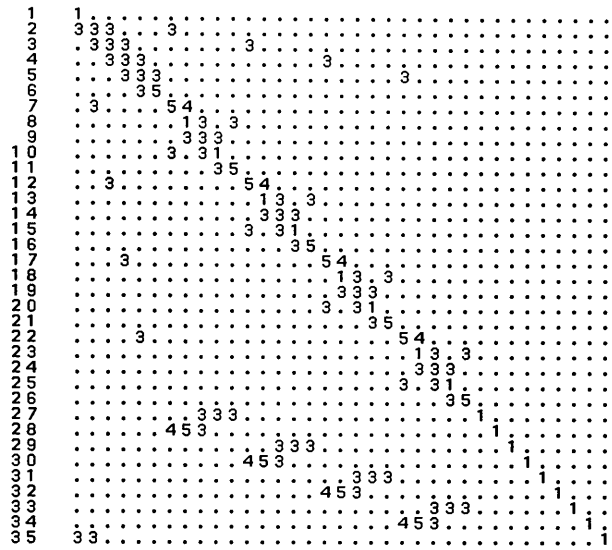


Figure 2.1 Network from which example matrix is derived

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	1196	DIVISIONS	34	TOTAL	1230
ADDITIONS	195	SUBTRACTIONS	906	TOTAL	1101
LOADS	1374	STORES	410	TOTAL	1784

TOTAL STATISTICS

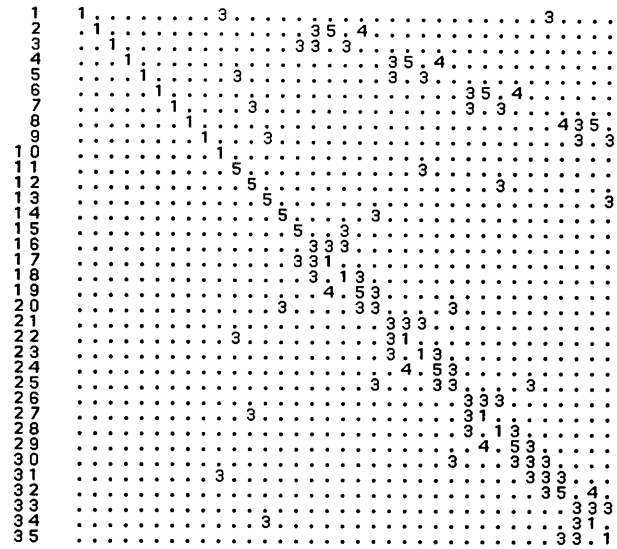
MULTIPLICATIONS	1830	DIVISIONS	34	TOTAL	1864
ADDITIONS	320	SUBTRACTIONS	1381	TOTAL	1701
LOADS	2079	STORES	512	TOTAL	2591

FILL STATISTICS

FILL	ORIGINAL TYPE				
TYPE	0	1	2	3	4
3	24	0	0	0	0
4	0	0	0	0	0
5	148	4	0	12	4
TOTALS	172	4	0	12	4

Figure 2.2 Original ordering

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	152	DIVISIONS	42	TOTAL	194
ADDITIONS	24	SUBTRACTIONS	63	TOTAL	87
LOADS	155	STORES	101	TOTAL	256

TOTAL STATISTICS

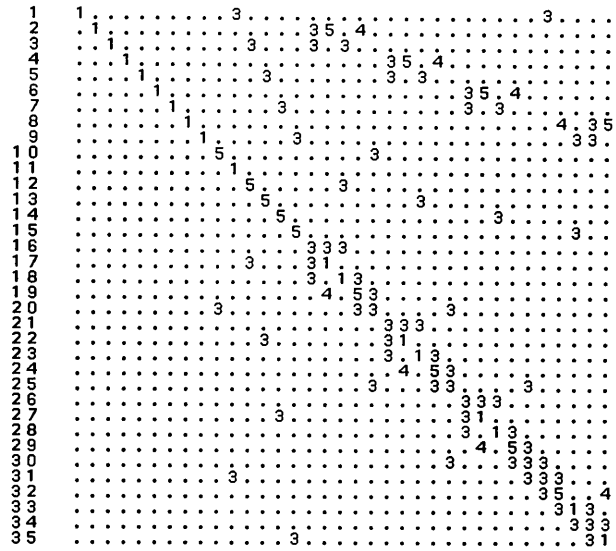
MULTIPLICATIONS	326	DIVISIONS	42	TOTAL	368
ADDITIONS	55	SUBTRACTIONS	200	TOTAL	255
LOADS	362	STORES	175	TOTAL	537

FILL STATISTICS

FILL TYPE	ORIGINAL TYPE				
	0	1	2	3	4
3	3	4	0	0	0
4	0	0	0	0	0
5	8	4	0	19	1
TOTALS	11	8	0	19	1

Figure 2.3 Markowitz ordering

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	155	DIVISIONS	41	TOTAL	196
ADDITIONS	25	SUBTRACTIONS	67	TOTAL	92
LOADS	159	STORES	100	TOTAL	259

TOTAL STATISTICS

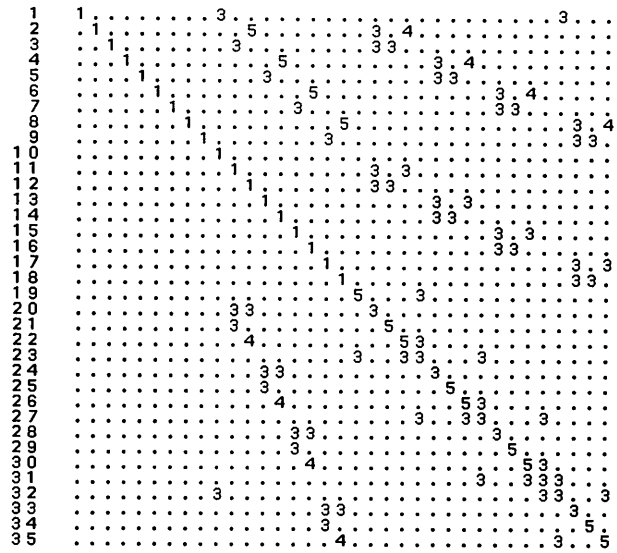
MULTIPLICATIONS	343	DIVISIONS	41	TOTAL	384
ADDITIONS	59	SUBTRACTIONS	213	TOTAL	272
LOADS	382	STORES	178	TOTAL	560

FILL STATISTICS

FILL	ORIGINAL TYPE				
TYPE	0	1	2	3	4
3	3	3	0	0	0
4	0	0	0	0	0
5	7	4	0	21	1
TOTALS	10	7	0	21	1

Figure 2.4 Minimal fill ordering

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	130	DIVISIONS	30	TOTAL	160
ADDITIONS	17	SUBTRACTIONS	68	TOTAL	85
LOADS	166	STORES	74	TOTAL	240

TOTAL STATISTICS

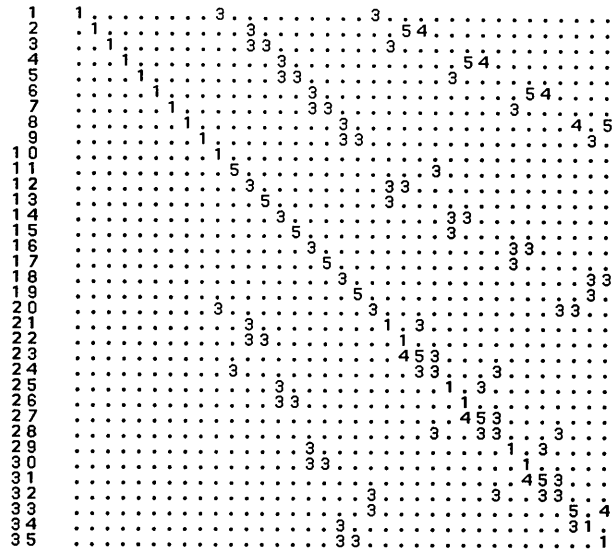
MULTIPLICATIONS	298	DIVISIONS	30	TOTAL	328
ADDITIONS	39	SUBTRACTIONS	210	TOTAL	249
LOADS	382	STORES	146	TOTAL	528

FILL STATISTICS

FILL TYPE	ORIGINAL TYPE				
	0	1	2	3	4
3	12	0	0	0	0
4	8	0	0	0	0
5	4	0	0	12	0
TOTALS	24	0	0	12	0

Figure 2.5 Domain minimal multiplication ordering

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	152	DIVISIONS	41	TOTAL	193
ADDITIONS	23	SUBTRACTIONS	65	TOTAL	88
LOADS	157	STORES	98	TOTAL	255

TOTAL STATISTICS

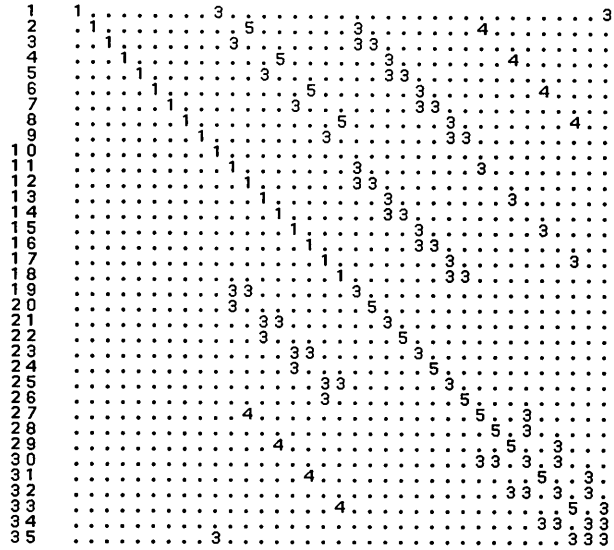
MULTIPLICATIONS	338	DIVISIONS	41	TOTAL	379
ADDITIONS	51	SUBTRACTIONS	213	TOTAL	264
LOADS	386	STORES	176	TOTAL	562

FILL STATISTICS

FILL	ORIGINAL TYPE				
TYPE	0	1	2	3	4
3	4	4	0	0	0
4	0	0	0	0	0
5	9	4	0	13	1
TOTALS	13	8	0	13	1

Figure 2.6 Domain minimal fill ordering

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	130	DIVISIONS	30	TOTAL	160
ADDITIONS	17	SUBTRACTIONS	68	TOTAL	85
LOADS	166	STORES	74	TOTAL	240

TOTAL STATISTICS

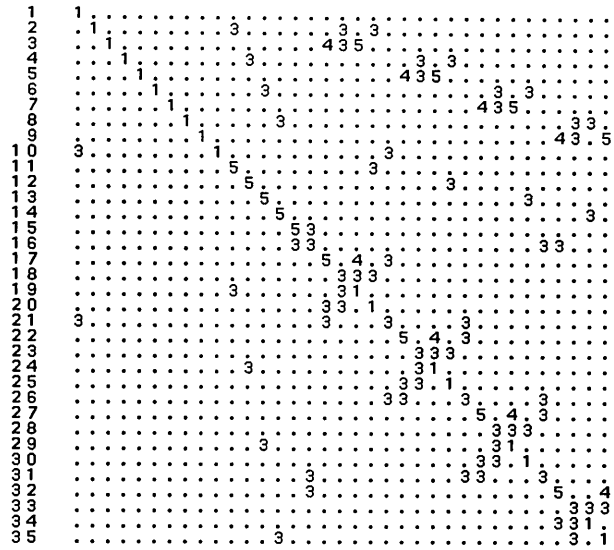
MULTIPLICATIONS	292	DIVISIONS	30	TOTAL	322
ADDITIONS	38	SUBTRACTIONS	207	TOTAL	245
LOADS	375	STORES	144	TOTAL	519

FILL STATISTICS

FILL	ORIGINAL TYPE				
TYPE	0	1	2	3	4
3	12	0	0	0	0
4	8	0	0	0	0
5	4	0	0	12	0
TOTALS	24	0	0	12	0

Figure 2.7 Domain hybrid ordering

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	206	DIVISIONS	42	TOTAL	248
ADDITIONS	32	SUBTRACTIONS	101	TOTAL	133
LOADS	198	STORES	120	TOTAL	318

TOTAL STATISTICS

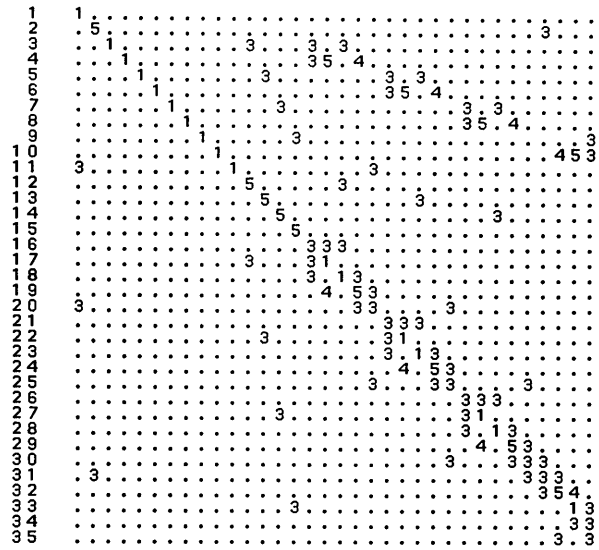
MULTIPLICATIONS	416	DIVISIONS	42	TOTAL	458
ADDITIONS	71	SUBTRACTIONS	260	TOTAL	331
LOADS	444	STORES	202	TOTAL	646

FILL STATISTICS

FILL TYPE	ORIGINAL TYPE				
	0	1	2	3	4
3	0	4	0	0	0
4	0	0	0	0	0
5	19	4	0	16	4
TOTALS	19	8	0	16	4

Figure 2.8 Markowitz ordering (forward)

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	155	DIVISIONS	41	TOTAL	196
ADDITIONS	25	SUBTRACTIONS	67	TOTAL	92
LOADS	159	STORES	100	TOTAL	259

TOTAL STATISTICS

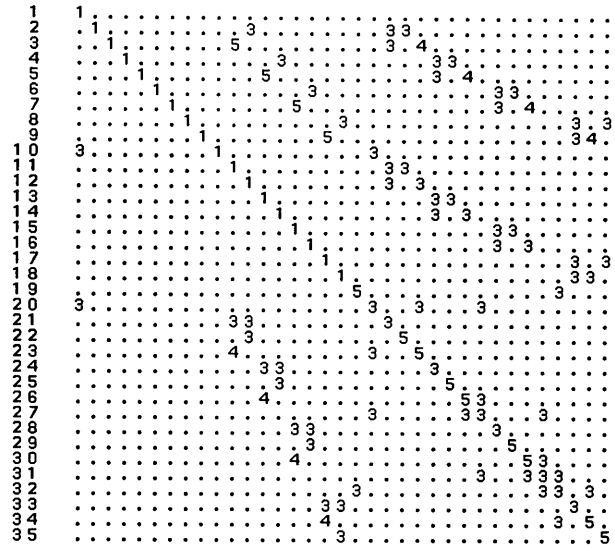
MULTIPLICATIONS	349	DIVISIONS	41	TOTAL	390
ADDITIONS	60	SUBTRACTIONS	214	TOTAL	274
LOADS	390	STORES	182	TOTAL	572

FILL STATISTICS

FILL TYPE	ORIGINAL TYPE				
	0	1	2	3	4
3	3	3	0	0	0
4	0	0	0	0	0
5	7	4	0	21	1
TOTALS	10	7	0	21	1

Figure 2.9 Minimal fill ordering (forward)

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	130	DIVISIONS	29	TOTAL	159
ADDITIONS	15	SUBTRACTIONS	71	TOTAL	86
LOADS	171	STORES	74	TOTAL	245

TOTAL STATISTICS

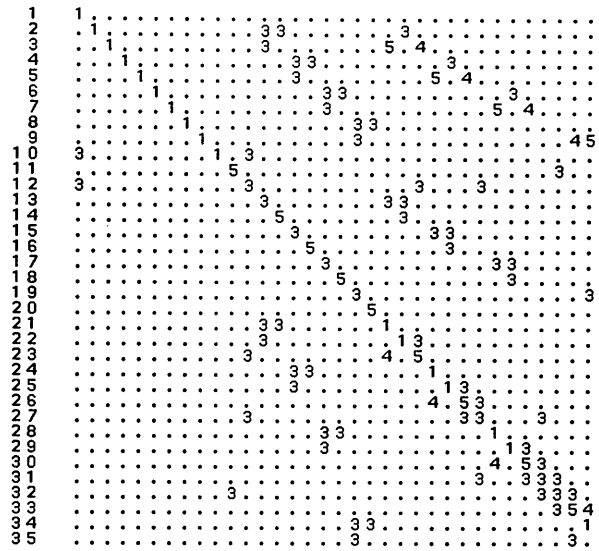
MULTIPLICATIONS	328	DIVISIONS	29	TOTAL	357
ADDITIONS	40	SUBTRACTIONS	230	TOTAL	270
LOADS	427	STORES	158	TOTAL	585

FILL STATISTICS

FILL	ORIGINAL TYPE				
TYPE	0	1	2	3	4
3	14	0	0	0	0
4	7	0	0	0	0
5	5	0	0	9	0
TOTALS	26	0	0	9	0

Figure 2.10 Domain minimal multiplication ordering (forward)

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	152	DIVISIONS	41	TOTAL	193
ADDITIONS	23	SUBTRACTIONS	65	TOTAL	88
LOADS	160	STORES	101	TOTAL	261

TOTAL STATISTICS

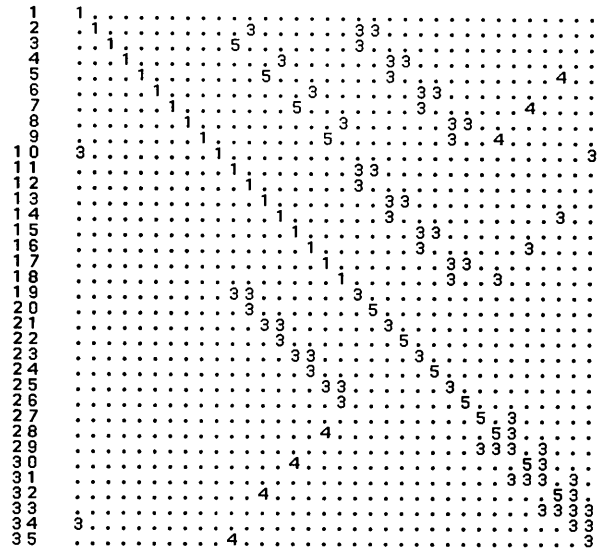
MULTIPLICATIONS	350	DIVISIONS	41	TOTAL	391
ADDITIONS	56	SUBTRACTIONS	216	TOTAL	272
LOADS	400	STORES	185	TOTAL	585

FILL STATISTICS

FILL	ORIGINAL TYPE				
TYPE	0	1	2	3	4
3	4	4	0	0	0
4	0	0	0	0	0
5	9	4	0	16	1
TOTALS	13	8	0	16	1

Figure 2.11 Domain minimal fill ordering (forward)

TYPE STRUCTURE BEFORE LU DECOMPOSITION



LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	130	DIVISIONS	30	TOTAL	160
ADDITIONS	17	SUBTRACTIONS	68	TOTAL	85
LOADS	166	STORES	74	TOTAL	240

TOTAL STATISTICS

MULTIPLICATIONS	298	DIVISIONS	30	TOTAL	328
ADDITIONS	39	SUBTRACTIONS	210	TOTAL	249
LOADS	384	STORES	148	TOTAL	532

FILL STATISTICS

FILL	ORIGINAL TYPE				
TYPE	0	1	2	3	4
3	12	0	0	0	0
4	8	0	0	0	0
5	4	0	0	12	0
TOTALS	24	0	0	12	0

Figure 2.12 Domain hybrid ordering (forward)

LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	57190	DIVISIONS	70	TOTAL	57260
ADDITIONS	14315	SUBTRACTIONS	41650	TOTAL	55965
LOADS	57122	STORES	2382	TOTAL	59504

TOTAL STATISTICS

MULTIPLICATIONS	62090	DIVISIONS	70	TOTAL	62160
ADDITIONS	15540	SUBTRACTIONS	45255	TOTAL	60795
LOADS	62022	STORES	2520	TOTAL	64542

Figure 2.13 Statistics for full matrix (no types taken into account)

LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	1966	DIVISIONS	70	TOTAL	2036
ADDITIONS	509	SUBTRACTIONS	1302	TOTAL	1811
LOADS	1814	STORES	502	TOTAL	2316

TOTAL STATISTICS

MULTIPLICATIONS	2790	DIVISIONS	70	TOTAL	2860
ADDITIONS	715	SUBTRACTIONS	1866	TOTAL	2581
LOADS	2618	STORES	604	TOTAL	3222

Figure 2.14 Statistics for original matrix (no types taken into account)

LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	442	DIVISIONS	70	TOTAL	512
ADDITIONS	128	SUBTRACTIONS	171	TOTAL	299
LOADS	326	STORES	192	TOTAL	518

TOTAL STATISTICS

MULTIPLICATIONS	686	DIVISIONS	70	TOTAL	756
ADDITIONS	189	SUBTRACTIONS	346	TOTAL	535
LOADS	568	STORES	266	TOTAL	834

Figure 2.15 Statistics for Markowitz ordering (no types taken into account)

LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	502	DIVISIONS	70	TOTAL	572
ADDITIONS	143	SUBTRACTIONS	210	TOTAL	353
LOADS	372	STORES	206	TOTAL	578

TOTAL STATISTICS

MULTIPLICATIONS	790	DIVISIONS	70	TOTAL	860
ADDITIONS	215	SUBTRACTIONS	410	TOTAL	625
LOADS	658	STORES	288	TOTAL	946

Figure 2.16 Statistics for forward Markowitz ordering (no types taken into account)

LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	558	DIVISIONS	70	TOTAL	628
ADDITIONS	157	SUBTRACTIONS	242	TOTAL	399
LOADS	410	STORES	218	TOTAL	628

TOTAL STATISTICS

MULTIPLICATIONS	818	DIVISIONS	70	TOTAL	888
ADDITIONS	222	SUBTRACTIONS	433	TOTAL	655
LOADS	668	STORES	288	TOTAL	956

Figure 2.17 Statistics for hybrid ordering (no types taken into account)

LU DECOMPOSITION STATISTICS

MULTIPLICATIONS	354	DIVISIONS	34	TOTAL	388
ADDITIONS	97	SUBTRACTIONS	200	TOTAL	297
LOADS	290	STORES	98	TOTAL	388

TOTAL STATISTICS

MULTIPLICATIONS	610	DIVISIONS	34	TOTAL	644
ADDITIONS	161	SUBTRACTIONS	390	TOTAL	551
LOADS	546	STORES	168	TOTAL	714

Figure 2.18 Statistics for hybrid ordering (topologicals taken into account)

REFERENCES FOR CHAPTER TWO

[1] I.S. Duff, "A survey of sparse matrix research", Proc. IEEE, Vol. 65, No. 4, pp.500–535, 1977.

[2] J. Vlach and K. Singhal, "Computer methods for circuit analysis and design", Van Nostrand Reinhold, 1983.

[3] B. Dembart and A.M. Erisman, "Hybrid sparse–matrix methods", IEEE Trans. Circuit Theory, Vol. CT–20, No. 6, pp.641–649, 1973.

[4] G. D. Hachtel, R.K. Brayton and F.G. Gustavson, "The sparse tableau approach to network analysis and design", IEEE Trans. CT, Vol. CT–18, No. 1, pp.101–113, Jan. 1971.

[5] R.A. Willoughby, Ed., "Proceedings of the symposium on sparse matrices and their applications", IBM Res., Yorktown Heights, N.Y., IBM Report RA1 (#11707), 1969.

[6] F.G. Gustavson, W.M. Liniger and R.A. Willoughby, "Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations", J. ACM, Vol. 17, pp.87–109, 1970.

[7] A. Chang, "Application of sparse matrix methods in electric power systems analysis", in [5, pp.113–122].

[8] H. Lee, "An implementation of Gaussian elimination for sparse systems of linear equations", in [5, pp.75–83].

[9] A. Douglas, "Examples concerning efficient strategies for Gaussian elimination", Computing, Vol. 8, pp.382–394, 1971.

[10] H.M. Markowitz, "The elimination form of the inverse and its application to linear programming", Management Science, Vol. 3, pp.255–269, 1957.

[11] W.F. Tinney and J.W. Walker, "Direct solutions of sparse

network equations by optimally ordered triangular factorization", Proc. IEEE, Vol. 55, pp.1801−1809, 1967.

[12] R.D. Berry, "An optimal ordering of electronic circuit equations for a sparse matrix solution", IEEE Trans. CT, Vol. CT−18, No. 1, pp.40−50, Jan. 1971.

[13] L.B. Wolovitz, "Improved techniques for time−domain analysis of switched capacitor networks", M.Sc. Thesis, University of Hull, 1986.

[14] I.S. Duff and J.K. Reid, "A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination", J. Inst. Math. Appl., Vol. 14, pp.281−291, 1974

CHAPTER THREE

TIME DOMAIN SOLUTION OF LINEAR NETWORKS

3.1) INTRODUCTION

This chapter considers the problem of calculating the time domain response of linear, lumped time invariant networks. Many methods of computing the time response of linear networks have been proposed **[1]**, but few have received widespread acceptance. This is because the problem poses great computational difficulties and few methods can provide accurate and reliable solutions whilst claiming to be computationally efficient. This introduction discusses the underlying difficulties in calculating the time response and methods that have been proposed to overcome them.

Using modern tableau equation formulation methods **[2]**, a linear network is represented by the algebraic– differential system,

$$C\,\dot{x}(t) + G\,x(t) = w(t) \qquad x(0) = X_0 \qquad (3.1)$$

where x(t) is the unknown response vector
w(t) is the excitation vector
C and G are constant real matrices

Matrices C and G are generally sparse and therefore any method striving for maximum efficiency has to be capable of taking advantage of this fact. Generally matrix C is singular and therefore equation (3.1) cannot be written in normal form, on which many methods depend. A state–space equation formulation method can be used (which produces normal form equations), but then the simplicity of the tableau approaches and the sparsity of the equations is lost. Even in cases where matrix C is not singular, the formation of $C^{-1}G$ is not desirable as again sparsity is lost because generally the inverse of a sparse matrix is not sparse.

The problem then is to calculate the response vector x(t) at intervals (grid–points) over the time period of interest. These intervals need not necessarily be equal and the time difference between successive

grid–points is called the time–step. The analytic solution to equation (3.1) is derived in section 3.2 but is not practical from a computational point of view as it involves a convolution which is difficult and computationally expensive to compute. Furthermore the convolution is time dependent and therefore needs to be re–evaluated at each time–step. However the analytic solution approach does provide the basis for a number of methods which are termed the inverse Laplace transform methods. These methods together with the most common approach to solving (3.1), numerical integration methods, are discussed below and compared on the basis of certain requirements.

Two of the main requirements are accuracy and stability. Clearly it is desirable to have a method that can meet prescribed accuracy specifications, even better if this accuracy can be controlled. Generally most methods can improve accuracy by decreasing the solution step–size, increasing the order of the approximations used in the method, or both. However decreasing the step–size and increasing the order can dramatically increase the computation time of a method and therefore the trade–off between accuracy and computation time is probably the major criteria by which a method is judged, as it is this factor that determines whether a method is useful or not.

The requirement of stability is a requirement of reliability; that is if the system itself is stable, then the method must ensure a stable solution, irrespective of the step–size used. This definition of stability, called A–stability, was introduced by Dahlquist **[3]**. Many of the methods discussed do not meet this requirement, but can still ensure stable solutions if the step–size is kept smaller than some constant (which is determined by the characteristics of the system).

A major difficulty posed in solving equation (3.1) is when the system has a wide spread of eigenvalues, known as a 'stiff' system. This situation commonly arises in linear networks due to parasitic elements and a large range of time–constants in the networks. Stiff systems pose numerical difficulties because the smallest eigenvalues (largest time–constants) determine the dominant network response and thus the total length of time for which the solution must be calculated to

characterise this response, but for many methods the largest eigenvalue (smallest time–constant) controls the maximum allowable step–size that insures numerical stability. So for example if there is a 1000 to 1 ratio of largest to smallest time–constant, then a method that does not overcome this 'minimum time–constant barrier' [1], needs about 1000 more time–steps to obtain the same solution as a method which does.

However, even methods that overcome this problem face further difficulties. Dahlquist [3] has shown that the maximum order of an A–stable multistep integration method is 2, and that the trapezoidal method has the smallest error coefficient amongst all order 2 A–stable methods. This maximum order of 2 severely limits the accuracy of these methods as is shown in the comparison in section 3.6.2 with SPICE2 [4], which uses the trapezoidal method. A novel method of order 2 has been proposed [5] that improves on the efficiency of the trapezoidal method for very stiff problems, but nonetheless is limited by 2nd order accuracy.

To try and overcome this limitation, Gear relaxed the requirement of A–stability by introducing the concept of stiff stability [6], and showed that a set of methods up to order 6 are stiffly stable. In order to benefit from the higher accuracy avaliable, these methods, known as backward differentiation formulas (BDF), are implemented in a variable time–step, variable order scheme. An algorithm is then needed that attempts to optimally select the step–size and order that maximises the accuracy whilst minimising the computation time. These methods have provided the basis for a number of successful 'stiff–solver' packages, some of which have been specially modified to exploit any structure inherent in the system [7]. However in practice the higher order methods are not used [8] and therefore higher order methods are potentially far more efficient.

Compared to the above limitations of the numerical integration methods, the inverse Laplace transform methods have many advantages [9]. These methods compute the inverse Laplace transform by evaluating the response of the system in the complex Laplace domain and using a quadrature formula. They are applicable to stiff systems,

systems with multiple poles, are A− stable and equivalent to high order integration methods (> 6). The method was first proposed for homogeneous systems [10], which though very efficient is of limited use. A method for nonhomogeneous systems was presented in [9] which uses the same method for numerical inversion of the Laplace transform. The accuracy of these methods decreases with increasing time, therefore a technique of 'resetting' the problem was developed in [9]. This technique effectively makes the method equivalent to numerical integration methods, though without their limitations. However the method as presented only allows for non− periodic excitations, which is a major limitation. Even though the method has very high order, it requires a number of evaluations of the system response in the frequency domain at each time− step, which makes the method less efficient than a well implemented BDF method [2].

A new approach based on the above methods is presented in this chapter. This method shares all the advantages of the inverse Laplace transform methods, but overcomes the limitations of non− periodic excitations and inefficiency. The method uses the inverse Laplace transform and the stepping technique, hence the name Stepping Inverse Laplace Transform (SILT) method. The method hinges on the use of polynomial approximations for the excitations and efficient techniques for evaluating the inverse Laplace transform, which are discussed in detail. The results of this method are then presented and the accuracy and stability are evaluated. The SILT method is then compared with other methods for accuracy and efficiency. Finally the extension of the method to periodically switched linear networks is discussed.

3.2) STEPPING INVERSE LAPLACE TRANSFORM METHOD

Taking the Laplace transform of equation (3.1),

$$(sC+ G)\ X(s) \ = \ C\ X_0 + W(s) \tag{3.2}$$

after re–arranging,

$$X(s) \ = \ [sC + G]^{-1}\{\ C\ X_0 + W(s)\ \} \tag{3.3}$$

and taking the inverse transform,

$$x(t) \ = \ P(t)\ C\ x(0) + L^{-1}\{\ [sC + G]^{-1}W(s)\ \} \tag{3.4}$$

where $P(t) \ = \ L^{-1}\{\ [sC + G]^{-1}\}$

henceforth called the extended state transition (EST) matrix.

Analytic methods continue one step further and write equation (3.4) as,

$$x(t) \ = \ P(t)\ C\ x(0) + \int_0^t P(\tau)\ w(t - \tau)\ d\tau \tag{3.5}$$

As discussed in the introduction, the major difficulty is then the evaluation of the convolution in equation (3.5) which is time–dependent and therefore renders this formulation impractical.

The approach developed here is to approximate the excitation w(t) in equation (3.1) by a *m*th order polynomial

$$P_m(t) = \sum_{k=0}^{m} \alpha_k t^k \tag{3.6}$$

giving the new system differential equation

$$C\ \dot{x}(t) + G\ x(t) \ = \ \sum_{k=0}^{m} \alpha_k t^k \qquad x(0) = X_0 \tag{3.7}$$

The motivation for this approach is that the inverse Laplace transform in equation (3.4) may then be readily computed. The approximation need not necessarily be a polynomial approximation. A rational polynomial or even trigonometric approximation could be used, as long as the inverse Laplace transform may be readily determined. The polynomial approximation was chosen because it has a well characterised approximation error and the inverse Laplace transform can be computed exactly.

Taking the Laplace transform of equation (3.7),

$$(sC+ G)\ X(s) \;=\; C\,X_0 + \sum_{k=0}^{m} \frac{\alpha_k k!}{s^{k+1}} \qquad (3.8)$$

after re-arranging,

$$X(s) \;=\; [sC + G]^{-1} \left\{ C\,X_0 + \sum_{k=0}^{m} \frac{\alpha_k k!}{s^{k+1}} \right\} \qquad (3.9)$$

Taking the inverse Laplace transform,

$$x(t) \;=\; P(t)\ C\ x(0) \;+\; \sum_{k=0}^{m} \alpha_k B_k(t) \qquad (3.10)$$

where $B_k(t) \;=\; L^{-1}\{\ [sC + G]^{-1}\ \frac{k!}{s^{k+1}}\ \}$

henceforth called the excitation response (ER) matrix.

To evaluate the time response of x(t) for $t \in [0,T]$ the time axis is divided into equal steps Δt, where $t = n\Delta t$.

From equation (3.10) we then have,

$$x(n\Delta t) \;=\; P(n\Delta t)\ C\ x(0) \;+\; \sum_{k=0}^{m} \alpha_k B_k(n\Delta t) \qquad (3.11)$$

In lumped linear networks, time zero can be arbitarily selected by taking into account the initial conditions of the network, which effectively takes into account all previous history of the network, thus

$$x(n\Delta t + \Delta t) \;=\; P(\Delta t)\ C\ x(n\Delta t) \;+\; \sum_{k=0}^{m} \alpha_k^n\ B_k(\Delta t) \qquad (3.12)$$

where α_k^n are the coefficients of the polynomial approximation of w(t) in the interval $[n\Delta t,\ n\Delta t+ \Delta t]$.

Noting that $P(\Delta t)$ and $B_k(\Delta t)$ are independent of n (and therefore constant), equation (3.12) gives the recurrence relation,

$$x((n+1)\Delta t) \;=\; P\ x(n\Delta t) \;+\; \sum_{k=0}^{m} \alpha_k^n\ B_k \qquad (3.13)$$

where $P = P(\Delta t)C$ and $B_k = B_k(\Delta t)$

Inspection of equation (3.13) shows that m+2 matrix–vector multiplications are required at each step. Normally the excitation

vector has only a few nonzero entries (typically 1) and therefore only one column of the B_k matrix is required, corresponding to each nonzero entry. The computation is then reduced to one matrix–vector multiplication and m+1 vector–scalar multiplications, which compares very favourably with other methods.

3.3) POLYNOMIAL APPROXIMATION

From equation (3.13), a procedure is required to determine the coefficients α_k^n which determine the polynomials

$$P_m(t) = \sum_{k=0}^{m} \alpha_k^n \, t^k \qquad t \in [n\Delta t, (n+1)\Delta t] \qquad (3.14)$$

such that $P_m(t)$ is a 'best fit' in some sense to w(t) in the specified interval. Depending on the definition of 'best fit' various polynomial fitting procedures may be used, for example polynomial splines or Chebychev polynomials. The method derived here is based on fitting polynomials through the m+1 equispaced points,

$$n\Delta t, \; n\Delta t+\Delta t/m, \; n\Delta t+2\Delta t/m, \; \ldots\ldots, \; n\Delta t+\Delta t$$

such that they are exact at these points, that is,

$$P_m(t + \frac{k\Delta t}{m}) = w(t + \frac{k\Delta t}{m}) \qquad k = 0,\ldots,m \qquad (3.15)$$

This approach has the advantage of being simpler to implement than the other methods as well as having a correspondence with numerical integration methods e.g. the trapezoidal method (m=1).

Define the divided differences

$$\Delta^m f_0 = f_m - \begin{bmatrix} m \\ 1 \end{bmatrix} f_{m-1} + \begin{bmatrix} m \\ 2 \end{bmatrix} f_{m-2} + \ldots\ldots + (-1)^m f_0$$

where $f_k = w(t + \frac{k\Delta t}{m})$ (3.16)

and $\begin{bmatrix} \; \end{bmatrix}$ denotes the binomial coefficients.

Using the Newton–Gregory interpolation formula,

$$P_m(t) = f_0 + \begin{bmatrix} s \\ 1 \end{bmatrix} \Delta f_0 + \begin{bmatrix} s \\ 2 \end{bmatrix} \Delta^2 f_0 + \ldots\ldots + \begin{bmatrix} s \\ m \end{bmatrix} \Delta^m f_0$$

where $s = \frac{t - t_0}{h}$ and $h = \frac{\Delta t}{m}$ (3.17)

Writing equation (3.17) in compact notation and expanding the binomial terms,

$$P_m(t) = \sum_{k=0}^{m} \begin{bmatrix} s \\ k \end{bmatrix} \Delta^k f_0$$

$$= \sum_{k=0}^{m} \left[\prod_{i=0}^{k-1} (t-ih) \right] \frac{\Delta^k f_0}{h^k k!} \qquad (3.18)$$

Collecting terms in powers of t, equation (3.18) becomes

$$P_m(t) = \sum_{k=0}^{m} \left[\sum_{i=0}^{m} \gamma_{ki} f_i \right] \frac{t^k}{h^k} \qquad (3.19)$$

where γ_{ik} are constants independent of h or t.

From equation (3.19) we then finally have

$$\alpha_k^n = \sum_{i=0}^{m} \gamma_{ki} f_i$$

$$= \sum_{i=0}^{m} \gamma_{ki} w(n\Delta t + \frac{i\Delta t}{m}) \qquad (3.20)$$

3.4) DETERMINING THE γ COEFFICIENTS

The γ coefficients can be computed by noting that there is a recurrence relationship

$$P_m(t) = P_{m-1}(t) + D_m \qquad (3.21)$$

where $D_m = \begin{bmatrix} s \\ m \end{bmatrix} \Delta^m f_0$

so that the problem reduces to one of finding for D_m the coefficients of t in terms of f_k.

Define coefficients b_k and p_k by

$$\Delta^m f_0 = \sum_{k=0}^{m} b_k f_k$$

$$\begin{bmatrix} s \\ m \end{bmatrix} = \sum_{k=0}^{m} p_k \frac{t^k}{k!} \qquad (3.22)$$

These coefficients are easily computed using the following algorithms.

```
t1 := (-1)^m
t2 := m
b_0 := t1
FOR j := 1 TO m DO                                    (3.23)
  t1 := -t1 × t2 / j
  t2 := t2 - 1
  b_j := t1
END
```

```
p_m := 1
FOR j := m-1 DOWNTO 1 DO                              (3.24)
  p_j := (1-m) × p_j + p_j-1
END
```

The γ coefficients of order m are then obtained by adding the cross–product of the p_k and b_k coefficients to the γ coefficients of order m−1. If the γ coefficients are multiplied through by m! then the algorithms may be implemented using integer arithmetic. Combining the division by k! in equation (3.22) and the premultiplication of the coefficients by m!, the algorithm for calculating the γ coefficients is then,

```
t1 := m!
FOR k := 1 TO m DO
  calculate b_k coefficients of order k
  calculate p_k coefficients of order k
  t1 := t1 / k
  FOR i := 1 TO k DO                                  (3.25)
    FOR j := 0 TO k DO
      γ_ij := γ_ij + p_i × b_j × t1
    END
  END
END
```

The γ coefficients for orders 1 through 4 are given in Table 3.1.

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

$$\frac{1}{2}\begin{bmatrix} 2 & 0 & 0 \\ -3 & 4 & -1 \\ 1 & -2 & 1 \end{bmatrix}$$

$$\frac{1}{6}\begin{bmatrix} 6 & 0 & 0 & 0 \\ -11 & 18 & -9 & 2 \\ 6 & -15 & 12 & -3 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

$$\frac{1}{24}\begin{bmatrix} 24 & 0 & 0 & 0 & 0 \\ -50 & 96 & -72 & 32 & -6 \\ 35 & -104 & 114 & -56 & 11 \\ -10 & 36 & -48 & 28 & -6 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix}$$

Table 3.1 Gamma coefficients for orders 1 to 4

These tables have many interesting properties, two of which are discussed here. The first property is

$$\sum_{k=0}^{m} \gamma_{ik} = 0 \qquad i = 1..m \qquad (3.26)$$

that is (except for the first row) rows of the table sum to zero. This property is expected as given a constant excitation i.e. $f_0 = f_1 = \ldots = f_m$, the higher order terms for the polynomial approximation should be zero i.e. $\alpha_k^n = 0$ for $k > 0$.

The second property is

$$\sum_{i=0}^{m} \gamma_{ik} m^i = 0 \qquad k = 0..m-1$$

$$\sum_{i=0}^{m} \gamma_{im} m^i = 1 \qquad (3.27)$$

This property is derived by evaluating equation (3.19) for $t = \Delta t$, which then evaluates to exactly f_m, which is the exact result.

3.5) COMPUTING THE INVERSE LAPLACE TRANSFORM

The SILT method presented in section 3.2 depends on having a reliable, efficient and accurate method for computing the extended state transition matrix

$$P(t) = L^{-1}\{ [sC + G]^{-1} \} \qquad (3.28)$$

and the excitation response matrices

$$B_k(t) = L^{-1}\{ [sC + G]^{-1} \frac{k!}{s^{k+1}} \} \qquad (3.29)$$

In a review of methods for computing the matrix exponential [11], Moler and Van Loan describe the difficulty of finding reliable and efficient methods. Out of 19 different methods reviewed only one (scaling and squaring with diagonal Padé approximation) [12] is recommended as reliable. However most of the methods in the literature are unable to compute the extended state transition matrix where matrix C is singular, let alone the excitation response matrix.

The method developed here uses the I_{MN} approximant [13] and has been found to be both reliable and efficient. The use of the I_{MN}

approximant to compute the matrix exponential was first reported in [14]. Using the I_{MN} approximant to compute the extended state transition matrix was first suggested in [15] and was successfully implemented, though in a different way to the method presented here.

The approach used here is based on the derivation in [9], which uses a numerical approximation of the Laplace Transform inversion integral

$$v(t) = \frac{1}{2\pi j} \int_{c-j\infty}^{c+j\infty} V(s)\, e^{st}\, ds \qquad (3.30)$$

The quadrature approximation is derived in [9] as

$$\hat{v}(t) = \frac{1}{t} \sum_{i=1}^{M} K_i V(z_i/t) \qquad (3.31)$$

where K_i and z_i are tabulated complex constants. Various strategies may be used to derive the constants K_i and z_i, the optimum set is derived from the Padé approximation of the exponential function. The I_{MN} constants for M=1,..,10 are tabulated in [16]. A program for generating the constants to higher precision is presented in [17] and constants for various orders are tabulated in [2, p.286].

Inspection of these tables reveals that for M even the constants occur in M/2 complex conjugate pairs. Therefore the computation required can be halved by using

$$\hat{v}(t) = \frac{1}{t} \sum_{i=1}^{M/2} \mathrm{Re}\left[\, 2K_i V(z_i/t) \,\right] \qquad (3.32)$$

Applying equation (3.32) to (3.28), we get the extended state transition matrix approximation,

$$P(t) \approx \sum_{i=1}^{M/2} \mathrm{Re}\left[\, 2K_i/t \, \left[z_i/tC + G\right]^{-1} \right] \qquad (3.33)$$

and applying (3.32) to (3.29), we get the excitation response matrix approximation,

$$B_k(t) \approx \sum_{i=1}^{M/2} \mathrm{Re}\left[\frac{2K_i/t \; k!}{(z_i/t)^{k+1}} \left[z_i/tC + G\right]^{-1} \right] \qquad (3.34)$$

It has been shown [9] that the approximation (3.31) is exact for a polynomial of degree up to $N+M+1$. To achieve maximum accuracy, whilst avoiding roundoff error problems, $M=10$ and $N=8$ were selected for the I_{MN} constants. Therefore equation (3.34) is exact for $k=0,..,19$, which is more than adequate for practical use.

3.5.1) EFFICIENT IMPLEMENTATION STRATEGIES

If equation (3.33) were implemented directly using full matrix techniques, it would require $2MN^3+2MN^2$ flops where N is the size of the matrices. This cost is obviously excessive and therefore sparse matrix techniques must be used.

An efficient method for computing the inverse of a sparse matrix was first suggested by Takahasi, et al., [18] and refined in [19]. However this method has 3 drawbacks when used to implement equation (3.33). The first is that it is still necessary to multiply the inverse by K_i (which requires $2MN^2$ flops) though the method can be easily adapted to absorb this factor, only requiring an extra 2MN flops. The second disadvantage is that the method cannot selectively compute columns of the inverse, which is required for evaluating equation (3.34). The third disadvantage is that the method is not suited to interpretive code generation and therefore cannot take advantage of domain types as presented in Chapter Two.

The method used in this work is based on the direct approach of computing the LU factorisation and then backsubstituting the unit vector to get a column of the inverse. This approach has the major advantage that the methods developed in Chapter Two can be used to perform the LU decomposition and backsubstitution, thus achieving near optimal efficiency. Furthermore the multiplication by K_i is achieved by multiplying the unit vector by K_i which does not require any multiplications, thus saving $2MN^2$ flops. Finally as the method computes a column at a time, only columns that are required need be calculated, leading to further savings.

The computational cost of the method is difficult to ascertain as it is dependant on the sparsity of the matrices in (3.33), which vary

substantially from problem to problem. However timing results indicate that the method generally exhibits $O(N^2)$ computational complexity.

The method used to compute equation (3.34) is similar to that described above, except for the method of including the factor $(z_i/t)^{-(k+1)}$. Two different approaches are used. The first is to combine this factor with K_i and proceed as above. The second approach uses the result computed for $k-1$ and divides this by z_i/t, giving the required result. The latter approach requires 4N flops whereas the former depends on the number of flops required for the backsubstitution and is therefore problem dependant. The cheapest method is then selected as the appropriate technique. In practice it is found that the first method is generally superior, though the difference between the two is insignificant compared to the overall cost of the method.

3.6) RESULTS

The SILT method was implemented in the computer program FOOLSCAP, using 16 digit double precision arithmetic throughout. A number of different and independent tests were used to verify the implemetation of the theory and determine the accuracy and stability of the method. These tests were carried out on a number of networks that differed in size, complexity and range of time-constants. In the following SILTm is used to denote the SILT method that uses a polynomial approximation of degree m.

3.6.1) ACCURACY AND STABILITY

The first test was to evaluate the accuracy of the extended state transition matrix approximation. This was done by evaluating the time response of the network from specified initial conditions and with no excitations. Where possible analytic solutions were used for comparison, otherwise SPICE2 was used. However it was soon found that the low accuracy of SPICE2 meant that these results could not be used for objective comparison, but rather subjective comparison i.e. visually inspecting that the responses compared favourably. The poor performance of SPICE2 is discussed in a later section. The tests quickly verified that the method is indeed very accurate (18th order)

and efficient. To try and verify the order of the approximation, the tests were run with different step−sizes and the errors compared. However because of the high order, the change in accuracy is then proportional to the change in step−size raised to the power of 18, which is orders of magnitude change. In practice it was found that roundoff errors then dominate and are then the limiting factor of the maximum accuracy attainable. For most networks the accuracy attained is about 11 decimal places, though dropping to between 9 and 10 for large networks.

The second test was to evaluate the accuracy of the SILT method for polynomial excitations. The excitations used were degree 0 (step input) and degree 1 (ramp input). The results for these inputs should be exact (to within the 18th order approximation of the ER vectors). Again analytic solutions were used where available, otherwise SPICE2 was used for subjective verification. The results again showed that roundoff errors dominate for small steps, limiting the maximum achievable accuracy to about 10 decimals. The method again proved to be highly accurate and efficient, in fact the step−size required for producing the plots of the responses always provided more than sufficient accuracy, therefore the computation time was not determined by accuracy considerations, but the number of points to produce a visually smooth curve!

For the case of a step input, the SILT0 method is of sufficient order to provide the exact response and higher order methods are then superfluous as $\alpha_k^n = 0$ for $k > 0$ (see equation 3.26). Similarly for a ramp input the SILT1 method provides the exact response and higher order methods are again superfluous as $\alpha_k^n = 0$ for $k > 1$. These two cases were tested in FOOLSCAP by printing out the α_k^n, which verified the above, and by comparing the results using the higher order methods.

The third set of tests was to evaluate the accuracy of the SILT method for non−polynomial excitations using a sinusoidal input. In this case accuracy is controlled both by step−size and the order of the method. The Newton−Gregory interpolation formula has an error term of $O(h^{m+1})$, and therefore we expect the error of the

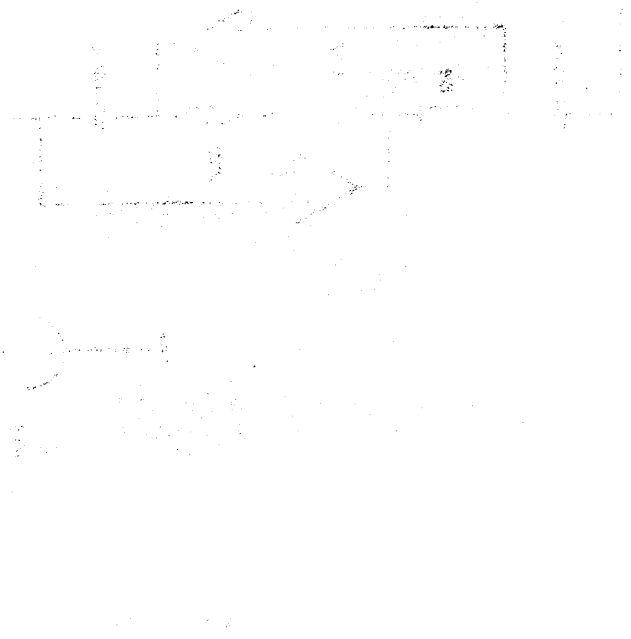
polynomial approximation in equation (3.12) to be $O((\Delta t/m)^{m+1})$. The first stage was to verify that the method did in fact work and that accuracy increased for increasing order and decreasing step–size. Again analytic solutions were used where available, otherwise SPICE2 was used for subjective verification of the response. A further check that was found to be useful was to monitor the response at the input node. This response should be identical to the input excitation and therefore does not require an analytic solution or results from some other source. These tests verified the implementation of the SILT method and were found to be very accurate and efficient. As in previous tests, it was found that roundoff errors dominate for small step–sizes, but this was further complicated by the extra degree of freedom provided by the selection of different orders. For a fixed step–size it was found that for high order methods, roundoff errors become dominant and were found to grow without bound. The maximum order that was found to be reliable is order 10, though the results produced by order 10 and order 9 are generally indistinguishable. Therefore it is suggested that orders 0 through 9 be used in practice.

The second stage was to check the behaviour of the error as a function of step–size and order to see if the predicted patterns are obtained. An example of one of these tests is presented here. The network used is shown in Fig. 3.1, with a sinewave excitation of 500Hz and zero initial conditions **[2, p.142]**. The methods are applied for one time–step only so that only the contributions from the excitations are evaluated and thus any possible roundoff errors from the EST matrix are excluded. Methods of order 1 through 5 were used for step–sizes of 0.5mS, 0.25mS and 0.125mS. The results of this experiment are given in Table 3.1.

From this table it is evident that the error decreases with decreasing step–size. To verify the order of the approximations, the ratio of successive errors are compared to the predicted ratio. Calculating the average for the two steps, we get for successive orders; 4.47 (4), 8.05 (8), 15.6 (16), 34.8 (32) and 66.4 (64). The numbers shown in brackets are the theoretical results. The agreement is very good and is typical of the results obtained for other networks. It is also evident

from Table 3.1 that the error decreases with increasing order, however the ratio of decreases between successive orders do not conform to a regular pattern as in the case of decreasing step–size.

The final test was to test the method for numerical stability. The stability of the Laplace transform inversion formula (3.31) is proved in [2, pp.305–306] for N= M–1 and N= M–2, therefore the SILT method for the homogeneous case is also stable. Ideally one would like to be able to rigourously prove that the SILT method is stable for the non–homogeneous case. This however is difficult and is left as an open problem. Therefore the stability had to be verified experimentally. The procedure to do this was as follows. For each network tested the step response was calculated. From this response the largest time–constant of the network can be estimated. The networks were then re–analysed using step–sizes larger than the estimated maximum time–constants. Although, as expected, accuracy deteriorated for these large step–sizes, stable responses were obtained in all cases. On the basis of these tests it is therefore conjectured that the SILT method is A–stable.



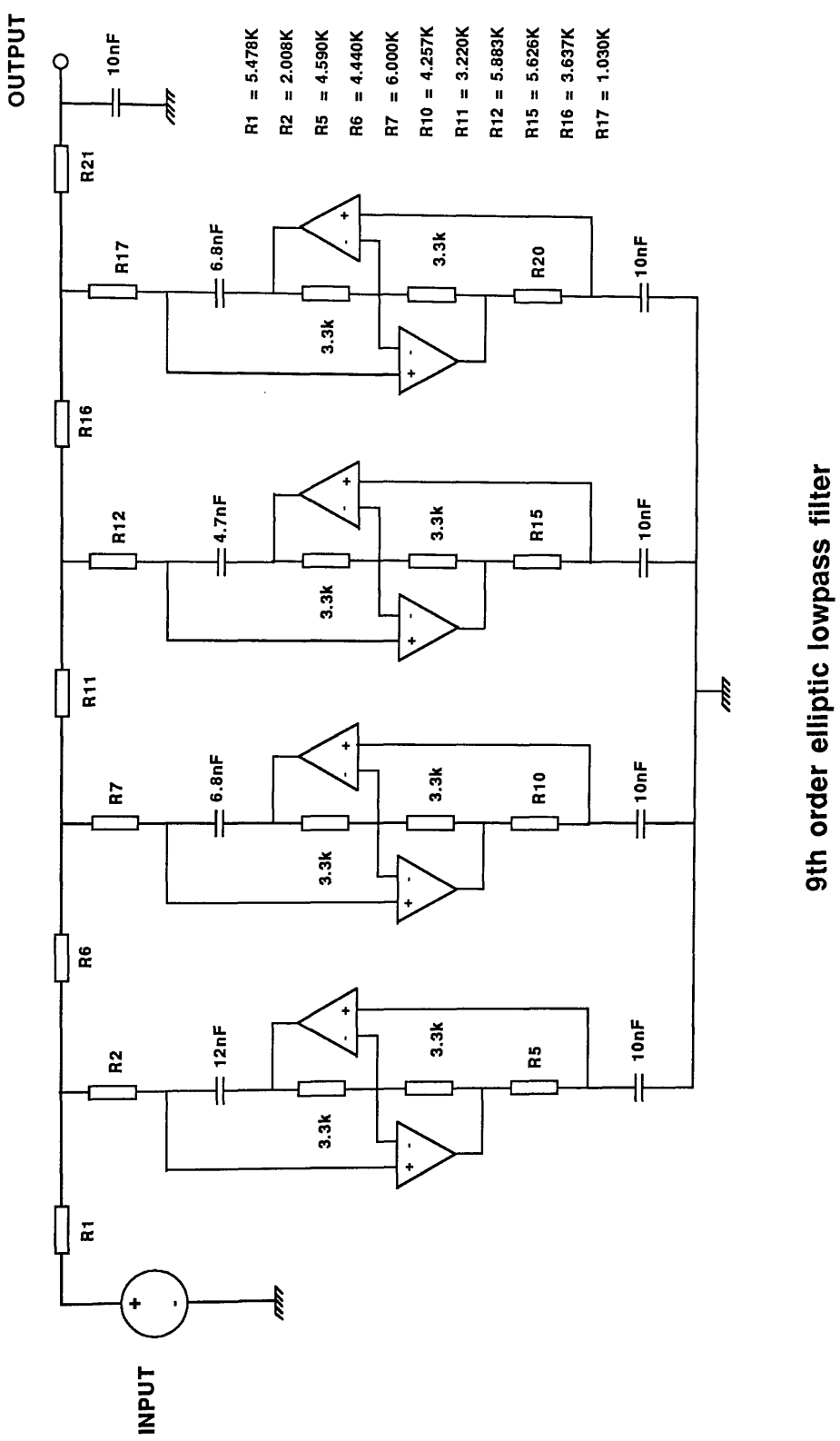


Figure 3.1 Network used for examples

ORDER	STEP–SIZE		
	5×10^{-4}	2.5×10^{-4}	1.25×10^{-4}
1	2.094×10^{-1}	3.873×10^{-2}	1.095×10^{-2}
2	9.732×10^{-3}	1.159×10^{-3}	1.505×10^{-4}
3	7.281×10^{-4}	5.454×10^{-5}	3.112×10^{-6}
4	3.753×10^{-5}	1.293×10^{-6}	3.194×10^{-8}
5	6.071×10^{-6}	4.676×10^{-8}	1.574×10^{-8}

Table 3.1 Error after one step for different orders and step–sizes

3.6.2) COMPARISON WITH OTHER METHODS

A major difficulty in comparing a number of methods is that generally methods are presented together with only one or two examples and computer programs implementing the methods are not available. Of the methods discussed in the introduction, the only program avaliable for comparison is the trapezoidal method implemented in SPICE2. The general comments made about the methods in the introduction still hold, but cannot be experimentally compared with the SILT method.

The example used to compare SPICE2 and the SILT method is shown in Fig. 3.1. The response of this network to a sinewave of 1kHz was evaluated from 0 to 2mS. This response is shown in Fig. 3.2. The SILT1 method has the same order as SPICE2 and therefore these two methods were compared for step-sizes of 5×10^{-5}, 5×10^{-6} and 5×10^{-7}. The error of the two methods was evaluated at 40 points over the 2mS interval and are plotted in Figs. 3.3, 3.4 and 3.5. As can be seen from these results, the SILT1 method has a consistently smaller error and the error varies far more smoothly than is the case for SPICE2. The SILT method of orders 2 to 5 were then applied with a step-size of 5×10^{-5}. The error of these methods are shown in Fig. 3.5. The rapid reduction in error for increasing order is clearly shown in this graph. Even more impressive is that the SILT4 method has a smaller error than the SPICE2 method with a step-size 1000 times smaller. Therefore one expects the SILT methods to be vastly more efficient than SPICE2. This is in fact the case as the run-times in Table 3.2 clearly show.

To be fair, SPICE2 does solve a set of linear equations at each grid-point, which could be avoided if the fact that a fixed step-size was being used was taken into account. This however would only decrease the run-times by some factor (about 5 in this case), it would not solve the problem of requiring step-size 1000 times smaller than the SILT4 method for similar accuracy. The SILT method is therefore seen to be a very efficient method, orders of magnitude more efficient than SPICE2.

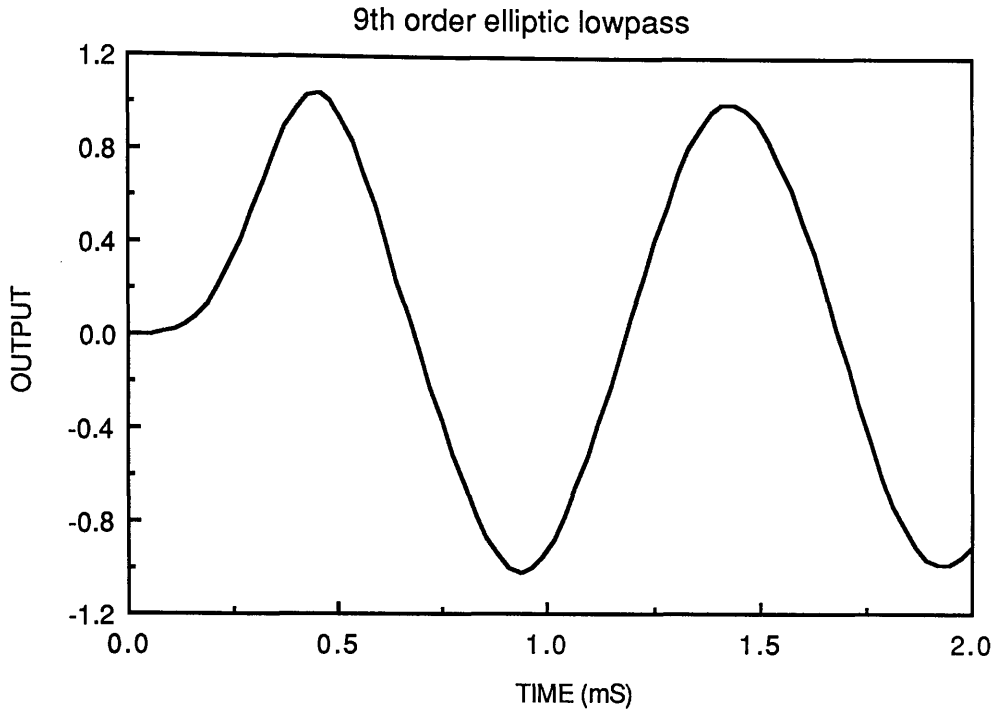


Figure 3.2 Response of network used for examples

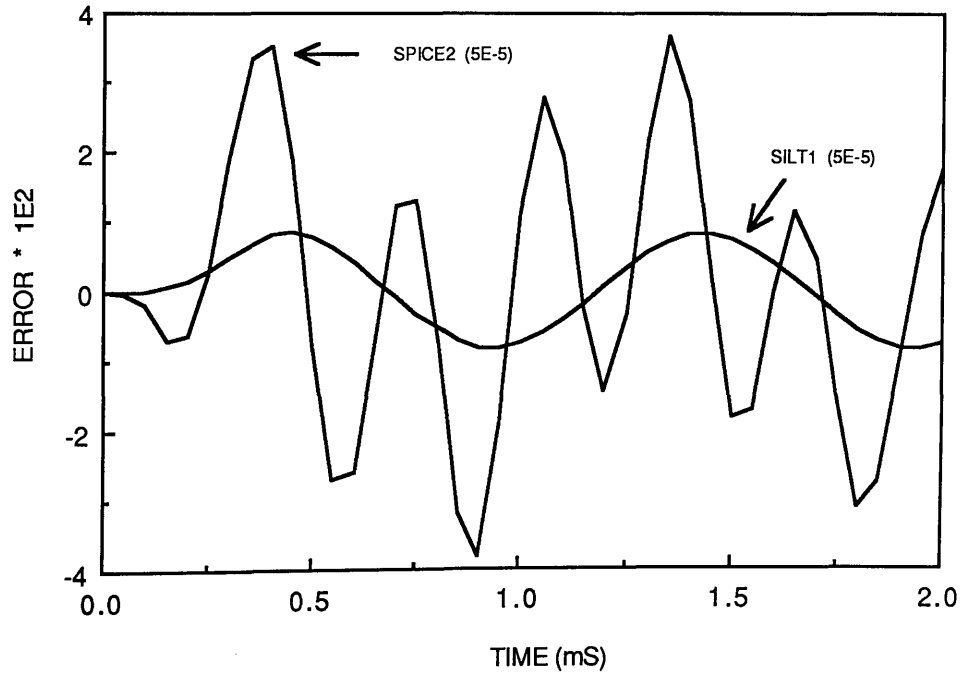


Figure 3.3 Comparison of SPICE2 and SILT1

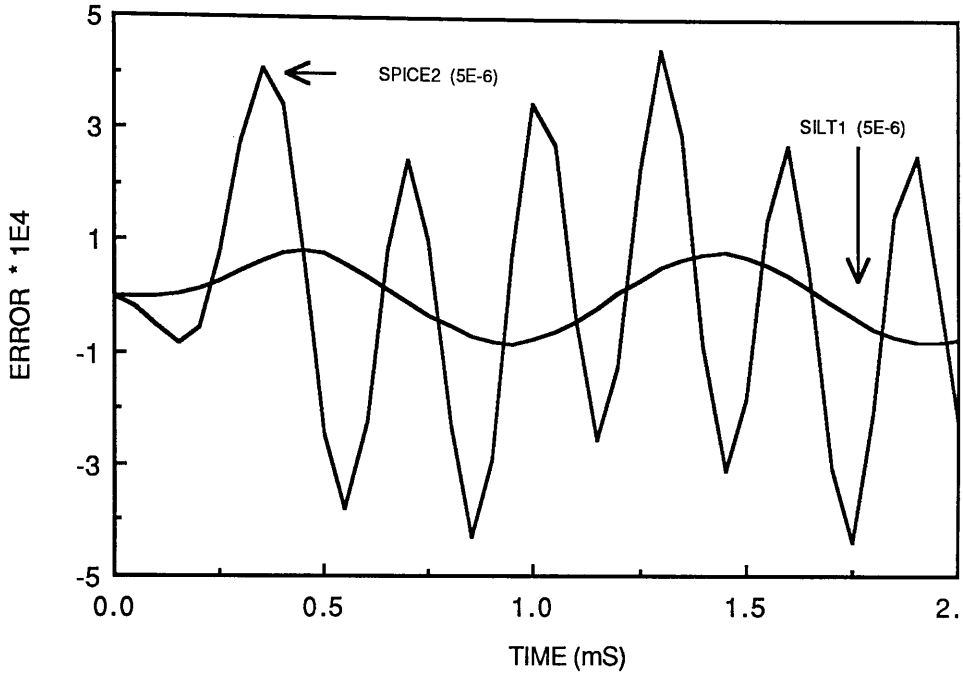


Figure 3.4 Comparison of SPICE2 and SILT1

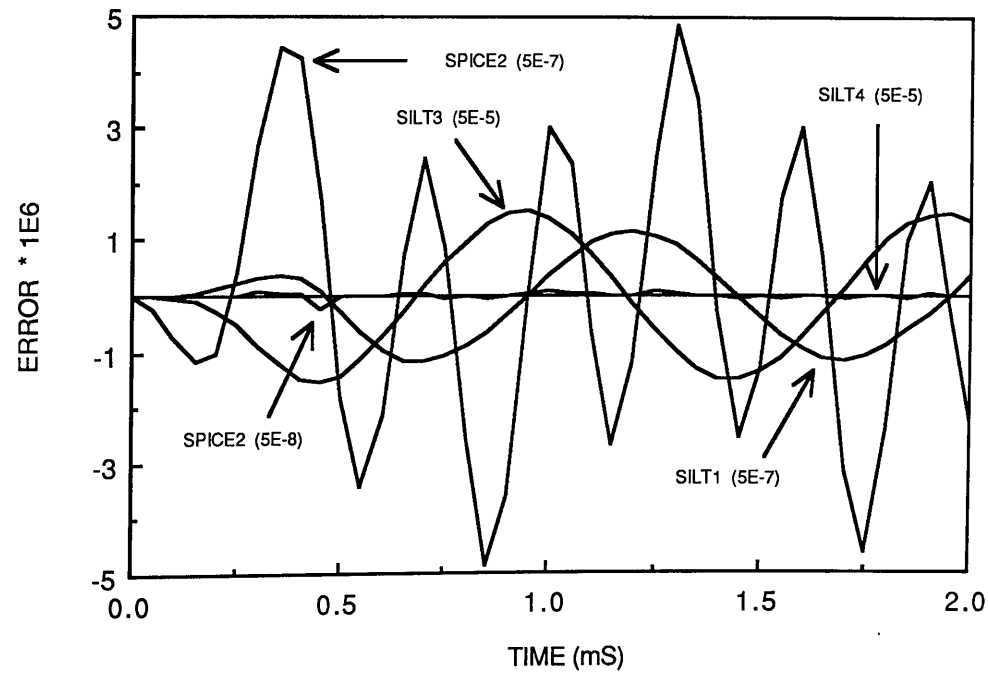


Figure 3.5 Comparison of SPICE2 and SILT method

METHOD	STEP-SIZE	MAX. ERROR	CPU TIME
SPICE2	5×10^{-5}	3.81×10^{-2}	12.13s
SPICE2	5×10^{-6}	4.44×10^{-4}	1:03.99s
SPICE2	5×10^{-7}	4.86×10^{-6}	6:52.62s
SPICE2	5×10^{-8}	2.44×10^{-7}	1:04:49.73s
SILT 1	5×10^{-5}	8.51×10^{-3}	2.51s
SILT 1	5×10^{-6}	8.55×10^{-5}	6.61s
SILT 1	5×10^{-7}	1.17×10^{-6}	47.79s
SILT 2	5×10^{-5}	1.61×10^{-5}	2.55s
SILT 3	5×10^{-5}	1.55×10^{-6}	2.57s
SILT 4	5×10^{-5}	4.00×10^{-9}	2.80s
SILT 5	5×10^{-5}	7.24×10^{-10}	2.82s

Table 3.2 Comparison of SPICE2 and SILT methods

3.7) APPLICATION TO PERIODICALLY SWITCHED NETWORKS

The SILT method developed can be applied to periodically switched linear networks by treating the network as a finite periodic sequence of networks, each unique switched state being treated as a different network. The final conditions of each network then determine the initial conditions for the next network in the sequence. The derivation of consistent initial conditions are discussed in detail in [20]. The generalisation of the SILT method to periodically switched networks is developed in Chapter Four, which then lays the basis for frequency domain analysis of these networks.

The SILT method for periodically switched linear networks was implemented in the computer program FOOLSCAP. To confirm the theory and verify that it had been implemented correctly in FOOLSCAP, comparisons were made with other computer programs designed to analyse switched–capacitor networks.

The first two programs, SCNAPIT [21], [22], [23] and SWITCAP [24], assume that the network has ideal switches, infinite bandwidth amplifiers and no resistors. Therefore the correct simulation of transient effects cannot be tested with these programs. Nonetheless they do provide a very useful test of the correct functioning of the switching mechanism and consistent formulation of initial conditions at each switching instant. Extensive comparisons with these two programs showed that the theory as implemented in FOOLSCAP provides a very efficient and accurate method for the computation of the time–domain response of periodically switched linear networks.

To test the correct simulation of transient effects in these networks, the computer program SCNAPNIT [25], [26] was used. This program can deal with finite bandwidth amplifiers, resistors in the network and nonideal switches. The switches, however, are modelled as nonlinear devices and therefore nonlinear switching effects are taken into account. Nevertheless by selecting parameters for the switches such that their behaviour is near ideal, comparisons could be made for the other transient effects due to nonideal amplifiers. These comparisons verified that FOOLSCAP correctly simulates these transient effects.

REFERENCES FOR CHAPTER THREE

[1] F.H. Branin, "Computer methods of network analysis", Proc. IEEE, Vol. 55, No. 11, pp.1787–1801, Nov. 1967.

[2] J. Vlach and K. Singhal, "Computer methods for circuit analysis and design", Van Nostrand Reinhold, 1983.

[3] G. Dahlquist, "A special stability problem for linear multistep methods", B.I.T., Vol. 3, No. 1, pp.27–43, 1963.

[4] L.W. Nagel, "SPICE2 : A computer program to simulate semiconductor circuits", ERL Memo ERL–M520, University of California, Berkeley, 1975.

[5] F. Maloberti, "An efficient method for the numerical analysis of transients in linear dynamic circuits", IEEE Trans. CAS, Vol. CAS–32, No. 8, pp.848–851, Aug. 1985.

[6] C.W. Gear, "Simultaneous numerical solution of differential–algebraic equations", IEEE Trans. CT, Vol. CT–18, No. 1, pp.89–95, Jan. 1971.

[7] W. Enright, "On the efficient and reliable numerical solution of large linear systems of ODE's", IEEE Trans. Automatic Control, Vol. AC–24, No. 6, pp.905–908, Dec. 1979.

[8] R.K. Brayton, F.G. Gustavson and G.D. Hachtel, "A new efficient algorithm for solving differential–algebraic systems using implicit backward differentiation formulas", Proc. IEEE, Vol. 60, No. 1, pp.98–108, Jan. 1972.

[9] K. Singhal and J. Vlach, "Computation of the time domain response by numerical inversion of the Laplace transform", Journal of the Franklin Institute, Vol. 299, No. 2, pp.109–126, Feb. 1975.

[10] V. Zakian, "Solution of homogeneous ordinary linear

differential systems by numerical inversion of Laplace transforms", Electronic Letters, Vol. 7, No. 18, pp.546–548, Sep. 1971.

[11] C.B. Moler and C. Van Loan, "Nineteen dubious ways to compute the exponential of a matrix", SIAM Review, Vol. 20, No. 4, pp.801–836, Oct. 1978.

[12] R.C. Ward, "Numerical computation of the matrix exponential with accuracy estimate", SIAM Journal of Numerical Analysis, Vol. 14, No. 4, pp.600–610, Sep. 1977.

[13] V. Zakian, "Properties of the I_{MN} and J_{MN} approximants and applications to numerical inversion of Laplace transforms and initial value problems", Journal Maths. and Appl., Vol. 50, 1975, pp.191–222.

[14] V. Zakian, "Rational approximants to the matrix exponential", Electronic Letters, Vol. 6, No. 25, pp.814–815, Dec. 1970.

[15] C.K. Pun and J.I. Sewell, "Symbolic analysis of ideal and nonideal switched capacitor networks", Proc. IEEE Int. Symp. on Circuits and Systems, pp.1165–1172, 1985.

[16] V. Zakian and M.J. Edwards, "Tabulation of constants for full grade I_{MN} approximants", Mathematics of Computation, Vol. 32, No. 142, pp.519–531, Apr. 1978.

[17] K. Singhal and J. Vlach, "Program for numerical inversion of Laplace transforms", Electronic Letters, Vol.7, No.14, pp.413–415, July 1971.

[18] K. Takahashi, J. Fagan and C. Mo–Shing, "Formation of a sparse bus impedance matrix and its application to short circuit study", Proc. 8th PICA Conf., June 1973, Minneapolis, Minn.

[19] A.M. Erisman and W.F. Tinney, "On computing certain elements of the inverse of a sparse matrix", Comm. ACM,

Vol. 18, No. 3, pp.177–179, Mar. 1975.

[20] A. Opal, J. Vlach and K. Singhal, "Time–domain analysis of switched networks", Proc. IEEE Int. Symp. on Circuits and Systems, May 1987, pp.60–63.

[21] L.B. Wolovitz, "Improved techniques for time–domain analysis of switched capacitor networks", M.Sc. Thesis, University of Hull, 1986.

[22] A.D. Meakin, J.I. Sewell and L.B. Wolovitz, "Techniques for improving the efficiency of analysis software for large switched–capacitor networks", Proc. 28th Midwest Symposium on Circuits and Systems, pp.390–393, Aug. 1985.

[23] L.B. Wolovitz and J.I. Sewell, "Advanced switched–capacitor analysis software", Annual Report 1, Department Electronic Engineering, University of Hull, 1984.

[24] S.C. Fang, "Switcap users guide", Dept. Electrical Engineering, Columbia University, 1982.

[25] L.B. Wolovitz and J.I. Sewell, "Efficient computer techniques for the exact analysis of all nonideal effects of switched–capacitor networks in the time–domain", Proc. IEEE Int. Symp. on Circuits and Systems, pp.373–376, May 1986.

[26] L.B. Wolovitz and J.I. Sewell, "Advanced switched–capacitor analysis software", Final Report, Dept. Electronic Engineering, University of Hull, 1985.

CHAPTER FOUR

FREQUENCY DOMAIN ANALYSIS OF PERIODICALLY SWITCHED LINEAR NETWORKS.

4.1) INTRODUCTION

This chapter considers the problem of calculating the frequency response of periodically switched linear networks. The methods developed are applicable to general switched networks, but the methods were developed specifically for switched capacitor networks. The derivations are perfectly general as no assumptions are made about the type or characteristics of the networks.

Many different methods for calculating the frequency response have been proposed and a number actually implemented in computer programs [1]. However because of the difficulty of the problem, many have limitations on applicability or other considerable drawbacks. Consequently no method or program developed to date has received widespread acceptance, unlike the situation for ideal switched capacitor analysis programs and techniques.

The main linear imperfections of switched capacitor networks are the parasitic capacitances, finite amplifier gain, switch resistances and finite amplifier gain bandwidth products [1]. The first two imperfections can be modelled by ideal analysis programs, but different techniques are required to take into account the transient effects caused by the latter two imperfections.

A number of techniques have been developed to model the effects of amplifier finite gain bandwidth product. One approach is to use the equivalent circuit approach and then use continuous time frequency analysis (AC) programs [2]. However this approach suffers from the general limitations of the equivalent circuit method [1], the main limitation being that the method is only applicable to 2 phase 50% duty cycle networks. The second approach is to include the analytic solution of the time domain response of single pole amplifiers in the analytic solution of the ideal SC network in which the amplifier is

embedded. Because this analysis is done by hand, it is extremely difficult to extend the method beyond 2nd order networks. Indeed all the methods presented to date are limited to 2 phase 2nd order filters. The results obtained using this approach are quite good [3], [4] and are particularly useful for obtaining insight into the behaviour of these networks. The third approach, which allows computer implementation, was to derive an admittance matrix in the Z− domain for the amplifier poles [5], [6]. This admittance matrix is then imbedded in a definite nodal admittance matrix formulation for ideal SC networks. However this approach models the imperfections due to the amplifier approximately and is limited to 2 phase networks.

To include the effects of finite switch resistances, as well as nonideal amplifiers, the behaviour of the networks has to be modelled in general by differential difference equations. The exact time domain solution of these type of systems was first obtained in [7]. Closed form frequency domain solution for cisoidal input were also derived. The method is based on the state space formulation which makes general computer implementation difficult and inefficient. The method is also limited to 2 phase networks. This approach was generalised for multi− phase networks and arbitrary deterministic inputs [8]. However this algorithm is also based on the state space approach and therefore not compatible with traditional CAD tools. A computer algorithm was presented in [9] which is equivalent to this approach and allows a simpler implementation. However this method is still very complicated and does not appear to be very efficient. Another state space based method is presented in [10]. This method uses intricate manipulations to derive the state space forms from nodal equations and is limited to 2 phase networks. The method also assumes that the input signal is sampled and held in each phase and therefore does not consider the effects of continuous input− output coupling. However this method is of interest as it uses a similarity transformation to reduce the computation at each frequency point to $O(n^2)$ operations. A similar approach is developed in Chapter Five and is applied to the frequency analysis method developed in this chapter.

A number of MNA based methods have been developed which overcome the problems of equation formulation encountered with the

state space approaches. The first method to use the MNA formulation [11], is based on the generalisation of the method presented in [8]. This approach is orientated to computer implementation and uses many of the efficient methods developed for ideal SC analysis. The major drawback of this approach is the poor method used to evaluate the extended state transition matrix. Two methods were proposed. The first assumes that all transients in the network have died out at the end of each time–slot, which is invalid for many networks, especially high frequency networks. The second method uses a very dubious and low accuracy technique which cannot be considered to give reliable results. A method based on the above MNA formulation, but overcoming the problem of accurately determining the extended state transition matrix, was presented in [12]. This approach derives the transfer function of the network in the S and Z domains in a symbolic form. These symbolic polynomials may then be used to evaluate the frequency response and have been used for evaluating the noise response [13]. The main drawback of this approach is that it is very slow, mainly due to the method of deriving the symbolic polynomials. It also suffers from inaccuracy for large networks. A method using the same extended state transition matrix evaluation method and using the theory developed in [11] was presented in [14]. This method suffers the same drawback as the method in [11], namely that AC analyses are required in each clock phase at each frequency point, which makes the method very slow. To try and overcome this problem, a novel method was presented in [15], which also uses the MNA formulation. This approach, instead of tackling the complete spectral analysis, restricts the method to solving the discrete system associated with a nonideal SC network. However an inefficient and inaccurate method is used for calculating the transition matrices. The advantage of this approach is that it allows a very efficient technique developed for ideal SC analysis [16] to be used. This technique is also used to great effect in the method developed in this chapter.

The method developed in this chapter is an attempt to overcome the problems of the previous methods. This approach uses the MNA formulation and the accurate and reliable method presented in Chapter Three for calculating the extended state transition matrices. To avoid the computation of AC responses at each frequency point, the time

domain method developed in Chapter Three is used as the basis for the frequency domain method. This approach leads to a discrete system similar to that derived in [15] which can be very efficiently solved, except that the restriction of sampled and held inputs is not required and therefore full spectral analysis is possible.

Finally the results of this method are compared with other techniques to verify the theory and implementation. The performance of the algorithm is also compared with these other techniques to evaluate whether the efficiency of the method meets the requirements for use in a productive CAD environment.

4.2) DEFINITIONS

Consider a periodically switched linear network controlled by clock signals $\varphi_i(t)$ with a common switching period T, i.e.

$$\varphi_i(t+T) = \varphi_i(t) \qquad \forall t, i \tag{4.1}$$

where φ_i is the state of clock i, either on or off.
Each period T is partitioned into N time–slots,

$$I_{n,k} = (nT+\sigma_{k-1},\ nT+\sigma_k] \qquad k = 1,..,N \tag{4.2}$$

such that the clock signals (and therefore the network) does not vary in $I_{n,k}$. Here k denotes the kth time–slot. As seen from Fig. 4.1 these time–slots are not necessarily of equal duration. The switching frequency is defined as,

$$\omega_s = \frac{2\pi}{T} \tag{4.3}$$

Other important definitions from the figure are

$$\begin{aligned} \sigma_0 &= 0 \\ \sigma_N &= T \\ \tau_k &= \sigma_k - \sigma_{k-1} \end{aligned} \tag{4.4}$$

Define the signals

$$\begin{aligned} v_k(t) &= v(t) && t \in I_{n,k} \\ v_{n,k}(t) &= v_k(nT+\sigma_{k-1}+t) && 0 < t \leq \tau_k \end{aligned}$$

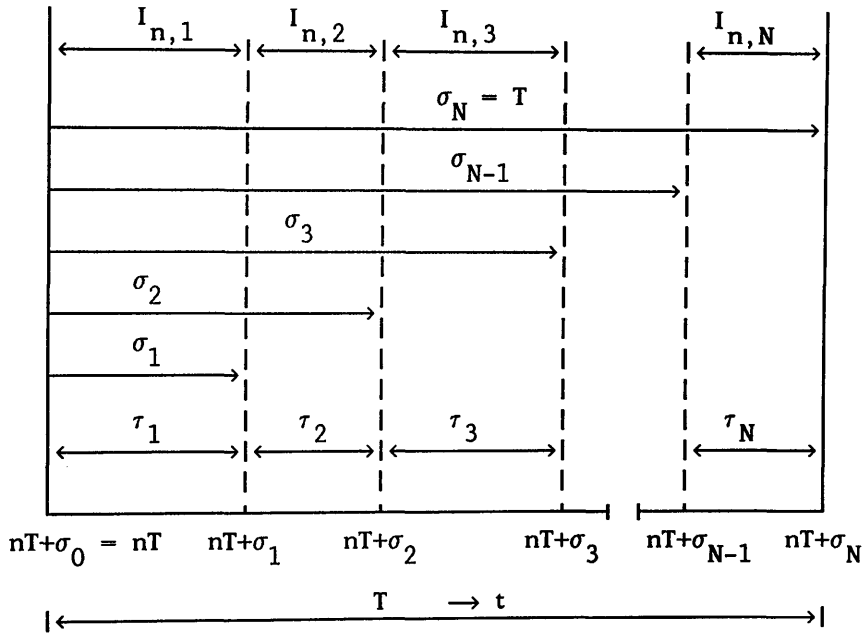


Figure 4.1 Definitions for a N− slot switched linear network

$$= v\,(nT+\sigma_k-\tau_k+t) \qquad (4.5)$$

$$v_k(nT+\sigma_k) = v(nT+\sigma_k)^- \qquad (4.6)$$

That is $v_{n,k}(t)$ are functions of time, whilst $v_k(nT+\sigma_k)$ are sequences of values at the instants just prior to switching.

Define the Z– transform

$$\sum_{n=0}^{\infty} v_k(nT+\sigma_k)z^{-n} = V_k(z) \quad k = 1,..,N-1$$

$$\sum_{n=0}^{\infty} v_N(nT)z^{-n} = V_N(z) \qquad (4.7)$$

and therefore

$$\sum_{n=0}^{\infty} v_N(nT+\sigma_N)z^{-n} = \sum_{n=0}^{\infty} v_N(nT+T)z^{-n} = zV_N(z) \qquad (4.8)$$

4.3) TIME DOMAIN ANALYSIS

In each time–slot $I_{n,k}$ of the nth switching period [nT,(n+1)T] a periodically switched linear network can be represented by the system differential equation

$$C_k\,\dot{v}_{n,k}(t) + G_k\,v_{n,k}(t) = w_{n,k}(t) \quad k = 1,..,N \qquad (4.9)$$

where $w_{n,k}(t)$ is the vector of excitations
$v_{n,k}(t)$ is the vector of unknown system variables
C_k and G_k are constant matrices

In the following, the range of subscripts for k (k = 1,..,N) is dropped for brevity but is assumed throughout unless otherwise stated.

Using the approach developed in Chapter Three we approximate the excitations by *m*th order polynomials

$$w_{n,k}(t) \approx \sum_{i=0}^{m} \alpha_{i,k}^{n}\, t^i \qquad (4.10)$$

where $\alpha_{i,k}^{n}$ are the coefficients of the polynomial approximations of

$w_k(t)$ in the interval $I_{n,k}$. Substituting the approximation (4.10) in the system (4.9) gives the new system differential equation

$$C_k \dot{v}_{n,k}(t) + G_k v_{n,k}(t) = \sum_{i=0}^{m} \alpha_{i,k}^{n} t^i \qquad (4.11)$$

Following the steps in Chapter Three, the solution of equation (4.11) is,

$$v_{n,k}(t) = P_k(t) C_k v_{k-1}(nT+\sigma_{k-1}) + \sum_{i=0}^{m} \alpha_{i,k}^{n} B_{i,k}(t) \qquad (4.12)$$

where $$P_k(t) = L^{-1}\{ [sC_k + G_k]^{-1}\} \qquad (4.13)$$

$$B_{i,k}(t) = L^{-1}\{ [sC_k + G_k]^{-1} \frac{i!}{s^{i+1}}\} \qquad (4.14)$$

$v_{k-1}(nT+\sigma_{k-1})$ are the initial conditions

The sequence $v_{k-1}(nT+\sigma_{k-1})$ which are the initial conditions of the system for interval $I_{n,k}$ are by definition the final states of the system for interval $I_{n-1,k}$. Therefore a recurrence relation giving the sequence of final states of the system within each time-slot is obtained by substituting $t = \tau_k$ in equation (4.12),

$$v_k(nT+\sigma_k) = P_k v_{k-1}(nT+\sigma_{k-1}) + \sum_{i=0}^{m} \alpha_{i,k}^{n} B_{i,k} \qquad (4.15)$$

where $$P_k = P_k(\tau_k) C \qquad (4.16)$$

$$B_{i,k} = B_{i,k}(\tau_k) \qquad (4.17)$$

This recurrence relationship is similar to equation (3.13) derived for nonswitched linear systems in Chapter Three, except that in this case the time-steps τ_k are not necessarily equal. Thus the same efficient computational techniques may be used to evaluate (4.15). This formula only provides solutions of the system (4.11) at the switching instants, which for many applications gives sufficient information. However if intermediate results are required, for example to observe transients within each time-slot, then the time-slots may be subdivided into equal subintervals and (4.15) modified to take these subintervals into account. This approach provides a technique for efficiently evaluating the time domain response of periodically switched linear systems described in Chapter Three.

As in Chapter Three a procedure is required to determine the coefficients $\alpha^n_{i,k}$. Defining

$$h_k = \frac{\tau_k}{m} \tag{4.18}$$

$$f^n_{\ell,k} = w_{n,k}(\ell h_k) \tag{4.19}$$

and applying equation (3.20), gives

$$\alpha^n_{i,k} = \sum_{\ell=0}^{m} \gamma_{\ell i} \, f^n_{\ell,k} \tag{4.20}$$

4.4) COMPUTING THE INVERSE LAPLACE TRANSFORM

The formula (4.15) requires the matrices P_k, given by equation (4.16), and the vectors $B_{i,k}$, given by the equation (4.17). Using the definition of the extended state transition matrix (4.13) and applying the approximation (3.33) gives,

$$P_k(\tau_k) \approx \sum_{i=1}^{M/2} \mathrm{Re}\left[2K_i/\tau_k \left[z_i/\tau_k C_k + G_k \right]^{-1} \right] \tag{4.21}$$

Similarly applying the excitation response approximation (3.34) to the definition (4.14) gives,

$$B_{i,k}(\tau_k) \approx \sum_{j=1}^{M/2} \mathrm{Re}\left[\frac{2K_j/\tau_k \; i!}{(z_j/\tau_k)^{i+1}} \left[z_j/\tau_k C_k + G_k \right]^{-1} \right] \tag{4.22}$$

The efficient implementation strategies discussed in section 3.5.1 are equally applicable to these approximations. In this case because the calculations are repeated for each slot, the benefits of the sparse matrix methods of Chapter Two are even greater.

4.5) Z– DOMAIN ANALYSIS

To obtain the Z– domain series of final states $V_k(z)$ for k=1,..,N, we need to take the Z– transform of equation (4.15) and solve for $V_k(z)$.

The Z– transform of (4.15) is,

$$V_k(z) = P_k \; V_{k-1}(z) + \Sigma W_k(z) \tag{4.23}$$

where

$$\Sigma W_k(z) = \sum_{n=0}^{\infty} \left[\sum_{i=0}^{m} \alpha_{i,k}^{n} B_{i,k} \right] z^{-n} \qquad (4.24)$$

To evaluate $\Sigma W_k(z)$, substitute equations (4.19) and (4.20), which gives,

$$\Sigma W_k(z) = \sum_{n=0}^{\infty} \left[\sum_{i=0}^{m} \left(\sum_{\ell=0}^{m} \gamma_{\ell i} W_{n,k}(\ell h_k) \right) B_{i,k} \right] z^{-n} \qquad (4.25)$$

From definition (4.15) this becomes,

$$\Sigma W_k(z) = \sum_{n=0}^{\infty} \left[\sum_{i=0}^{m} \left(\sum_{\ell=0}^{m} \gamma_{\ell i} W_k(nT+\sigma_k+\ell h_k-\tau_k) \right) B_{i,k} \right] z^{-n} \qquad (4.26)$$

And finally applying definition (4.6) and using the shifting theorem,

$$\Sigma W_k(z) = \sum_{i=0}^{m} B_{i,k} \left(\sum_{\ell=0}^{m} \gamma_{\ell i} W_k(z)\, z^{(\ell h_k-\tau_k)/T} \right) \qquad (4.27)$$

The system (4.23) can be represented in matrix form as,

$$\begin{bmatrix} I & & & -z^{-1}P_1 \\ -P_2 & I & & \\ & \ddots & \ddots & \\ & & \ddots & \\ & & -P_N & I \end{bmatrix} \begin{bmatrix} V_1(z) \\ V_2(z) \\ \vdots \\ \vdots \\ zV_N(z) \end{bmatrix} = \begin{bmatrix} \Sigma W_1(z) \\ \Sigma W_2(z) \\ \vdots \\ \vdots \\ z\Sigma W_N(z) \end{bmatrix} \qquad (4.28)$$

Equation (4.28) can be rewritten to remove the factor z^{-1} from $-z^{-1}P_1$ as,

$$\begin{bmatrix} I & & & -P_1 \\ -P_2 & I & & \\ & \ddots & \ddots & \\ & & \ddots & \\ & & -P_N & zI \end{bmatrix} \begin{bmatrix} V_1(z) \\ V_2(z) \\ \vdots \\ \vdots \\ V_N(z) \end{bmatrix} = \begin{bmatrix} \Sigma W_1(z) \\ \Sigma W_2(z) \\ \vdots \\ \vdots \\ z\Sigma W_N(z) \end{bmatrix} \qquad (4.29)$$

This modification which puts the variable z into the bottom right submatrix of the system matrix is necessary for the efficient solution method developed in the next section.

4.6) SOLVING THE DISCRETE SYSTEM

A substantial amount of frequency independent preprocessing can be performed in solving equation (4.29). The approach used is based on the approach developed in [16] for ideal SC networks. Firstly all the P_k are frequency independant and therefore are pre-computed using equation (4.21) and stored. Similarly the $B_{i,k}$ in equation (4.14) are independent of frequency and are pre-computed using equation (4.22) and stored.

Performing a block Gaussian elimination on equation (4.29),

$$(zI - E)\, V_N(z) = \sum_{k=1}^{N} E_k\, \Sigma W_k(z) \tag{4.30}$$

where $E = P_N P_{N-1} \cdots P_2 P_1$ (4.31)

$$E_k = \begin{cases} P_N P_{N-1} \cdots P_{k+1} & k = 1,..,N-1 \\ I & k = N \end{cases} \tag{4.32}$$

Matrices E and E_k are frequency independ nt and are only computed once prior to frequency analysis. The multiplication by E_k can be distributed over the summation in the excitation $\Sigma W_k(z)$ as a preprocessing step, giving,

$$\Sigma W_k^P(z) = \sum_{i=0}^{m} F_{i,k} \sum_{\ell=0}^{m} \gamma_{\ell i} W_k(z)\, z^{(\ell h_k - \tau_k)/T} \tag{4.33}$$

where $F_{i,k} = E_k\, B_{i,k}$ (4.34)

Equation (4.30) then reduces to

$$(zI - E)\, V_N(z) = \sum_{k=1}^{N} \Sigma W_k^P(z) \tag{4.35}$$

which can be very efficiently solved for $V_N(z)$ using methods presented in Chapter Five. The solutions for $V_k(z)$ for k=1,..,N-1 are then obtained by block backsubstitution,

$$\begin{aligned} V_1(z) &= P_1\, V_N(z) + \Sigma W_1(z) \\ V_k(z) &= P_k\, V_{k-1}(z) + \Sigma W_k(z) \qquad k = 2,..,N-1 \end{aligned} \tag{4.36}$$

4.7) FREQUENCY ANALYSIS

To solve the discrete system (4.29) for a particular frequency ω, substitute,

$$z = e^{j\omega T} \tag{4.37}$$

Closely following the development in [16], applying Poisson's formula to $\Sigma W_k(z)$† gives

$$\Sigma W_k(e^{j\omega T}) = \sum_{n=0}^{\infty} \left[\sum_{i=0}^{m} \left(\sum_{\ell=0}^{m} \gamma_{\ell i} W_k(nT+\sigma_k+\ell h_k-\tau_k) \right) B_{i,k} \right] e^{-jn\omega T}$$

$$= \frac{1}{T} \sum_{n=0}^{\infty} \left[\sum_{i=0}^{m} B_{i,k} \sum_{\ell=0}^{m} \gamma_{\ell i} W(\omega-n\omega_s)\, e^{j(\omega-n\omega_s)(\sigma_k+\ell h_k-\tau_k)} \right] \tag{4.38}$$

Consider a complex exponential input signal of frequency ω_0

$$w(t) = e^{j\omega_0 T} \tag{4.39}$$

which has a Fourier transform

$$W(\omega) = 2\pi\, \delta(\omega-\omega_0) \tag{4.40}$$

Now

$$W(\omega-n\omega_s) = 2\pi\, \delta(\omega-\omega_0-n\omega_s) \tag{4.41}$$

so $\Sigma W_k(e^{j\omega T})$ consists of an infinite number of terms occuring at frequencies

$$\omega = \omega_0 + n\omega_s \tag{4.42}$$

Now it is shown in [16] that,

$$\sum_{n=0}^{\infty} W(\omega-n\omega_s)\, e^{j(\omega-n\omega_s)\sigma_k} = 2\pi\, e^{j\omega_0\sigma_k} \tag{4.43}$$

which is independent of n. Applying this result to equation (4.38) gives,

$$\Sigma W_k(e^{j\omega_0 T}) = \frac{2\pi}{T} \sum_{i=0}^{m} B_{i,k} \sum_{\ell=0}^{m} \gamma_{\ell i} e^{j\omega_0(\sigma_k+\ell h_k-\tau_k)} \tag{4.44}$$

With input frequency ω_0 an infinite number of output frequencies

$$\omega = \omega_0 + n\omega_s \tag{4.45}$$

are defined. However as noted above the discrete system (4.29) need

† using equation (4.26)

only be solved once for ω_0 as the solution is independent of n. However switched linear networks are not discrete systems so window functions have to be used to obtain the frequency domain response of the systems [17]. Applying the window functions to the output signals and taking into account the sinx/x sampling effect and possibly unequal time–slots, the frequency response of the system is given by [16],

$$S_n = \sum_{k=1}^{N} D_{k,n} V_k(e^{j\omega_0 T}) \qquad (4.46)$$

where $$D_{k,n} = \frac{e^{-j\omega\sigma_{k-1}} - e^{-j\omega\sigma_k}}{j\omega T} \qquad k \neq N \qquad (4.47)$$

$$D_{N,n} = e^{j\omega_0 T} \frac{e^{-j\omega\sigma_{N-1}} - e^{-j\omega\sigma_N}}{j\omega T} \qquad (4.48)$$

The individual phase responses $V_k(e^{j\omega_0 T})$ are obtained for input frequency ω_0 by substituting equation (4.44) into (4.29) and using (4.35) and (4.36) to solve the system.

The overall algorithm for frequency domain analysis is then,

A *Preprocessing independent of frequency and n*

1) Formulate the matrices G_k and C_k and the vectors W_k
2) Calculate P_k matrices using equation (4.16) and (4.21)
3) Calculate $B_{i,k}$ vectors using equation (4.22) for i = 0,..,m
4) Calculate matrix E using equation (4.31)
5) Calculate $F_{i,k}$ vectors using equation (4.34) for i = 0,..,m

B *Frequency analysis independent of n*

1) Prepare matrix $(e^{j\omega_0 T}I - E)$
2) Build RHS of equation (4.35) using equation (4.33)
3) Solve equation (4.35) for $V_N(e^{j\omega_0 T})$
4) Calculate $V_k(e^{j\omega_0 T})$ using equation (4.36)

C *Spectral analysis*

1) For selected n calculate weights $D_{k,n}$ using equations (4.47) and (4.48)
2) Calculate S_n using equation (4.46)

(4.49)

4.8) RESULTS

The theory developed above was implemented in the program FOOLSCAP, using 16 digit double precision arithmetic throughout. To verify the implementation of the theory and determine the efficiency of the approach presented, the program was compared with a number of switched capacitor analysis programs. The programs which were available and which were used are the ideal SC analysis programs SWITCAP [18] and SCNAPIF [14], and the nonideal SC analysis program SCNAPNIF [14]. The ideal analysis programs have been independently verified and both agree exactly with one another. The nonideal program SCNAPNIF has been verified by comparison with the program SCNAP3 [12].

A number of papers describing various nonideal SC analysis techniques that give worked examples were also used. Many of these papers also compare their results with experimental measurements which provides another useful source for verification.

All the programs were implemented on a μVAX II under VMS and are written in FORTRAN. All the run statistics given are for these implementations.

4.8.1) VERIFICATION

The first step of the verification process was to perform a comparison with the ideal SC analysis programs. Because these programs do not take switch resistances into account, it is necessary for comparison with FOOLSCAP to set the switch resistances to values such that their effect is negligible. In general there is no unique set of on/off values which is suitable as the sensitivities of the various networks to the switch values vary from network to network. However 'on' resistances (Ron) of 1 ohm and 'off' resistances (Roff) of 10^{10} ohms proved in general to be satisfactory. A bit of experimentation with slightly perturbed values was used to quickly ascertain whether these values were suitable. If not then higher/lower values were tried until a suitable set was found.

The results from FOOLSCAP generally agreed with the other programs to between 2 and 6 decimal places. For a few examples it was found that small constant differences in the passband of the filters were obtained, with good agreement in the stopband. The cause of this could not be ascertained and therefore it is not known whether this is a real characteristic of the filters or due to inaccuracies in the program. However the nonideal analysis program SCNAPNIF showed similar behaviour, so it seems unlikely that the problem is due to inaccuracies (unless of course this program suffers from the same inaccuracy). Given the above results it was concluded that the theory as implemented is correct, at least for very small Ron and large Roff.

The second step of the verification process was to compare the results from FOOLSCAP for networks where the effects of switch Ron/Roff and finite amplifier gain–bandwidth are taken into account. The program SCNAPNIF was used for comparison. Generally the agreement between the two programs was very good, between 2 and 5 decimal places. Better agreement is unlikely to be obtained as the two programs use different methods for calculating the extended state transition matrix (besides the fact that very different approaches are used overall). Comparisons of the EST matrices showed that on average about 5 figure agreement is obtained.

The third set of tests carried out were comparisons with results presented in the literature for nonideal SC networks. In all cases the results are presented in graphical form so that exact numerical comparison is not possible. However the graphs provide a good visual check of whether the correct trends are obtained, for example peaking at passband edges or shifting of centre frequency. Two hand analytical methods were presented in [3], [4] and both compare their results with experimental measurements. The results obtained from FOOLSCAP agreed very well with these measurements. Unfortunately both papers considered only 2nd order bandpass filters, which limits the degree to which the program is tested for accuracy. The other comparisons were obtained from papers describing other analysis methods [9], [10], [11], [15] of which two also had experimental comparisons. Again, unfortunately, all but one considered 2nd order bandpass filters, the other network considered was a 6th order

bandpass filter. The results obtained from FOOLSCAP compared very well with these results.

Considering the above results it is concluded that the theory as implemented in FOOLSCAP is able to correctly analyse nonideal SC networks taking into account all linear imperfections.

4.8.2) PERFORMANCE

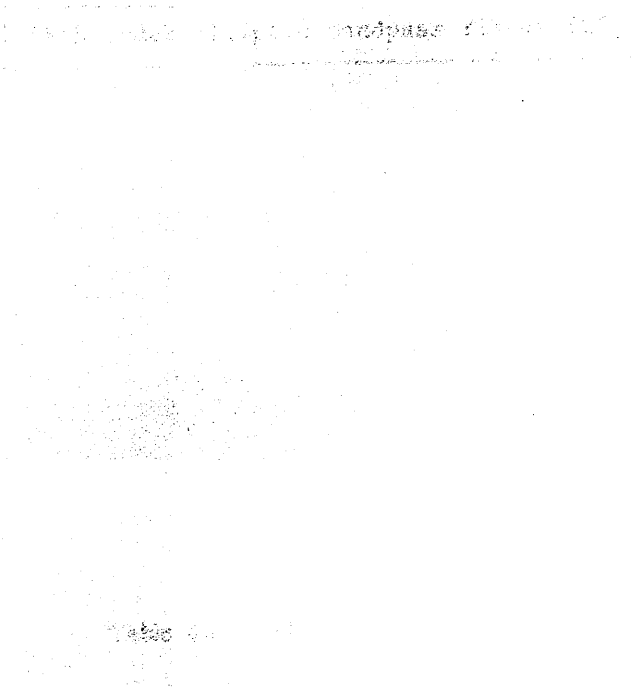
To compare the performance of FOOLSCAP with the other analysis programs, the programs were used to analyse a number of example networks. The examples used, given in Table 4.1, were selected to cover a broad range of network size, order, complexity and number of clock phases. The statistics for FOOLSCAP, SCNAPIF, SWITCAP and SCNAPNIF are presented in Tables 4.2, 4.3, 4.4 and 4.5 respectively.

From the results it is seen that in terms of storage requirements FOOLSCAP compares favourably with the other programs. For larger networks the storage requirements are fairly high but present no problem for most minicomputers. Comparing the FOOLSCAP run–times with those of SCNAPIF, it is seen that FOOLSCAP is about twice as slow as SCNAPIF, though the difference between the two reduces as the network size increases. However the time required for preprocessing for FOOLSCAP is very much greater than that for SCNAPIF. Similarly comparing the results with SWITCAP, it is seen that FOOLSCAP is approximately twice as fast as SWITCAP. This is quite remarkable considering that FOOLSCAP performs a full nonideal analysis whilst SWITCAP only performs an ideal analysis. The preprocessing times for FOOLSCAP are comparable for smaller networks, but the preprocessing times for FOOLSCAP increase more rapidly than those for SWITCAP for larger networks.

Comparing the results of FOOLSCAP with the nonideal analysis program SCNAPNIF, it is seen that FOOLSCAP is as much as 200 times faster. The preprocessing times also show a speedup of about 10. The main contributory factors to this speedup is the efficient solution method developed in Chapter Five, but the block solution of the system also contributes significantly. Another important factor is

that SCNAPNIF performs an AC analysis for each phase at each frequency point, which is not required by FOOLSCAP. The speedup in preprocessing is mostly due to the highly efficient solution method developed in Chapter Two.

To graphically illustrate the relative performances of the programs which were compared, the run–times per frequency point for 2 phase networks are given in Fig. 4.2. From this graph it is seen that the run time for SCNAPNIF increases dramatically for larger networks, whereas FOOLSCAP increases apace with the ideal analysis programs. To obtain an overall comparison of efficiency it is necessary to compare the total time required for a typical analysis run, which includes preprocessing time. Therefore the run times for 2 phase networks which include the total preprocessing time and the time required for 150 frequency points are given in Fig. 4.3. From this graph it is seen that the same observations as above apply and FOOLSCAP clearly performs very favourably compared to the ideal analysis programs and is significantly better than the nonideal analysis program.



	DESCRIPTION
1	5th order elliptic lowpass filter [19]
2	6th order Chebyschev bandpass filter [19]
3	2nd order bandpass filter, Q = 20 [20]
4	5th order elliptic lowpass filter [21]
5	11th order elliptic lowpass filter [21]
6	7th order Chebyschev lowpass filter [22]
7	3rd order elliptic lowpass filter [22]
8	15th order elliptic lowpass, leapfrog design [23]
9	15th order elliptic lowpass, LUD design [23]
10	6th order elliptic bandpass filter [24]
11	SPFT elliptic bandpass filter system [24]
12	18th order elliptic bandpass filter [25]

Table 4.1 Examples used for comparison

FOOLSCAP RUN STATISTICS

EXAMPLE	NO. NODES	NO. SLOTS	PRE-Pr. (secs)	TIME/PT (secs)	STORAGE (words)
1	22	2	4.25	0.075	12288
2	28	2	5.55	0.114	16890
3	11	2	1.14	0.034	3664
4	20	4	5.26	0.092	11032
5	41	4	22.7	0.320	39944
6	30	6	12.4	0.234	27002
7	12	12	4.82	0.099	8520
8	70	2	47.4	0.701	94564
9	69	2	47.6	0.679	95210
10	27	2	6.21	0.104	17184
11	86	36	14:52	13.44	824944
12	77	5	93.4	1.512	145854

Table 4.2 Statistics for FOOLSCAP

SCNAPIF RUN STATISTICS

EXAMPLE	NO. NODES	NO. SLOTS	PRE-Pr. (secs)	TIME/PT (secs)	STORAGE (words)
1	22	2	0.61	0.042	50217
2	28	2	0.64	0.046	51849
3	11	2	0.37	0.027	48020
4	20	4	0.80	0.063	49941
5	41	4	1.94	0.161	57162
6	30	6	1.32	0.137	53276
7	12	12	1.66	0.336	50429
8	70	2	1.93	0.169	74369
9	69	2	2.33	0.237	74415
10	27	2	0.77	0.050	51794
11	86	36	1:43	17.48	456383
12	77	5	5.88	1.156	410674

Table 4.3 Statistics for SCNAPIF

SWITCAP RUN STATISTICS

EXAMPLE	NO. NODES	NO. SLOTS	PRE-Pr. (secs)	TIME/PT (secs)	STORAGE (words)
1	22	2	5.51	0.219	11292
2	28	2	6.14	0.207	11852
3	11	2	2.53	0.108	10556
4	20	4	-	-	-
5	41	4	-	-	-
6	30	6	-	-	-
7	12	12	-	-	-
8	70	2	24.6	1.135	28092
9	69	2	25.9	1.621	34188
10	27	2	7.50	0.279	13048
11	86	36	-	-	-
12	77	5	-	-	-

Table 4.4 Statistics for SWITCAP

SCNAPNIF RUN STATISTICS

EXAMPLE	NO. NODES	NO. SLOTS	PRE-Pr. (secs)	TIME/PT (secs)	STORAGE (words)
1	22	2	31.3	3.288	27760
2	28	2	35.7	5.367	39682
3	11	2	4.82	0.549	7590
4	20	4	41.7	4.244	27317
5	41	4	4:25	28.65	100489
6	30	6	2:21	18.32	71070
7	12	12	33.5	6.281	32561
8	70	2	7:15	74.54	233532
9	69	2	6:17	74.11	230217
10	27	2	47.3	5.407	41221
11	86	36	-	-	-
12	77	5	21:48	3:36	368254

Table 4.5 Statistics for SCNAPNIF

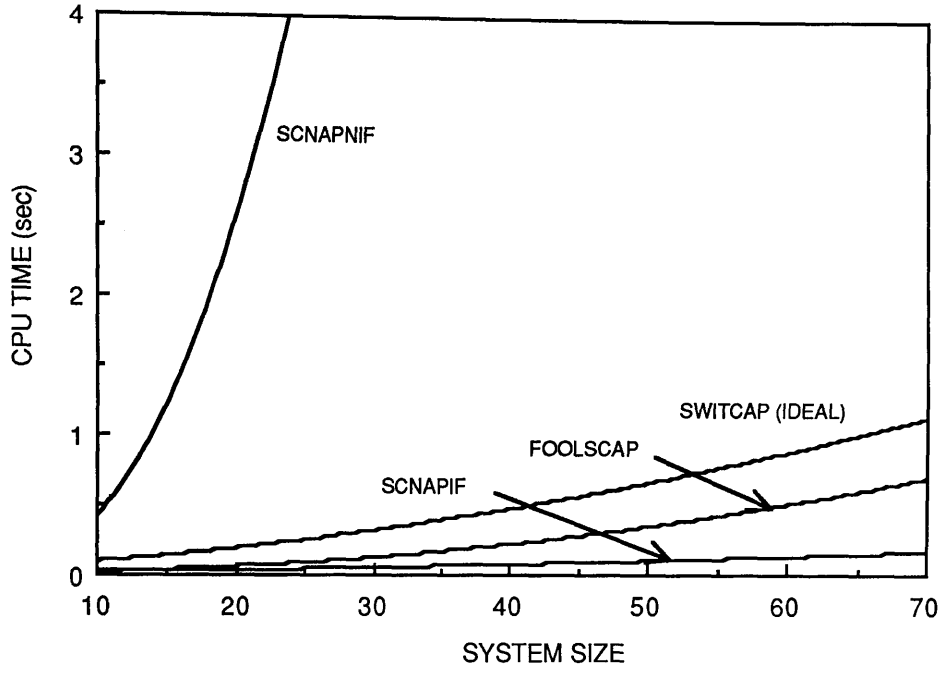


Figure 4.2 Comparison of run–times per frequency point

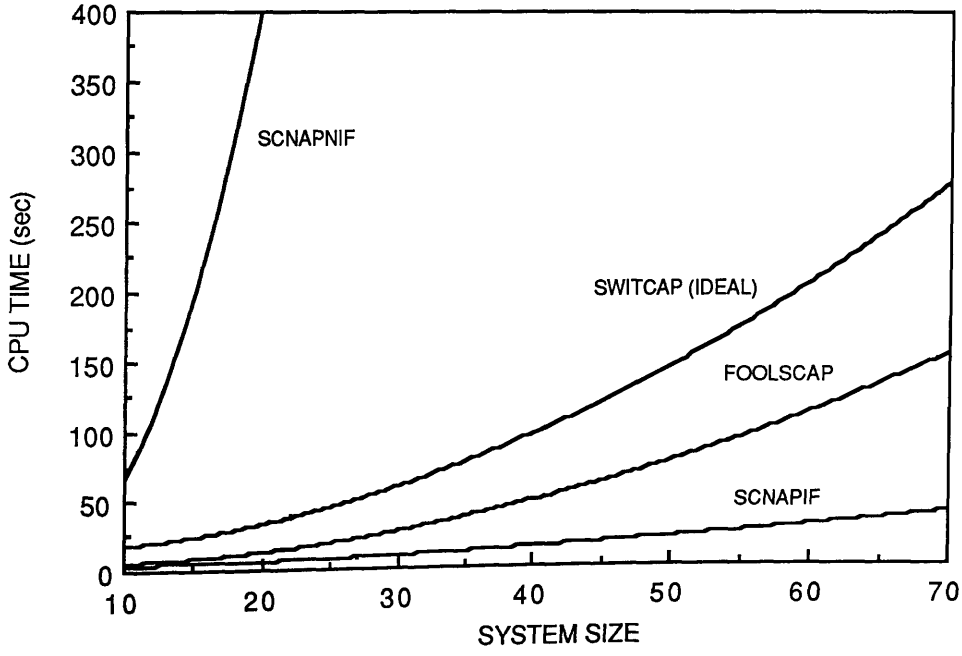


Figure 4.3 Comparison of overall run–times (150 frequency points)

REFERENCES FOR CHAPTER FOUR

[1] M.L. Liou, Y.L. Kuo and C.F. Lee, "A tutorial on computer aided analysis of switched capacitor circuits", Proc. IEEE, Vol. 71, No. 8, Aug. 1983, pp.987–1005.

[2] H. Kunieda, "Effects of finite gain–bandwidth products of op amps on switched capacitor networks – Approached via equivalent representations", Proc. IEEE Int. Symp. on Circuits and Systems, pp.464–467, May 1982.

[3] B.B. Bhattacharyya and R. Raut, "Analysis of switched capacitor networks containing operational amplifiers with finite DC gain and gain–bandwidth product values", Proc. IEE, Vol. 130, Pt. G, No. 4, pp.114–123, Aug. 1983.

[4] J.S. Martinez, E. Sanchez–Sinencio and A.S. Sedra, "Effects on the performance of a pair of SC biquads due to the op amp gain–bandwidth product", Proc. IEEE Int. Symp. on Circuits and Systems, pp.373–376, May 1982.

[5] J. Lau and J.I. Sewell, "Inclusion of amplifier finite gain and bandwidth in analysis of switched capacitor filters", Electronic Letters, Vol. 16, No. 12, pp.462–463, June 1980.

[6] N.J. Cutland, C. Lau and J.I. Sewell, "General computer analysis of switched capacitor networks including nonideal amplifier", Proc. IEEE Int. Symp. on Circuits and Systems, pp.17–20, May 1982.

[7] M.L. Liou, "Exact analysis of linear circuits containing periodically operated switches with applications", IEEE Trans. CT, Vol. CT–19, Mar. 1972, pp.146–154.

[8] T. Ström and S. Signell, "Analysis of periodically switched linear networks", IEEE Trans. CAS, Vol. CAS–24, Oct. 1977, pp.531–540.

[9] E. Wehrhahn, "Exact analysis of periodically switched linear networks", Proc. IEEE Int. Symp. on Circuits and Systems, April 1981, pp.884–887.

[10] E.P. Rudd and R. Schaumann, "A program for the analysis of high frequency behaviour of switched capacitor networks", Proc. IEEE Int. Symp. on Circuits and Systems, pp.1173–1176, 1985.

[11] J. Rabaey, J. Vanderwalle and H. De Man, "On the frequency domain analysis of switched capacitor networks including all parasitics", Proc. IEEE Int. Symp. on Circuits and Systems, April 1981, pp.868–871.

[12] C.K. Pun and J.I. Sewell, "Symbolic analysis of ideal and nonideal switched capacitor networks", Proc. IEEE Int. Symp. on Circuits and Systems, pp.1165–1172, 1985.

[13] C.K. Pun, A.G. Hall and J.I. Sewell, "Noise analysis of switched capacitor networks in symbolic form", Proc. 29th Midwest Symposium on Circuits and Systems, Lincoln, 1986.

[14] A.D. Meakin, J.I. Sewell and L.B. Wolovitz, "Techniques for improving the efficiency of analysis software for large switched–capacitor networks", Proc. 28th Midwest Symposium on Circuits and Systems, pp.390–393, Aug. 1985.

[15] D.B. Ribner and M.A. Copeland, "Computer–Aided analysis of nonideal linear switched–capacitor networks for high–frequency applications", Proc. IEEE Int. Symp. on Circuits and Systems, May 1986, pp.1169–1172.

[16] J. Vlach, K. Singhal, and M. Vlach, "Computer oriented formulation of equations and analysis of switched–capacitor networks", IEEE Trans. CAS, Vol. CAS–31, Sept. 1984, pp.753–765.

[17] J. Vanderwalle, H. De Man and J. Rabaey, "Time, frequency and z–domain modified nodal analysis of switched capacitor

networks", IEEE Trans. CAS, Vol. CAS-28, No. 3, pp.186-195, Mar. 1981.

[18] S.C. Fang, Y.P. Tsividis and O. Wing, "SWITCAP, a switched capacitor network analysis program", IEEE Circuit and Systems Magazine, pp.4-9 and pp.42-46, Dec. 1983.

[19] D.G. Haigh, B. Singh and J.E. Franca, "Filters for state of the art microelectronics fabrication", Annual Report, Imperial College, London, Sep. 1982.

[20] S.J. Harrold, I.A.W. Vance, J. Mun and D.G. Haigh, "A GaAs switched-capacitor bandpass filter IC", Proc. 1985 IEEE GaAs IC Symposium, Monterey, 4pp., November 1985.

[21] J.A. Nossek and G.C. Temes, "Switched-capacitor filter design using bilinear element modelling", IEEE Trans. CAS, Vol. CAS-27, No.6, pp.488-491, 1980.

[22] A. Fettweiss, D. Herbst, B. Hoefflinger, J. Pandel and R. Schweer, "MOS switched-capacitor filters using voltage inverter switches", IEEE Trans. CAS, Vol. CAS-27, No. 7, pp.527-538, 1980.

[23] P. Li, University of Glasgow, private communication.

[24] J.E. Franca, "Switched-capacitor systems for narrow bandpass filtering", Ph.D. Thesis, Imperial College of Science and Technology, London, 1985.

[25] J. Pandel, D. Bruckmann, A. Fettweiss, B.J. Hosticka, U. Kleine, R. Schweer and G. Zimmer, "Integrated 18th order pseudo N-path filter in VIS-SC technique", IEEE Trans. CAS, Vol. CAS-33, No. 2, pp.158-166, Feb. 1986.

CHAPTER FIVE

EFFICIENT METHODS FOR SOLVING $(zI - E)x = b$

5.1) INTRODUCTION

This chapter is concerned with methods that can be used to efficiently solve the system of equations

$$(zI - E)x = b \tag{5.1}$$

where z is a complex scalar

E is a constant, real N × N matrix and is full

x and b are complex vectors.

Matrices of this form commonly arise in frequency response calculations [1], and in particular the formulation developed in Chapter Four, which requires the solution of equation (4.35). In frequency response calculations the frequency variable z is varied over the desired range and equations of the form (5.1) have to be solved repeatedly. The matrix E is constant (independent of z) and the known RHS vector b can be frequency dependent or independent as both cases are considered.

From these characteristics of the problem a number of methods are explored that attempt to take advantage of these characteristics and thereby reduce the overall amount of computation to solve (5.1) over a range of values of z.

The first approach considered is the direct method of solving equation (5.1). This approach uses the LU decomposition methods discussed in Chapter Two. These methods have the distinct advantage that they are reliable, accurate and easy to implement. However the major drawback is that the solution requires $O(N^3)$ flops and cannot take advantage of the fact that matrix E is real, which necessitates the use of complex arithmetic throughout. Although computationally expensive, these methods do provide reliable solutions against which other methods can be compared.

The second approach is to use iterative methods for solving equation

(5.1). The motivation for this approach is that these methods require $O(N^2)$ flops per iteration and can take advantage of the fact that matrix E is real. Another advantage of this approach is that the solution of (5.1) for a particular frequency can be used as the starting guess for the iterations at another frequency. Now in a frequency sweep successive solutions generally do not vary greatly so the ability to use previous information can dramatically reduce the number of iterations. The last advantage of this approach is that because the methods iteratively construct better approximations to the exact solution the amount of computation is in proportion to the desired accuracy. Therefore in cases where full machine accuracy is not required this approach is able to reduce the number of iterations even further. From the above it is clear that this approach is competitive, provided that the number of iterations can be kept very low, certainly much less than N and ideally constant (independent of N). The major drawback of this approach is that the methods are not always convergent or converge very slowly.

The third approach considered is to first transform equation (5.1) to a simpler form where a direct method of solution requires significantly less computation, for example $O(N^2)$ or $O(N)$ flops. This transformation is only performed once, is independent of frequency and requires $O(N^3)$ flops. Because the transformation only involves transforming matrix E, real arithmetic may be used. The major advantages of this approach are that it reduces the computation to $O(N^2)$ flops for each solution of equation (5.1) and may be implemented to provide reliable and accurate solutions. The only drawbacks are perhaps the $O(N^3)$ flops required for the initial transformation and the reliability of one of the methods presented (the most efficient one).

To highlight the performance of these different approaches, results are presented from an actual implementation of the different methods in an analysis program. The application considered is the solution of equation (4.35) which is crucial for the efficient frequency analysis method presented in Chapter Four. The necessary modifications to the algorithm (4.49) are presented and the results clearly show the enormous gains in efficiency from the most efficient methods.

5.2) DIRECT METHODS

The direct method of solving equation (5.1) is to form the complex matrix

$$A = zI - E \tag{5.2}$$

and then factor A into a product of triangular matrices

$$A = LU \tag{5.3}$$

The solution is then obtained by solving the triangular systems using forward elimination and backsubstitution. As discussed in Chapter Two there are two methods of obtaining the LU factors, the Crout and Gauss elimination methods. Because the matrix E is full, sparse matrix techniques are not applicable and full algorithms are used. Both these methods are unable to exploit the fact that matrix E is real and that only the diagonal of matrix A is complex. In fact the L and U matrices obtained are both complex and therefore complex arithmetic has to be used throughout, which effectively quadruples the number of multiplications compared to the case where the matrices are real.

5.2.1) CROUT METHOD

The Crout method produces successive elements of the L and U matrices using the formulae

$$\ell_{ij} = a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj} \qquad i \geq j \tag{5.4}$$

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj}}{a_{ii}} \qquad i < j \tag{5.5}$$

This method has the advantage that the inner products in the formulae may be accumulated in extra precision to avoid roundoff errors [2]. In general the method provides accurate solutions, but suffers from instability for ill-conditioned matrices. To overcome this partial pivoting may be used whereby at each pivot step the element of largest magnitude in the pivot column is selected as the pivot. This process ensures that the growth of errors is bounded [2]. To further

increase the reliability of the solution, iterative refinement may be used to reduce any potential errors in the solution down to the machine precision of the computer [3].

The Crout method with partial pivoting, extra precision accumulation and iterative refinement is implemented in the NAG library routine F04ADF [4]. A library routine was used as this provides a thoroughly tested and reliable method for obtaining solutions, which are accurate to machine precision. The disadvantage of this approach is that it is slow, requiring $4/3N^3$ flops for the LU decomposition, $2N^2$ flops for the solution steps and $4N^2$ flops for each iterative refinement step. The overheads for partial pivoting and extra precision accumulation are quite considerable. However as this method is only used to provide reference solutions and not for production use, the speed is not important.

5.2.2) GAUSS ELIMINATION

The Gauss elimination method produces the L and U factors by organising the computations in a different way to the Crout method, though the Gauss elimination method with row normalisation produces the same LU factors as the Crout method. The kth step of this method is,

$$a_{ki} = a_{ki} \,/\, a_{kk} \qquad i = k+1,..,N$$

$$a_{ij} = a_{ij} - a_{ik} \times a_{kj} \qquad i,j = k+1,..,N \qquad (5.6)$$

This method was implemented without partial pivoting or iterative refinement to investigate the stability of the method when applied to matrices that typically arise from equation (4.35). It was found that the solutions obtained by this method compared exactly with those obtained by the NAG routine F04ADF. Therefore it was concluded that for this specific application partial pivoting and iterative refinement are not needed and this method could be used as a reliable reference. Although this method was typically at least twice as fast as the library routine (because there is much less overhead), the method still requires $O(N^3)$ flops and becomes impractical for large matrices.

5.3) ITERATIVE METHODS

These methods attempt to solve equation (5.1) by successively producing better approximations to the exact solution from an initial guess. The main motivation for this approach is that each iteration generally only requires $O(N^2)$ flops and therefore if the number of iterations can be kept low (prefably constant) then an overall $O(N^2)$ method is obtained. This approach has a number of advantages over the direct approach. The first is that the direct approach modifies the matrix and therefore an extra copy of the original matrix must be kept, whereas the iterative methods do not alter the matrix and therefore save this extra storage. The direct methods cannot exploit the information given by the solution in a nearby frequency point whereas the iterative approach can use these solutions as initial guesses. Finally the direct approach does not allow computational effort to be reduced when the desired accuracy is lower than the machine precision related accuracy, whereas the iterative approach has control over the accuracy. For this particular application the iterative approach has a further advantage as it allows the structure of the matrix (real with complex diagonal) to be exploited.

Two iterative methods are discussed, the first is the Gauss–Seidel (GS) method and the second is the Least–Squares Approach (LSA).

5.3.1) GAUSS–SEIDEL METHOD

The Gauss–Seidel method is perhaps the most well known iterative method as it is easy to implement and is quite effective for certain classes of matrices. The method is described by the algorithm

REPEAT

FOR i := 1 TO N DO

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} \times x_j - \sum_{j=i+1}^{N} a_{ij} \times x_j}{a_{ii}} \qquad (5.7)$$

END

UNTIL converged

The convergence criteria is usually

$$\left| x_i^{n+1} - x_i^n \right| < tolerance \qquad i = 1,..,N \qquad (5.8)$$

where x_i^n is the value of x_i at the nth iteration.

Applying algorithm (5.7) to equation (5.1), it is seen that a_{ij} in (5.7) is replaced by e_{ij} and $a_{ii} = z - e_{ii}$. Therefore this algorithm can take advantage of the fact that matrix E is real and thereby halve the amount of computation. To further improve efficiency the values b_i/a_{ii} may be precomputed and stored in a temporary vector and the division by a_{ii} replaced by multiplication by its inverse $1/(z - e_{ii})$, which again may be precomputed and stored. Taking these savings into account, each full iteration requires $2N^2$ flops. So provided that the method converges rapidly, it is potentially very efficient.

It can be shown [5, p.73] that a sufficient condition for the method to converge is that the matrix is diagonally dominant, i.e.

$$|a_{ii}| > \sum_{\substack{j=1 \\ i \neq j}}^{N} |a_{ij}| \qquad i = 1,..,N \qquad (5.9)$$

It was conjectured that the matrices arising from equation (4.35) would be diagonally dominant as the modulus of z is equal to 1 and the elements of matrix E are generally bounded by 1. However actual tests for diagonal dominance showed in fact that the matrices are not diagonally dominant. However condition (5.9) is a sufficient condition for convergence and not a necessary one and therefore the GS method could still prove to converge. Unfortunately tests verified that the GS method diverges for the systems of the form (5.1) which are derived from equation (4.35) and therefore is not suited to solving these equations.

5.3.2) LEAST SQUARES APPROACH

The least squares approach [6] has the distinct advantage over the GS method of being globally convergent, though convergence can sometimes be very slow. Before presenting the algorithm, some notation must be defined. Let $a_{(i)}$ denote the ith column of matrix A

and define the norm of the column $a_{(i)}$ as the Euclidean length $a_{(i)}{}^{T}a_{(i)}$,

$$\| a_{(i)} \|^2 = \sum_{j=1}^{N} a_{ji}^2 \tag{5.10}$$

The LSA constructs successive approximations to the least squares solution of the matrix equation such that the norm of the error is reduced at each iteration and is therefore guaranteed to converge. The algorithm is,

$\psi = b - Ax$
REPEAT
  FOR i := 1 TO N DO
    $\alpha = \psi^{T} \cdot a_{(i)} / \|a_{(i)}\|^2$
    $x_i = x_i + \alpha$ (5.11)
    $\psi = \psi - \alpha \times a_{(i)}$
  END
UNTIL converged

The convergence criteria (5.8) is used, except in this case the number of iterations was limited to N (as then the algorithm requires $O(N^3)$ flops and ceases to be of any benefit). As in the case of the GS method this algorithm can also take advantage of the fact that matrix E is real, though not quite as effectively as the GS method. Another slight disadvantage is that the vector ψ and scalar α are complex which doubles the amount of computation. The computation of the norms $\|a_{(i)}\|^2$ can be precomputed and the reciprocals stored, which leads to significant savings. A further saving is possible as all the elements in a column $a_{(k)}$ are real (and therefore independent of frequency) except for the diagonal elements $(z - e_{ij})$. Thus the partial sums excluding the diagonal elements may be computed before frequency analysis, which saves N^2 flops for each solution. Taking these savings into account, each full iteration requires approximately $4N^2$ flops.

The algorithm was implemented and tested on matrices which were known to converge rapidly and was found to perform very well. However when the algorithm was applied to the matrices arising from equation (4.35) it was found that the algorithm never converged within N iterations (in fact was still wildely out after N iterations). Therefore this algorithm too was found to be ineffective for solving equation (4.35) and therefore the iterative approach had to be abandoned.

5.4) REDUCTION TO SIMPLER FORMS

The objective of this approach is to transform equation (5.1) into a form such that the resulting system may be efficiently and accurately solved. The two methods considered both depend on finding similarity transformations of matrix E,

$$H = T^{-1}ET \qquad (5.12)$$

such that H is of a form which greatly simplifies the direct LU solution process and thereby dramatically reduces the computational cost. The requirement of such a transformation is that the matrix $(zI - H)$ retains the form of H, so that the transformation need only be done once, independent of the frequency variable z. Two such transformations are considered, the first reduces E to upper Hessenberg form and the second to tridiagonal form. The application of the Hessenberg approach to frequency analysis of linear systems was suggested in [1] and was applied to SC frequency analysis in [9] . The method has been very effectively applied to the solution of large systems of linear ordinary differential equations [7].

5.4.1) HESSENBERG APPROACH

An upper Hessenberg matrix H is defined as

$$h_{ij} = 0 \qquad j < i-1 \quad i = 1,..,N \qquad (5.13)$$

A general matrix E can always be reduced to Hessenberg form by stabilised elementary transformations [2],

$$H = T^{-1}P^{-1}EPT \qquad (5.14)$$

where T is a triangular transformation matrix and
P is a permutation matrix.

The transformation matrix T and Hessenberg matrix H can be determined in approximately $5/6N^3$ flops using real arithmetic throughout. A direct method analogous to the Crout method may be used which allows extra precision accumulation of inner products, the kth step of the direct method is [8],

$$h_{ik} = e_{ik} + \sum_{j=k+1}^{N} e_{ij} \times t_{jk} - \sum_{j=1}^{i-1} t_{ij} \times h_{jk} \qquad i = 1,..,k+1 \tag{5.15}$$

$$t_{i,k+1} = \frac{e_{ik} + \sum_{j=k+1}^{N} e_{ij} \times t_{jk} - \sum_{j=1}^{k} t_{ij} \times h_{jk}}{h_{k+1,k}} \qquad i = k+2,..,N \tag{5.16}$$

To avoid numerical instability the pivot $h_{k+1,k}$ is selected as the maximum of $|e_{ik}|$ for $i = k+1,..,N$ and the respective rows and columns interchanged. The algorithm determines the matrices in the following form,

$$\underset{E}{\begin{bmatrix} X & X & X & X \\ X & X & X & X \\ X & X & X & X \\ X & X & X & X \end{bmatrix}} \underset{PT}{\begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & X & 1 & \\ 0 & X & X & 1 \end{bmatrix}} = \underset{PT}{\begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & X & 1 & \\ 0 & X & X & 1 \end{bmatrix}} \underset{H}{\begin{bmatrix} X & X & X & X \\ X & X & X & X \\ & X & X & X \\ & & X & X \end{bmatrix}}$$

where blank entries are zero. In a practical implementation the triangular matrix PT is stored in the zero portion of the Hessenberg matrix as the unity diagonal and first column need not be stored. This then allows the algorithm to be performed inplace and no extra storage is needed.

Applying the stabilised elementary transformation to equation (5.1),

$$PT(zI - T^{-1}P^{-1}EPT)T^{-1}P^{-1}x = b \tag{5.17}$$

which may be written in a simplified form as,

$$(zI - H)y = b^P \tag{5.18}$$

where $H = T^{-1}P^{-1}EPT$

$y = T^{-1}P^{-1}x$

$b^P = T^{-1}P^{-1}b$

Inspecting equation (5.18) one sees that this transformation satisfies the requirement that $(zI-H)$ is upper Hessenberg. To form the preprocessed vector b^P, the inverse of T is not actually calculated as the equivalent operation is accurately and efficiently obtained by the process of backsubstitution using matrix T. If vector b is independent of z, then this need only be done once prior to frequency analysis and requires approximately N^2 flops.

Equation (5.18) is solved using the direct approach discussed in section 5.2. By taking advantage of the structure of the Hessenberg matrix, both in terms of zero/nonzero and real/complex structure, the computation is reduced from $4/3N^3$ flops to $2N^2$ flops. The forward elimination and backsubstitution steps together require approximately $2N^2$ flops. Finally the required solution x is determined by multiplying y by the transformation matrix which requires a further N^2 flops. The total solution process then requires approximately $5N^2$ flops which is a dramatic improvement over the direct approach.

To ensure numerical stability partial pivoting may be used in the LU decomposition process, which because of the structure of the Hessenberg matrix only requires the comparison of two elements (the diagonal and the element below it) and is therefore not too costly. Similarly iterative refinement may be used to improve the accuracy of the solution.

The simplified Gauss elimination algorithm was implemented without partial pivoting and iterative refinement and the solutions were found to agree exactly with those for the full approach. Therefore the extra safeguard for stability is not needed for this particular application. The Hessenberg method is therefore seen to be a very efficient method that is reliable and accurate.

5.4.2) TRIDIAGONAL APPROACH

The tridiagonal approach is an attempt to improve on the efficiency of the Hessenberg method. The motivation for this approach is that a tridiagonal matrix T, defined as,

$$t_{ij} = 0 \qquad j<i-1,\ j>i+1 \qquad i = 1,..,N \qquad (5.19)$$

may be LU decomposed in O(N) flops and a tridiagonal system solved in O(N) flops.

It can be shown, [2, p.396] that a lower Hessenberg matrix H^T can be reduced to tridiagonal form by a similarity transform,

$$T = Q^{-1} H^T Q \tag{5.20}$$

This transform is identical to the transformation of a general matrix to Hessenberg form, except that stabilisation is not allowed as this destroys the structure of the matrix. Therefore the same algorithms may be used by transposing all operations on the matrices, and may also be implemented inplace.

Consider the stabilised elementary transformation of a general matrix E to upper Hessenberg form

$$H = R^{-1}P^{-1}EPR \tag{5.21}$$

Transposing equation (5.20) gives,

$$T^T = Q^T H (Q^T)^{-1} \tag{5.22}$$

For notational simplicity the cumbersome $(Q^T)^{-1}$ is replaced by Q^{-T}. Substituting equation (5.21) into (5.22) gives the transformation of a general matrix E to tridiagonal form,

$$T^T = Q^T R^{-1} P^{-1} E P R Q^{-T} \tag{5.23}$$

Applying this transformation to equation (5.1) gives,

$$PRQ^{-T} (zI - Q^T R^{-1} P^{-1} E P R Q^{-T}) Q^T R^{-1} P^{-1} x = b \tag{5.24}$$

which may be written in the simplified form,

$$(zI - T^T) y = b^P \tag{5.25}$$

where $y = Q^T R^{-1} P^{-1} x$

$$b^P = Q^T R^{-1} P^{-1} b$$

The transpose of a tridiagonal matrix is still tridiagonal, so this reduction to tridiagonal form satisfies the requirement that $(zI - T^T)$ is tridiagonal. The transformation matrix Q^T and tridiagonal matrix T^T can be determined in approximately $1/6N^3$ flops, so the total cost of

transforming a general matrix to tridiagonal form is then N^3 flops. The elementary transformation algorithm presented in [2] is then modified to work on the transposed Hessenberg matrix H^T.
The kth step of the algorithm is,

```
FOR  i := k+2  TO  N  DO
    q_{k+1,i} = e_{ki} / e_{k,k+1}
    subtract  q_{k+1,i} × row (k+1) from row i
    add  q_{k+1,i} × column i to column (k+1)
END                                                    (5.26)
```

Because stabilisation is not allowed, the method is potentially unstable and therefore accuracy cannot be guaranteed, though the use of double precision arithmetic can improve the accuracy. If the pivot $e_{k,k+1}$ is zero then the algorithm fails as no row or column interchanges are allowed. In practice this problem has not been encountered for the many examples tested, but unfortunately this is not a guarantee that such a matrix will not occur.

The algorithm determines the matrices Q^T and T^T in the following form,

$$\underset{Q^T}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ & 1 & X & X \\ & & 1 & X \\ & & & 1 \end{bmatrix}} \underset{H}{\begin{bmatrix} X & X & & \\ X & X & X & \\ X & X & X & X \\ X & X & X & X \end{bmatrix}} = \underset{T^T}{\begin{bmatrix} X & X & & \\ X & X & X & \\ & X & X & X \\ & & X & X \end{bmatrix}} \underset{Q^T}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ & 1 & X & X \\ & & 1 & X \\ & & & 1 \end{bmatrix}}$$

where blank entries are zero. As in the Hessenberg case, the matrix Q^T may be stored in the upper zero portion of the matrix T^T as the unity diagonal and first row need not be stored.

Equation (5.25) may be solved using Gauss elimination. In this case the algorithm reduces dramatically and only requires 7N flops for the LU decomposition and 8N flops for the forward elimination and backsubstitution, giving a total of 15N flops. Partial pivoting may be used and is even simpler to implement than the Hessenberg case, as only two comparisons are required and only 3 elements per row are

swapped. Finally the required solution x is obtained by backsubstitution of y with Q^T and then multiplying this result by PR which requires a total of $2N^2$ flops. Therefore it is seen that even though the solution of the tridiagonal system only requires O(N) flops, the inverse transformation of the results still requires $O(N^2)$ flops and thus any possible transformation method will always require $O(N^2)$ flops. The total tridiagonal method therefore requires approximately $2N^2$ flops, which is believed to be the lowest cost attainable.

The tridiagonal method was implemented and tested on a large number of examples. It was found that for matrices of up to about order 60 the tridiagonal method without partial pivoting produced results which agreed exactly with the full LU methods. However for larger matrices instability sometimes occurred (though not always and some large matrices were successfully solved) and inaccurate results were obtained. Partial pivoting did not affect these inaccuracies and it was therefore concluded that the instability is introduced in the reduction to tridiagonal form. Unfortunately nothing can be done to eliminate this problem, except perhaps using extra precision arithmetic, which would be very costly and is still not guaranteed. Therefore the tridiagonal method cannot be unconditionally recommended and so the Hessenberg method, though $2\frac{1}{2}$ times slower, is the preferred method.

5.5) APPLICATION AND RESULTS

All three above approaches were implemented in the program FOOLSCAP as part of the frequency analysis presented in Chapter Four. The frequency analysis algorithm (4.49) has to be slightly modified to include the Hessenberg and tridiagonal methods. The modified algorithm for the Hessenberg method is,

A *Preprocessing independent of frequency and n*

1) Formulate the matrices G_k and C_k and the vectors W_k
2) Calculate P_k matrices using equation (4.16) and (4.21)
3) Calculate $B_{i,k}$ vectors using equation (4.22) for i = 0,..,m
4) Calculate matrix E using equation (4.31)
5) Calculate $F_{i,k}$ vectors using equation (4.34) for i = 0,..,m
6) Transform matrix E to upper Hessenberg form H using

equation (5.14)

7) Transform vectors $F_{i,k}$ and $B_{i,k}$ using equation (5.18)

B *Frequency analysis independent of n*

1) Prepare matrix $(e^{j\omega_0 T}I - H)$
2) Build RHS of equation (4.35) using equation (4.33)
3) Solve equation (4.35) for $V_N(e^{j\omega_0 T})$ using the Hessenberg method
4) Calculate $V_k(e^{j\omega_0 T})$ using equation (4.36)

C *Spectral analysis*

1) For selected n calculate weights $D_{k,n}$ using equations (4.47) and (4.48)
2) Calculate S_n using equation (4.46)

Similarly for the tridiagonal method the modified algorithm is,

A *Preprocessing independent of frequency and n*

1) Formulate the matrices G_k and C_k and the vectors W_k
2) Calculate P_k matrices using equation (4.16) and (4.21)
3) Calculate $B_{i,k}$ vectors using equation (4.22) for i = 0,..,m
4) Calculate matrix E using equation (4.31)
5) Calculate $F_{i,k}$ vectors using equation (4.34) for i = 0,..,m
6) Transform matrix E to tridiagonal form T^T using equation (5.23)
7) Transform vectors $F_{i,k}$ and $B_{i,k}$ using equation (5.25)

B *Frequency analysis independent of n*

1) Prepare matrix $(e^{j\omega_0 T}I - T^T)$
2) Build RHS of equation (4.35) using equation (4.33)
3) Solve equation (4.35) for $V_N(e^{j\omega_0 T})$ using the tridiagonal method
4) Calculate $V_k(e^{j\omega_0 T})$ using equation (4.36)

C *Spectral analysis*

1) For selected n calculate weights $D_{k,n}$ using equations (4.47) and (4.48)
2) Calculate S_n using equation (4.46)

The theoretical costs of the three different methods are determined and compared to actual timings obtained for a number of typical applications, given in Table 5.1. Apart from the preprocessing steps, the different methods differ only in the time required for solving the last slot as all the other slots are obtained by backsubstitution which requires a total of $2(M-1)N^2$ flops, where M is the number of slots.

The LU decomposition method requires approximately $4/3N^3$ flops for the LU decomposition which swamps the $O(N^2)$ flops required for the solution. The results of this method are given in Table 5.2. A graph of the CPU time required per frequency point for 2 phase networks is shown in Fig. 5.1. From this graph the N^3 dependence is clearly seen and the enormous benefit of the other methods is apparent.

The Hessenberg method requires a total of $5N^2$ flops, and so for a 2 phase network the total is $7N^2$ flops per frequency point. This dramatic saving over the LU method is clearly seen from the results in Table 5.3. The small increase in preprocessing time is for the extra processing required for the reduction to Hessenberg form ($5/6N^3$ flops) and the preprocessing of the excitation vectors (MN^2 flops). The N^2 dependence of the Hessenberg method is shown in Fig. 5.1.

The tridiagonal method requires a total of $2N^2$ flops, and so for a 2 phase network the total is $4N^2$ flops per frequency point, which is just slightly over half that required for the Hessenberg method. This result is verified in the results of Table 5.4. There is a slight increase in preprocessing time compared to the Hessenberg method, which accounts for the reduction to tridiagonal form ($1/6N^3$ flops) and the preprocessing of the excitation vectors (MN^2 flops). The N^2 dependence of the tridiagonal method and the improvement over the Hessenberg method is shown in Fig. 5.1.

From these results it is evident that the Hessenberg and tridiagonal methods can reap enormous savings compared to the full LU method, especially for large networks. Even though the tridiagonal method is faster than the Hessenberg method, the latter is recommended as it is thoroughly reliable even for large networks.

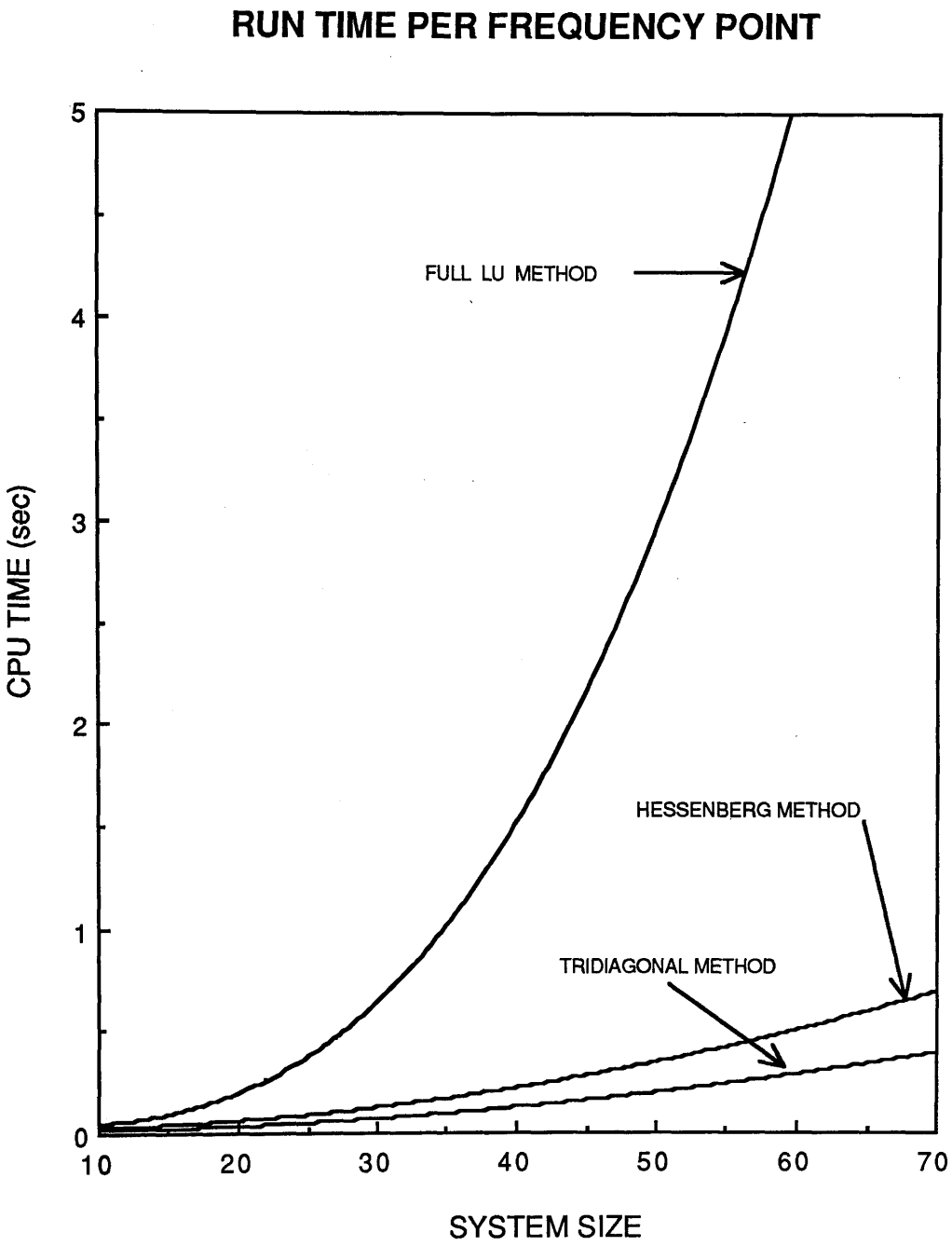


Figure 5.1 Comparison of execution speed of various methods

	DESCRIPTION
1	5th order elliptic lowpass filter [10]
2	6th order Chebyschev bandpass filter [10]
3	2nd order bandpass filter, Q = 20 [11]
4	5th order elliptic lowpass filter [12]
5	11th order elliptic lowpass filter [12]
6	7th order Chebyschev lowpass filter [13]
7	3rd order elliptic lowpass filter [13]
8	15th order elliptic lowpass, leapfrog design [14]
9	15th order elliptic lowpass, LUD design [14]
10	6th order elliptic bandpass filter [15]

Table 5.1 Examples used for comparison

EXAMPLE	NO. NODES	NO. SLOTS	PRE-Pr. (secs)	TIME/PT (secs)	STORAGE (words)
1	22	2	4.05	0.259	12288
2	28	2	5.13	0.527	16890
3	11	2	1.14	0.054	3664
4	20	4	5.10	0.230	11032
5	41	4	22.6	1.819	39944
6	30	6	11.8	0.688	27002
7	12	12	4.76	0.128	8520
8	70	2	40.6	8.168	94564
9	69	2	41.1	7.871	95210
10	27	2	5.79	0.471	17184

Table 5.2 Run statistics (LU method)

EXAMPLE	NO. NODES	NO. SLOTS	PRE-Pr. (secs)	TIME/PT (secs)	STORAGE (words)
1	22	2	4.25	0.075	12288
2	28	2	5.55	0.114	16890
3	11	2	1.14	0.034	3664
4	20	4	5.26	0.092	11032
5	41	4	22.7	0.320	39944
6	30	6	12.4	0.234	27002
7	12	12	4.82	0.099	8520
8	70	2	47.4	0.701	94564
9	69	2	47.6	0.679	95210
10	27	2	6.21	0.104	17184

Table 5.3 Run statistics (Hessenberg method)

EXAMPLE	NO. NODES	NO. SLOTS	PRE-Pr. (secs)	TIME/PT (secs)	STORAGE (words)
1	22	2	4.37	0.052	12288
2	28	2	5.76	0.070	16890
3	11	2	1.22	0.024	3664
4	20	4	5.37	0.072	11032
5	41	4	23.5	0.231	39944
6	30	6	12.8	0.189	27002
7	12	12	4.98	0.093	8520
8	70	2	49.0	0.390	94564
9	69	2	49.5	0.381	95210
10	27	2	6.46	0.067	17184

Table 5.4 Run statistics (Tridiagonal method)

REFERENCES FOR CHAPTER FIVE

[1] A.J. Laub, "Efficient multivariable frequency response computations", IEEE Trans. Automatic Control, Vol. AC-26, No. 2, pp.407-408, Apr. 1981.

[2] J.H. Wilkinson, "The algebraic eigenvalue problem", Clarendon Press, Oxford, 1965.

[3] J.H. Wilkinson and C. Reinsch, "Handbook for automatic computation", Vol. II Linear Algebra, Springer-Verlag, Berlin, 1971.

[4] NAG FORTRAN Library Manual, Mark 11, Vol.5, Numerical Algorithms Group, Oxford, 1984.

[5] R.S. Varga, "Matrix iterative analysis", Prentice-Hall, London, 1962.

[6] J. Staar, J. Vanderwalle and E. Van den Meersch, "An efficient numerically stable algorithm for the computer aided analysis of circuits and systems", Proc. IEEE Int. Symp. on Circuits and Systems, pp.218-221, May 1983.

[7] W. Enright, "On the efficient and reliable numerical solution of large linear systems of ODE's", IEEE Trans. Automatic Control, Vol. AC-24, No. 6, pp.905-908, Dec. 1979.

[8] R.S. Martin and J.H. Wilkinson, "Similarity reduction of a general matrix to Hessenberg form", Numer. Math., Vol. 12, pp.349-368, 1968.

[9] E.P. Rudd and R. Schaumann, "A program for the analysis of the high-frequency behaviour of switched capacitor networks", Proc. IEEE Int. Symp. on Circuits and Systems, pp.1173-1176, May 1985.

[10] D.G. Haigh, B. Singh and J.E. Franca, "Filters for state of the art microelectronics fabrication", Annual Report, Imperial College, London, Sep. 1982.

[11] S.J. Harrold, I.A.W. Vance, J. Mun and D.G. Haigh, "A GaAs switched–capacitor bandpass filter IC", Proc. 1985 IEEE GaAs IC Symposium, Monterey, 4pp., November 1985.

[12] J.A. Nossek and G.C. Temes, "Switched–capacitor filter design using bilinear element modelling", IEEE Trans. CAS, Vol. CAS–27, No.6, pp.488–491, 1980.

[13] A. Fettweiss, D. Herbst, B. Hoefflinger, J. Pandel and R. Schweer, "MOS switched–capacitor filters using voltage inverter switches", IEEE Trans. CAS, Vol. CAS–27, No. 7, pp.527–538, 1980.

[14] P. Li, University of Glasgow, private communication.

[15] J.E. Franca, "Switched–capacitor systems for narrow bandpass filtering", Ph.D. Thesis, Imperial College of Science and Technology, London, 1985.

CHAPTER SIX

CONCLUSION

6.1) CONCLUSIONS

This thesis has addressed the problem of developing techniques for analysing periodically switched linear networks in the time and frequency domains that are suited to computer implementation. The motivation for this work, which was discussed in Chapter 1, was the increasingly widespread use of switched capacitor networks and the corresponding growing need for computer aided analysis tools capable of efficiently analysing SC networks taking nonideal effects into account.

Very fast methods for the time and frequency domain analysis of periodically switched linear networks were developed and implemented in a computer program. The per point computation cost of these methods is $O(N^2)$ flops, which allows very large networks to be successfully analysed. It was demonstrated that these methods are orders of magnitude faster than existing nonideal analysis programs. This performance is due to the way in which the problem was formulated, the analysis methods used and the techniques developed to implement this analysis in a practical computer program. The thesis was divided into chapters each of which covers aspects of this work which are central to the overall efficient algorithms which were developed.

Chapter 2 addressed the problem of solving large sparse sets of complex linear equations. The concept of sparsity was extended to include the type of the nonzero elements, called domain types. This concept of domain types provided the key to the efficient approach developed to solve these equations. This approach used an interpretable code generation scheme capable of taking advantage of the domain type structure of the matrix. Three new optimal ordering algorithms were presented and were extensively compared with other ordering algorithms. The very substantial savings that can be achieved using domain type interpretable code generation in conjunction with a

domain type ordering algorithm was demonstrated.

Chapter 3 was concerned with the efficient time domain solution of linear networks with arbitrary inputs. A new approach based on the analytic solution, which uses numerical Laplace transform inversion and polynomial approximation of the excitations, was developed. The motivation for using polynomial approximations was to avoid the convolution in the analytic solution, which would otherwise be required for an arbitrary input. A new formula for polynomial approximation which calculates the coefficients of the polynomial explicitly was developed. An efficient technique for evaluating the numerical inverse Laplace transforms needed to calculate the extended state transition matrix and excitation response vectors was developed that makes optimal use of the sparsity of the matrices using the approach developed in Chapter 2.

Extensive results demonstrated that the time domain method is A–stable and is equivalent to high order numerical integration methods (which are not A–stable). The method is well suited to solving large stiff networks, which frequently arise in network analysis, and in particular SC networks with the large spread in resistance values associated with the 'on' and 'off' switches. This method was shown to be extremely efficient and orders of magnitude faster than numerical integration methods.

Chapter 4 presented the development of efficient time and frequency domain analysis methods for periodically switched linear networks. The time domain method of Chapter 3 was successfully generalised to periodically switched linear networks. This solution provided the basis for a very efficient technique for computing the time domain response of nonideal SC networks. The efficiency of the frequency domain method derives from three key aspects of the algorithm. The first is that the method, based on the time domain solution, uses polynomial approximation of the excitations, which avoids the need for AC analysis at each frequency point. This formulation enables the network to be modelled as a discrete system. A special system compression algorithm reduces the solution of this discrete system to the solution of the network in one slot only. This algorithm ensures that the

computational cost of the overall method increases linearly with the number of slots. The third key aspect is the method used to solve the compressed system. The cost of this solution, which is $O(N^3)$ flops using standard techniques, was dramatically reduced to $O(N^2)$ flops by using the Hessenberg method, presented in Chapter 5.

The detailed steps taken to verify the analysis methods and their implementation in a computer program were discussed. The performance of this implementation was compared with other methods. These results showed that the frequency domain analysis method is as accurate as the other analysis methods, yet is orders of magnitude faster and is capable of analysing very large networks, which the other methods fail to correctly analyse. This efficient method provides the basis for efficient sensitivity, noise and optimisation analyses.

Chapter 5 was concerned with methods for efficiently solving the complex linear equations derived from the compressed system produced by the frequency analysis method. Three different approaches were investigated and discussed in detail. The iterative approach was rejected because the methods either failed to converge or converged very slowly. The direct LU decomposition methods were found to be very accurate and reliable, but require $O(N^3)$ flops for each solution, which makes the solution of large matrices impractical. The reduction to simpler form approach proved to be very effective in transforming the cost of each solution to $O(N^2)$ flops. The Hessenberg method was found to be numerically stable and very reliable, even for large matrices. The tridiagonal method, though $2\frac{1}{2}$ times faster than the Hessenberg method, was found to be unstable for large matrices and was therefore rejected. The frequency domain algorithms were modified to include the Hessenberg and tridiagonal solution methods. Detailed comparisons of these methods and the direct approach clearly showed the substantial savings that can be achieved by the reduction to simpler form methods and that they are crucially important in transforming the per point computation cost of the frequency domain analysis method from $O(N^3)$ flops to $O(N^2)$ flops.

6.2) SUGGESTIONS OF FURTHER WORK

The application of the adjoint method to the frequency domain analysis method presented in this thesis would provide the basis for efficient sensitivity calculations, which in turn provides the basis for efficient optimisation, group delay and group delay sensitivity calculations. The adjoint technique provides a convenient and efficient means of simultaneously determining the transfer functions from every node of the network to the output [1], which then provides the basis for efficient noise analysis. Due to the sampled data nature of SC networks, high frequency noise is aliased into the baseband and therefore this noise foldover effect must be taken into account [2], [3]. The analysis method presented in Chapter 4 is well suited to this calculation as the frequency domain response may be calculated independently of n (the frequency band number). The aliasing effects are taken into account in a simple post processing step and therefore many frequency bands may be evaluated at very little extra cost.

The adjoint method requires the solution of the transpose of the discrete system used for frequency domain analysis. The block Gauss elimination algorithm for solving this discrete system may be adapted to efficiently solve the transpose system [4]. The Hessenberg method will have to be similarly extended to provide the efficient solution of the transposed system.

The sensitivity analysis requires the calculation of the sensitivities of the extended state transition matrices and excitation response vectors. These calculations could be efficiently performed by using the adjoint method in conjunction with the numerical inverse Laplace transform approximation. These sensitivities are frequency independent and may therefore be calculated in the preprocessing phase.

Three years ago it was not contemplated that the speed of a nonideal analysis program could be reduced as dramatically as has been demonstrated in this thesis. Further dramatic savings could be achieved by applying a compaction process [5] to reduce the size of the discrete system. This compaction process, also known as pivotal condensation, has been very effectively applied to the time and

frequency domain analysis of ideal SC networks [1], [6]. Pivotal condensation is applicable to the formulation presented in this thesis because the formulation does not use an AC analysis at each frequency point (unlike all the other nonideal analysis methods), and therefore the RHS vector may be compacted in a frequency independent step. This compaction process could reduce the system down to one row per slot corresponding to each output of interest, for all but the last slot. This slot cannot be reduced because of the frequency dependent diagonal in the matrix. This process would then reduce the overall computation cost to approximately $5N^2$ flops, the cost of solving the last slot using the Hessenberg method. The compaction algorithm would be of particular benefit for networks with many time–slots, particularly if the outputs are only sampled at a few slots, which would enable the intermediate slots to be effectively eliminated from the calculations.

Polynomial symbolic analysis can be used to provide additional insight into the behaviour of SC networks. Transfer functions can be produced as H(z) in the ideal case and H(s,z) in the nonideal case [7]. These polynomials have been used for frequency response and noise calculations [3]. For the formulation presented in this thesis, the transfer functions are functions only of z and therefore the complete nonideal response is given by H(z). This is in contrast to other formulations which require transfer functions in terms of z and the continuous frequency variable s. The exact interpretation of the symbolic transfer function H(z) will require further investigation and could have interesting application in deriving Z–domain equivalent circuit models of nonideal SC components. A number of different symbolic forms are possible, which require different methods for computing them. The polynomial interpolation method [8] can be used to determine the coefficients of the transfer function, and root finding procedures used to find the poles and zeroes. However this approach suffers from accuracy problems for bandpass and high–Q networks. An alternative approach is to use the QZ algorithm [9] to determine the eigenvalues of the system directly. This algorithm has the advantage that it is numerically stable and is well suited to this application as the first step of the algorithm is to transform the system to upper Hessenberg form, which is already done as part of

the preprocessing for the Hessenberg solution method. This algorithm provides a quick and accurate method for computing the poles and zeroes of the network. If required, the coefficients of the transfer function polynomials can be obtained from the pole/zero information.

Further theoretical work is required to prove the A–stability of the time domain solution method presented in Chapter 3. A thorough error analysis of the algorithm is desirable to study the effects of truncation errors and the propogation of roundoff errors. A detailed study of the accuracy of the inverse Laplace transform approximation used for the calculation of the extended state transition matrix and excitation response vectors would be useful for obtaining strict bounds on the error of the time domain method.

Other forms of approximation of the excitations could possibly provide a more efficient or more accurate method, though perhaps at the expense of the simplicity and ease of implementation of the polynomial method. In particular a trigonometric approximation would appear to be attractive for the special case of sinusoidal input as the approximation would be exact and would only require one term. This approximation could then have important implications for the frequency domain analysis method which uses sinusoidal excitations.

The computation of the extended state transition matrices and excitation response vectors using the I_{MN} approximant is a relatively costly process. It is unlikely that the efficiency of the present algorithm could be improved much, therefore to significantly reduce this computation other techniques could be investigated. The techniques would be required to be numerically stable and able to handle large stiff systems. Iterative methods or methods that can tradeoff speed and accuracy would be of particular interest because sensitivity studies of the influence of the approximations on the time and frequency domain algorithms would allow the accuracy of the approximations to be controlled, which could significantly reduce the computation cost.

Finally, with the increasing availability and use of vector and parallel processors, the implementation of the time and frequency domain algorithms on these processors could be considered. These algorithms

are highly suited to vectorisation as almost all calculations are vector operations. The algorithms also have a high degree of parallelism, which could be exploited on a parallel processor. The only part of the algorithms which use strictly serial and scalar computations is the sparse matrix computations used in the calculation of the extended state transition matrices and excitation response vectors. Further investigation would be required to establish whether vectorised full matrix algorithms could be implemented that are faster than the scalar sparse matrix algorithms.

REFERENCES FOR CHAPTER SIX

[1] J. Vanderwalle, L. Claesen and H. De Man, "A very efficient computer algorithm for direct frequency, aliasing and sensitivity analysis of switched capacitor networks", Proc. IEEE Int. Symp. on Circuits and Systems, pp.856–859, 1981.

[2] J. Vanderwalle, H. De Man and J. Rabaey, "The adjoint switched capacitor network and its application to frequency, noise and sensitivity analysis", Int. J. Circuit Theory Appl., Vol. 9, pp.77–88, 1981.

[3] C.K. Pun, A.G. Hall and J.I. Sewell, "Noise analysis of switched capacitor networks in symbolic form", Proc. 29th Midwest Symposium on Circuits and Systems, Lincoln, 1986.

[4] J. Vlach, K. Singhal, and M. Vlach, "Computer oriented formulation of equations and analysis of switched–capacitor networks", IEEE Trans. CAS, Vol. CAS–31, Sept. 1984, pp.753–765.

[5] L.B. Wolovitz, "Improved techniques for time domain analysis of switched capacitor networks", M.Sc. Thesis, University of Hull, 1986.

[6] A.D. Meakin, J.I. Sewell and L.B. Wolovitz, "Techniques for improving the efficiency of analysis software for large switched–capacitor networks", Proc. 28th Midwest Symposium on Circuits and Systems, pp.390–393, Aug. 1985.

[7] C.K. Pun and J.I. Sewell, "Symbolic analysis of ideal and nonideal switched capacitor networks", Proc. IEEE Int. Symp. on Circuits and Systems, pp.1165–1172, 1985.

[8] K. Singhal and J. Vlach, "Symbolic analysis of analog and digital networks", IEEE Trans. CAS, Vol. CAS–24, pp.598–609, Nov. 1977.

[9] C.B. Moler and G.W. Stewart, "An algorithm for generalized matrix eigenvalue problems", SIAM J. Numer. Anal., Vol. 10, pp.241−256, Apr. 1973.