AUTOMATIC STABILITY CHECKING FOR LARGE ANALOG CIRCUITS

A Thesis

by

PARIJAT MUKHERJEE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2010

Major Subject: Computer Engineering

AUTOMATIC STABILITY ANALYSIS FOR LARGE ANALOG CIRCUITS

A Thesis

by

PARIJAT MUKHERJEE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2010

Major Subject: Computer Engineering

ABSTRACT

Automatic Stability Checking
for Large Analog Circuits. (December 2010)

Parijat Mukherjee, B.Tech, National Institute of Technology, Tiruchirappalli
Chair of Advisory Committee: Dr. Peng Li

Small signal stability has always been an important concern for analog designers. Recent advances such as the *Loop Finder* algorithm allows designers to detect and identify local, potentially unstable return loops without the need to identify and add breakpoints. However, this method suffers from extremely high time and memory complexity and thus cannot be scaled to very large analog circuits. In this research work, we first take an in-depth look at the loop finder algorithm so as to identify certain key enhancements that can be made to overcome these shortcomings. We next propose pole discovery and impedance computation methods that address these shortcomings by exploring only a certain region of interest in the s-plane. The reduced time and memory complexity obtained via the new methodology allows us to extend automatic stability checking to much larger circuits than was previously possible.

ACKNOWLEDGMENTS

I would like to thank my committee chair Dr. Peng Li for being a guiding force throughout the course of this research and my committee members Dr. Gwan Choi and Dr. Vivek Sarin for their support.

Special thanks go to Peter Fang and Rod Burt of Texas Instruments for sharing with us the preliminary work on which this research is based and for providing valuable insight and guidance in terms of designer requirements. Thanks also to Texas Instruments for providing the funding, the tools and the testcases used in this research.

Last but not the least, I would like to thank my friends and colleagues and the department faculty and staff for providing an environment so conducive to learning and research as well as making my time in Texas A&M University most pleasant.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Motivation

Small signal stability has always been a serious concern while designing analog circuits. While several methods (eg. [1, 2]) exist to analyze closed loop stability, all these methods either involve a detailed understanding of the analog circuit in question or do not scale too well to very large circuits with a large number of feedback loops.

Additionally, the more the number of parasatics that come into play, the larger the chances that local instability may exist around certain transistors or groups of transistors are greatly increased. Thus as our feature size reduces and the size of our analog circuits becomes larger, it becomes not just desirable but necessary to run stability analysis on extremely large extracted netlists.

To be truly useful, such a stability analysis method should not just identify whether a potential instability exists in the circuit in question but also aid the designer in narrowing down the problem to a specific portion of the circuit.

Existing stability analysis methods do not fulfill this growing need and hence forms the basis for this research work.

B. Stability of analog circuits

Feedback loops are of great importance to analog designers. A typical feedback loop can be described by the block diagram given in Fig. 1 where α defines the open loop transfer function and β defines the transfer function of the return path.

The journal model is *IEEE Transactions on Automatic Control.*

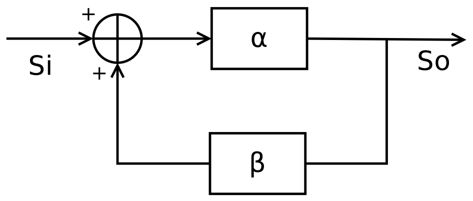


Fig. 1. A typical feedback loop

The closed loop transfer function of such a loop will be given by

$$\frac{s_o}{s_i} = \frac{\alpha}{1-\alpha\beta} \tag{1.1}$$

Such feedback networks can be found in a wide range of circuits dealing with an even wider range of applications be it oscillators, amplifiers, receivers, phased lock loops among many others. However, depending upon the design, any feedback loop is susceptible to self-oscillations. In few cases such as in oscillator design, such self-oscillation may be desirable.

A large majoirity of feedback loops used in analog circuits or RF design are however, negative feedback loops. These are often used to improve various circuit properties which may include bandwidth, impedance matching, variation tolerance, output waveform distortion and many others. Such circuits span over a large range of applications but irrespective of the target application, self-oscillations interfere with the normal functioning of the circuit.

Also, unwanted return loops also exist around individual transistors or groups of transistors. As our feature size continues to shrink, the relative effect of parasatics increase leading to greater chances of having to deal with such unwanted return loops.

Whether a feedback loop is added by design or accident, stability is always a serious concern for feedback based circuits. Unless introduced by design such as in oscillator design, problematic loops in a circuit may cause self-oscillation or increased susceptance to injected noise. Either of these can degrade circuit performance significantly and interfere with our desired circuit response.

Thus it is critical to evaluate stability and stability margin of a feedback circuit. Such information can be used to identify and eliminate potential stability issue before a circuit ever reaches the fabrication stage. This method can be applied to both the initial design stages (schematic design) or during the physical realisation stages (layout design, parasatic extraction etc).

This thesis will deal with the detection and identification of potentially unstable feedback loops. The actual task of fixing these loops to remove such instabilities is left upto the designer.

C. Traditional stability analysis methods

Feedback loop design methodologies are generally based upon using open loop transfer functions to characterise the closed loop response. However when applied to analysis of arbitrary feedback loops, opening the loop to compute the loop gain is not straightforward. One problem is that the DC operating point on both sides of the opening are usually different. Even if we identify a point where we can safely break open the circuit without affecting the DC operating point(for actual laboratory measurement) or we are dealing with an already linearised circuit(for circuit analysis), the small-signal ac impedances seen on both sides are different from the closed loop case. As we move from single loop to multiloop systems, these problems only get more complicated.

Thus our classical feedback loop design strategies does not automatically extend to analysis techniques and thus a need for a method to make these measurements without opening up the loop was identified. One such method was proposed by R.D. Middlebrook in 1975[1]. He discusses practical methods of measuring and interpreting the results of the closed loop system by a voltage injection or current injection technique. He further extends these methods to the case in which measurements can be made even if the system is unstable. This in turn provides to be the basis for stability analysis using the Middlebrook method. However the Middlebrook method also required the identification of an injection point at which to perform current or voltage injection. A "null double-injection" method was also discussed addressing the shortcomings of voltage or current injection when used individually. However since all these methods involve identifying a breakpoint where current or voltage injection could be performed, they all require a detailed understanding of the loop in question.

In 2001, Tian et all[2] built upon the null double-injection method by Middlebroook and proposed loop and device based algorithms for stability analysis of linear analog circuits. While this method did have the ability to deal with loop instabilities and local loops based around transistors, it still followed the common practice of analysing stability of intrinsically multiloop structures using single-loop theory. So while both the loop and device based algorithms could be applied to multiloop networks, the methodology worked only if we could ignore either the reverse transmission effects or local return loops. Also, it shares the same requirement as the Middlebrook method namely the identification of a breakpoint making it hard to extend to stability analysis over a large number of loops.

Besides the work presented here, several other methods have also been proposed to analyze closed loop gain but again they are all dependent upon being able to identify a breakpoint for every feedback loop. Identifying breakpoints for increasingly complex circuits with a large number of feedback loops is not a trivial process. It requires both a very detailed understanding of the circuit and is extremely time consuming since every loop needs to be examined seperately. As we design larger and larger circuits with more and more loops, it becomes increasingly difficult to perform stability analysis in such a fashion.

D. Automatic stability checking for large analog circuits

If we are to look at equation 1.1, the feedback factor $1 - \alpha\beta$ characterizes the stability of a loop. As $\alpha\beta$ approaches unity, our closed loop response approaches ∞. In fact Barkhausen criterion[3] states that the frequency of a linear(ized) oscillator is determined by the frequency point at which the phase shift $\angle\alpha\beta$ is $2\pi n, n \in \{1, 2, 3...\}$ provided that the magnitude $|\alpha\beta|$ also equals unity. If we are designing any other circuit such as an amplifier where the oscillatory behaviour is undesirable we need to make sure the circuit operates as far away as possible from this critical point over its entire frequency range of operation.

Thus if we are to look at a feedback network from a control system perspective, simply analyzing the zeros of the feedback factor $(1 - \alpha\beta)$ should give us the poles of the network. By analysing the poles of our system, we can identify the phase margin of the system as a whole and thus its stability. Since we are interested in potentially unstable systems, the poles we are most interested in are complex poles within a specified region of the s-plane as shall be discussed later.

Traditionally pole-zero analysis has not been preferred for performing stability analysis because of the numerical difficulties when dealing with large networks. Also, pole-zero analysis of the transfer function between inputs or outputs may not accurately capture potential instabilities within the circuit itself. Lastly, when dealing with simple circuits with one to a few loops, traditional methods to obtain stability margin such as gain margin and phase margin are preferred.

However, as we move on to analyze bigger and bigger circuits, as discussed in section I.A, we are more interested in detecting whether or not circuit instabilities exist and if they do, where on the circuit to look for them. Note that this work is not meant to replace existing stability analysis methods. It is only meant to serve as a tool to validate large circuits once they have been designed and to point out errors if any.

The concept of studying stability using pole-zero analysis has been around for a very long time. However only recently has it been applied to unstable loop detection as in [4]. The main idea behind the loop finder algorithm as described in [4] is extracting second order approximations for individual impedance transfer functions and then detecting the dominant unstable second order systems for each node to form a "loop".

As will be described in chapter II, extraction of second order systems from a large lumped linear system for loop finder analysis is a two step process. The first is extracting the actual poles of the system and the second is finding the residues. One of the first observations we shall make at this point is that only very few poles are actually of interest as we are only interested in complex poles that satisfy a certain set of conditions namely,

$$\begin{aligned} |p_n| &< 2\pi f_{max} \\ \frac{-p_{n,r}}{|p_n|} &< \zeta_{th} \end{aligned} \tag{1.2}$$

The loop finder algorithm described in [4] uses the QZ method[5] to first detect these poles and then compute their residues. However this proves to be computationally extremely expensive given that the QZ method is of complexity $O\left(N^3\right)$.

In chapter II, we shall first discuss some background work and in particular the loop finder algorithm so as to identify potential performance improvements that can be made to the algorithm. In chapter III, we develop a pole discovery method that explores only our region of interest. Once we have our poles of interest, we shall look into how to compute residues for only those poles in chapter IV using model order reduction techniques. Lastly in chapter V, we shall present experimental results based on some real circuits before discussing our conclusions in chapter VI.

CHAPTER II

BACKGROUND

In this chapter we shall describe the concepts and background work that made this thesis possible. We shall start off by explaining the general feedback equation and how it extends to multiloop circuits. We shall then discuss second order approximations for any arbitrary multiloop structure and how it helps us in predicting stability issues. Lastly we provide an in-depth analysis of the various steps involved in the loop finder algorithm as described in [4]. This will include discussing stability of an arbitrary transfer function, how we can obtain the poles of this transfer function, impedances of second order transfer functions and how it in turn corresponds to the concept of "dominant poles" and "loops".

A. Feedback equation and stability of multiloop systems

First we shall look at the feedback equation for a single loop system and how it determines the stability of the loop. We shall then proceed to try and extend this concept to a generalised multiloop system.

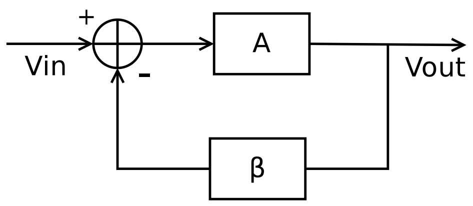


Fig. 2. A typical negative feedback loop in electrical circuits

A classical representation of an ideal single-loop feedback system has already been presented in Fig. 1. Expressing the input and output signals in electrical terms and assuming negative feedback, we get the feedback loop as given in Fig. 2. To ensure that we do not confuse the positive feedback representation shown in Fig. 1 with the negative feedback shown in Fig. 2, the open loop gain of this new loop is defined by A instead of α.

Using block diagram reduction concepts, we know that

$$V_{out} = EA \tag{2.1}$$

$$E = V_{in} - \beta V_{out} \tag{2.2}$$

Combining the two equations we get

$$\frac{V_{out}}{A} = V_{in} - \beta V_{out} \tag{2.3}$$

$$V_{out}\left(\frac{1}{A} + \beta\right) = V_{in} \tag{2.4}$$

$$\frac{V_{out}}{V_{in}} = \frac{A}{1 + A\beta} \tag{2.5}$$

The last equation gives us the classical form of the feedback equation. It describes the closed loop transfer function in the most generalised form since A and β can take any value. The quantity $A\beta$ is called loop gain and has a very special meaning in stability analysis. Since A and β are complex numbers, the loop gain $A\beta$ can take any value including negative values. When the loop gain approaches -1, our close loop response V_{out}/V_{in} approaches $\inf$. This means that the output rises till it hits up against the power rail. Once it hits the power rail, it can become uniformly high or it can reverse direction because of capacitive elements and then oscillate in the other direction. It is this kind of ringing / overshoot that we wish to prevent. The stability information of our loop can thus be obtained from our loop gain $A\beta$.

It is to be noted however that the ideal single-loop feedback network depicted in Fig. 2 is not an accurate respresentation of a practical feedback system involving multiple loops. In practice, the feedback path may not be strictly unilateral and the input and output coupling networks are often complicated[6]. Also, since the impedance characteristics of each node are different, we may observe different loading conditions as well. A general feedback network with input and output coupling as described in [6] is shown in Fig. 3.

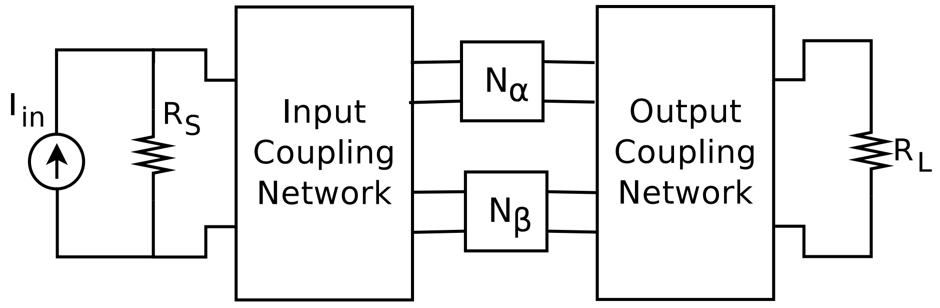


Fig. 3. A general feedback network with input and output coupling

It is to be noted that since the loop gain $(1 + A\beta)$ is still uniquely determined by the loop parameters alone, for a given loop, under an excitation of a given frequency, we would still expect to be able to detect these instabilities between any two points on the loop. Note that this argument does not hold true for two nodes lying on two different loops since they might not be coupled together. Also, there might be a unique case where the network zeros cancel out the effect of poles at the natural frequency making it difficult for us to accurately determine the stability measure using a simulation based approach.

B. The second order equation and ringing/overshoot predictions

The second order equation is a common approximation used for feedback system analysis because it deals with a two-pole circuit which is commonly used by designers to debug and fix such loops. Practical real world circuits are of course more complex than just two poles but most can be represented by a two pole equivalent[7]. Additionally, the second order approximation easily lends itself easily to stability analysis as will be explained by applying a step input to an unstable second order system.

Assuming that the loop gain $A\beta$ can be expressed as a second order system, we have

$$\begin{aligned}(1+A\beta) &= 1+\frac{K}{(1+\tau_1 s)(1+\tau_2 s)} && (2.6)\\ s^2+s\frac{\tau_1+\tau_2}{\tau_1\tau_2}+\frac{1+K}{\tau_1\tau_2} &= 0 && (2.7)\\ s^2+2\zeta w_n s+w_n^2 &= 0 && (2.8)\end{aligned}$$

What this means is that if we have a means to identify ζ and w_n, we also have a means to identify the stability of the second order approximation we are studying. Note that this second order approximation is highly frequency dependent as $s = j\omega$ can take any value over our frequency range of operation. Thus different second order systems may become dominant at different frequencies.

One of the ways to detect local instabilities has been to apply a pulse to the nodes in the suspected loop and observe the transient response. If the output shows significant ringing before settling down, it identifies that the node might be on a potentially unstable loop[8].

The basic concept behind this "node pulsing" technique works especially well for our second order approximations since the response of our second order approximation to a step or pulse input can be easily plotted if our ζ and w_n are known. ζ and w_n have a direct relation to the percentage overshoot and the frequency of any ringing that might takes place as shown in Fig. 4.

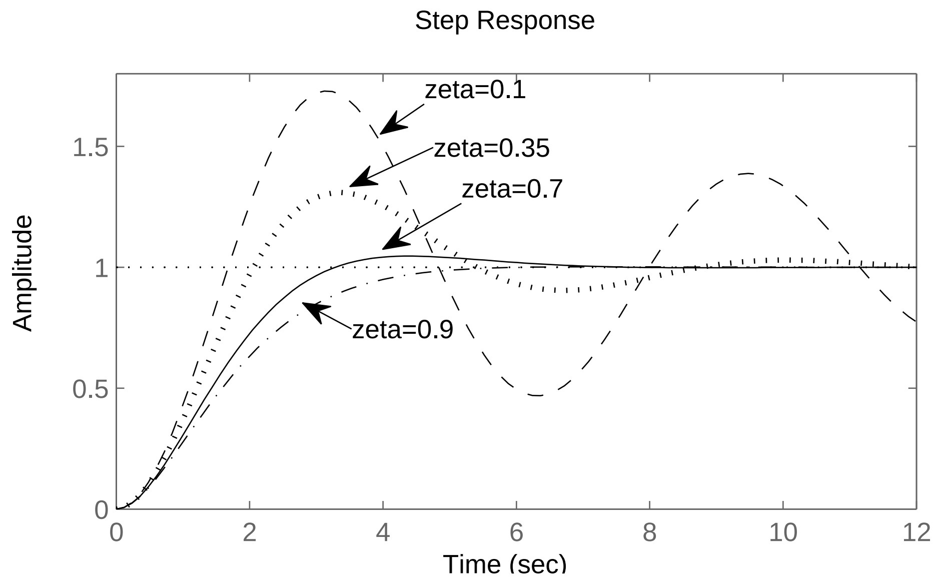


Fig. 4. Step response in time domain

However, while this method works well for smaller circuits or to confirm the existence of a suspected loop in a given portion of the circuit, it does not scale well to automatic stability checking as it involves running transient simulations over a large time period on a large number of nodes.

In 2004, Rod Burt developed a technique to evaluate local return loops in the frequency domain[9]. The key idea of his work is to extract dominant second-order under-damped continuous-time systems from frequency responses. This can be easily done since ζ and w_n have a very real effect on the frequency response as well as shown in Fig. 5

Originally, Milev and Burt developed an AC-stability tool that identifies second order systems by post processing AC waveforms[10]. However, this method required a large number of AC simulations over a wide frequency range which proves to be extremely time consuming. Numerical errors have also been reported in [4] owing

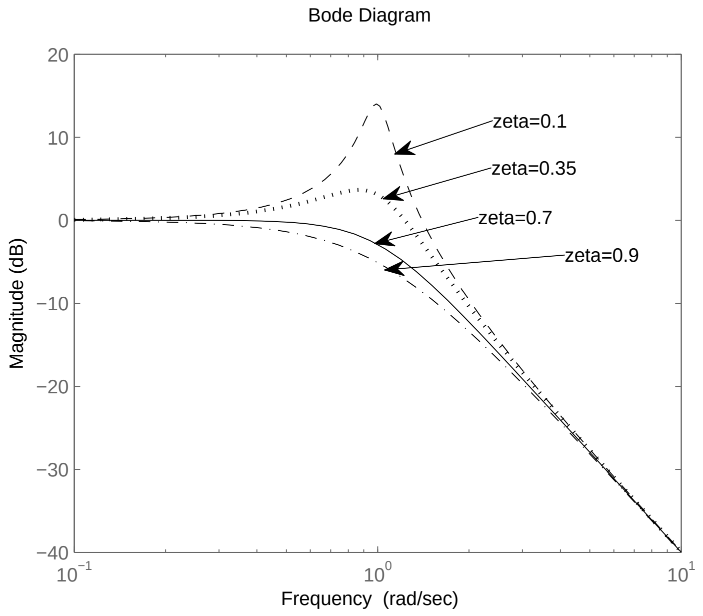


Fig. 5. Step response in frequency domain

to the use of numerical differentiation. Additionally, there is also the "notch problem" which is introduced when the presence of inductors causes the introduction of a network zero that cancels the effect of our potentially unstable pole value on an AC signal.

Instead of post-processing AC waveforms, it is possible to explicitly form system transfer functions and extract second order systems directly from the transfer function as shall be described in the next few sections.

C. Loop detection using pole-zero information

As we have discussed in the previous section, applying a step input to any of the nodes in an unstable loop will potentially give rise to overshoot/ringing. In reality however, we cannot accurately analyze stability using an arbitrary transfer function between any two nodes since to be able to detect a loop, both our nodes must be on the unstable feedback path to pick up the corresponding oscillations. Also, we wish not only to detect the existence of circuit instabilities but to try and determine where they are located. Clearly we wish to take a node transfer function based approach to node pulsing so that the response of any given node allows us to detect instabilities local to that node.

1. Impedance transfer function and local instability

When we talk about node transfer functions that we can use to detect circuit instabilities, two possible transfer functions come into mind. The driving point impedance and the driving point admittance. One gives us the voltage response of the node to an arbitrary current input. The other the current response to an arbitrary voltage input. The two are complementary to each other and are connected by the relationship

$$Y(s) = \frac{1}{Z(s)} \tag{2.9}$$

where $Y(s)$ represents the driving point admittance and $Z(s)$ represents the driving point impedance.

To understand how the driving point impedance can be used to determine whether a node lies on an unstable loop or not, instead of considering the step response we have been considering so far, let us consider the response of a node to current noise of a given frequency. If the loop is unstable, and the frequency of the disturbance is equal to the natural frequency of the loop, it will cause the loop to oscillate (as described in section II.A). Since injecting a very small amplitude current signal is able to give rise to extremely high voltage oscillation, we can say that the driving point impedance is going to be high for all the nodes in loop i as per equation 2.10.

$$Z_{in}\left(s_i\right) = \frac{V_{in}\left(s_i\right)}{I_{in}\left(s_i\right)} \tag{2.10}$$

The mechanism we use to detect local instabilities as described above tallies closely with how self-oscillations happen in the first place. Self oscillations are normally triggered by either injected noise or by noise generated within the device. As long as the node where this noise is being injected lies on an unstable loop with the same natural frequency as the noise excitation, the current signal will get amplified by the loop in question and manifest itself as a large voltage response.

Note that current injection at every node in a laboratory or simulation setup is not straightforward as it involves identifying ideal breakpoints without modifying the DC operating point and small signal impedance. It is thus complicated and may not be possible at every node. However if we are to observe it strictly from a control system perspective and deal only with the transfer functions, this is actually straightfoward to do via eigen methods.

2. Identification of dominant poles

Since we are going to discuss the concept of "dominant" second order approximations to identify loops, now would be a good time to discuss what makes a second order system "dominant".

Consider the thevenin equivalent of a noise source operating at a given frequency s_i as given in Fig. 6

Applying the voltage divider rule to compute V_{out} we get,

$$\begin{aligned} V_{out} &= \frac{Z_{in}}{Z_{in} + Z_{source}} V_{noise} \\ &= \frac{1}{1 + \frac{Z_{source}}{Z_{in}}} V_{noise} \end{aligned} \tag{2.11}$$

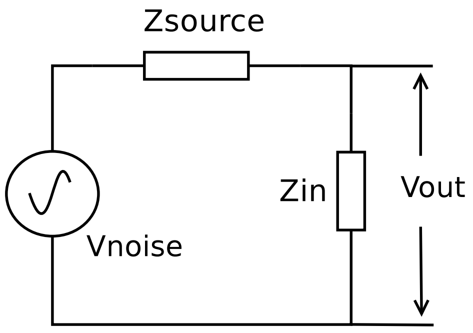


Fig. 6. Noise source represented in thevenin form

Since a high value of V_{out} means a high amplitude ringing, we wish to make sure $V_{out} \ll V_{noise}$. To do this we have to make sure that $Z_{source} \gg Z_{in}$. Thus in the presence of two or more loops with a significant input impedance Z_{in}, the one with the highest $Z_{in}(s_i)$ to a particular frequency s_i is most likely to cause undesirable ringing at that frequency and hence is the ones we wish to examine.

Seen from another perspective, if we have two or more second order systems with significant impedances, we will wish to choose the one with the highest impedance to approximate the transfer function at that frequency point since it is better able to approximate the response of the system as a whole at that frequency point.

Thus the dominant poles of a second order system can be defined as a set of second order systems that together are able to best capture the response of the system over the entire frequency spectrum. Note that when talking about dominant second order systems, these poles are dominant over different frequency points. At an arbitrary frequency s_i, we will consider only one pole to be dominant. This is demonstrated in Fig. 7 where we compare the second order frequency responses of various unstable poles at a particular node.

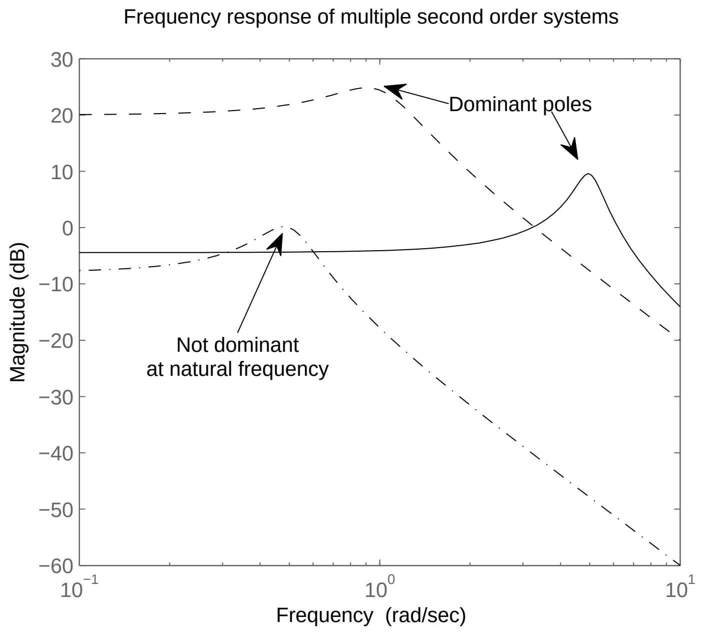


Fig. 7. Dominant pole identification from frequency response

3. Using dominant pole information to identify loops

Since the dominant poles we have discovered above each have a different natural frequency w_n and a different damping factor β, we would expect them to belong to physically seperate loops as well. This is because the unstable oscillatory behaviour is caused by the loop parameters alone as discussed in section II.A.

One of the first things to note here is that it is assumed that the system response can be adequately described by a second-order system transfer function for a given loop. The relative effect of all the loops in the circuit are considered to be additive and it has been assumed that the complex frequencies that cause the circuit to oscillate are dominant on the nodes that form the loop. Any of these assumptions can prove to be false for an arbitrary system but in most realistic circuits, this proves to be valid.

Since we already know that the nodes on an unstable loop can be used to perform node pulsing and detect a potential instability, the reverse should also hold true. That is, if we know nodes which share a common ζ and w_n, we should be able to detect the physical loop. However as we know, a multiloop circuit is extremely complex due to its coupling effects. In other words, no loop is independent from other loops and every node is potentially affected by all the loops in the circuit.

In such a situation, if we are to simply look at whether or not the impedance transfer function is affected by our unstable poles of interest, we might potentially end up including the entire circuit into the loop. Hence the concept of using our dominant poles such that we include nodes only into the loop that their response is dominated by. It is still possible that the node may also belong to loops that we have not included it in but such cases will be few and far between. Even if we do accidentally neglect to include a node into a loop, remember that fixing an analog circuit is an iterative procedure and once we deal with the loop that causes the dominant response at a node, we will be able to see the effect of the other poles as well.

D. Obtaining second order systems using eigen methods

In this section we shall first examine how second order approximations for various natural frequencies can be obtained from the transfer function between any two nodes. Furthermore we shall discuss how to obtain the poles of this transfer function by solving a generalised eigen value problem. Please note that we are only talking about detecting the circuit instabilities and not identifying where they lie at this point of time. The pole values obtained at this stage will be used to generate second order approximations at every node to perform loop identification.

1. System transfer function and second order systems

A time-invariant electrical circuit can be described by the system of linear first-order differential algebraic equations

$$\begin{aligned} C\dot{x}(t) &= -Gx(t) + Bu(t) \\ y(t) &= L^T x(t) \end{aligned} \tag{2.12}$$

The matrix G contains memoryless elements such as resistors, the matrix C contains memory elements such as capacitors and inductors. The vector $x(t)$ contains the state variables. $u(t)$ is the known input and $y(t)$ is the known output.

Applying laplace transform to this system and assuming zero initial condition, we get equation 2.13

$$\begin{aligned} sCx(s) &= -Gx(s) + Bu(s) \\ y(s) &= L^T x(s) \end{aligned} \tag{2.13}$$

Thus the laplace domain transfer function $H(s) = y(s)/u(s)$ can be expressed as

$$H(s) = L^T (G + sC)^{-1} B \tag{2.14}$$

In general, a circuit linearized around its DC operating point is a lumped high-order linear continous time system. Assuming a single input single output case (without worrying about what our input and output nodes are), we obtain a high order rational function as given in the equation 2.15. The poles p_i can be either real or complex.

$$H(s) = \frac{r_{n-1}(s)}{(s-p_1)(s-p_2)(s-p_3)\ldots(s-p_n)} \tag{2.15}$$

A high order rational transfer function such as the one given above can be broken up into a number of first order systems as given in equation 2.16

$$H(s) = \frac{k_1}{(s-p_1)} + \frac{k_2}{(s-p_2)} + \frac{k_3}{(s-p_3)} + ... + \frac{k_n}{(s-p_n)} \tag{2.16}$$

Splitting into real and complex conjugate pairs, we get a linear combination of multiple first order and second order systems. N_R corresponds to the number of real poles and thus the number of first order systems while N_C refers to one set of complex conjugate poles and thus the number of second order systems.

$$\begin{aligned} H(s) &= \sum_{i=1}^{N_R} \frac{k_i}{s-p_i} + \sum_{i=1}^{N_C} \left(\frac{k_i}{s-p_i} + \frac{k_i}{s-p_i^*} \right) \\ &= \sum_{i=1}^{N_R} \frac{k_i}{s-p_i} + \sum_{i=1}^{N_C} \frac{r_i(s)}{s^2 + 2\zeta_i w_i + w_i^2} \end{aligned} \tag{2.17}$$

Where

$$w_i = |p_i| \tag{2.18}$$

$$\zeta_i = \frac{-p_{i,r}}{|p_i|} \tag{2.19}$$

A second order system is underdamped for $0 \leq \zeta < 1$. Designers do not normally worry about loops with $\zeta > 0.7$ since a damping factor of 0.7 indicates that a loop has a phase margin of 65 degrees. A pole with $\zeta < 0.7$ is thus in our region of interest for stability checking purposes. w_i identifies the natural frequency of this potentially unstable pole and hence helps us identify whether the pole will degrade circuit performance within our frequency range of interest or not. In general it can be said that if any pole in our system satisfies the following two conditions, we have detected a potential instability in the circuit.

$$\begin{aligned} w_i &\leq 2\pi f_{max} \qquad (2.20) \\ \zeta_i &< \zeta_{th}\,(\text{approx. } 0.7) \qquad (2.21) \end{aligned}$$

Thus identifying the poles of our system transfer function $H(s)$ will enable us to in turn detect the existence of unstable loops in our circuit.

2. Relationship between poles and eigenvalues

We note from equation 2.13 that the input and output nodes selected to generate $H(s)$ does not effect our system poles in any way. Since our pole information is determined by the denominator term $(G + sC)^{-1}$, we shall proceed to show that we can obtain the poles to our transfer function $H(s)$ by solving the generalised eigenvalue problem as given in equation 2.22

$$Gv_j = -\lambda_j C v_j \qquad (2.22)$$

Assuming that the pencil (G,C) is diagonalisable and C is nonsingular, we can obtain W,Λ such that

$$\begin{aligned} W^H G V &= -\Lambda \qquad (2.23) \\ W^H C V &= I \end{aligned}$$

where

$$\Lambda = diag\,(\lambda_1, \lambda_2, ..., \lambda_n) \qquad (2.24)$$

Now our denominator term $(G + sC)^{-1}$ of our system transfer function can be expressed as

$$\begin{aligned}(G+sC)^{-1} &= \left(W^{-H}(-\Lambda+sI)V^{-1}\right)^{-1} \\ &= V(-\Lambda+sI)^{-1}W^{H}\end{aligned} \tag{2.25}$$

Setting $r^* = W^H B$ and $l = V^H L$, the system transfer function $H(s)$ can be expressed as

$$H(s) = \sum_{j=1}^{n} \frac{l_j^H r_j^*}{s-\lambda_j} \tag{2.26}$$

Thus the poles of the transfer function $H(s)$ are equal to the eigenvalues of the matrix pencil (G,C).

3. Residue computation and dominant pole extraction

Note that to compute the impedances and extract dominant poles from our second order approximations, we need both the pole values and the residue information. While the poles of the circuit remain the same over the entire circuit, the residue information needs to be computed on a per-node basis.

The transfer function given in equation 2.26 allows us to efficiently compute residue information as well. However, when performing eigen value decomposition via QZ as a black box routine, we are only returned the V and Λ matrices.

Expressing W^H in terms of V and Λ from equation 2.23 and plugging into $r^* = W^H B$, we get

$$r^* = -V^{-1}G^{-1}\Lambda B \tag{2.27}$$

Since Λ is a diagonal matrix, we can further express the j_{th} entry of the column vector r^* in equation 2.27 as $r_j^* = r_j \times \lambda_j$ where r is defined by

$$\begin{aligned} r &= -V^{-1}G^{-1}B \\ &= -\left(GV\right)^{-1}B \end{aligned} \tag{2.28}$$

The reason we have expressed $V^{-1}G^{-1}$ as $\left(GV\right)^{-1}$ in equation 2.28 is that this information stays the same over all the nodes of the circuits. Thus we can precompute LU factors of the matrix (GV) and simply perform linear solves at each node giving us a big performance improvement.

Thus 2.26 can be expressed as

$$H\left(s\right) = \sum_{j=1}^{n} \frac{k_j}{s-\lambda_j} \tag{2.29}$$

where

$$\begin{aligned} k_i &= r_i \times \lambda i \times l_i \\ r &= -\left(GV\right)^{-1}B \\ l &= V^H L \end{aligned}$$

E. The loop finder algorithm

The basic idea behind the loop finder algorithm has already been described in the previous sections.

The complete loop finder algorithm as described in the reference work by Peter Fang, Rod Burt and Ning Dong[4] describes a 5 step approach to perform loop finder analysis as given below :

1. Compute all the poles for the linearized circuit and selectively pick out potentially unstable poles (poles with damping factor within a bound of 0.7 and

within a designer specified frequency bound).

2. Compute residues of impedance transfer function for all the poles of interest over all nodes.

3. Ignore second order systems with very low DC impedance.

4. Determine dominant complex poles for each node transfer function.

5. Report identified loops. Nodes that share the same dominant complex pole can be said to form a loop.

As described earlier, the first two steps deals with the extraction of second order systems of interest from the system as a whole. It is also computationally most intensive.

The loop finder algorithm as described in [4] uses QZ decomposition of the matrix pencil $(G^{-1}C, I)$. In the analysis given in the previous section and the implementation of the loop finder algorithm that we shall use as a benchmark, we further improve upon this by avoiding the computation of $G^{-1}C$ and using the matrix pencil (G, C) instead. We further precompute the LU factors of the matrix (GV) and reuse them for every node. Thus the benchmark algorithm we use in chapter V is already heavily optimised from an algorithmic perspective.

The loop finder algorithm has been used in Texas Instruments' in house spice simulator to successfully identify instability problems but does not scale to large problem sizes due to reasons that will be described in the next section. The focus of the rest of the thesis will be to successfully extend the loop finder methodology to large circuits by utilizing various pole discovery and model order reduction techniques.

CHAPTER III

POLE DISCOVERY

As described earlier the pole discovery stage is just one part of the overall loop finder algorithm. However, two important things to note here are (1) it is an independent step (2) the pole information remains the same over all nodes and hence needs to be calculated only once.

The pole discovery stage effectively involves solving a subset of the generalised eigen value problem :

$$(G + \lambda C)\, x \;=\; 0 \tag{3.1}$$

The only difference from a complete generalised eigen value solution is that we are only interested in a certain subset of λ. That is those within a certain frequency bound and with a damping factor of less than 0.7.

A. Background

In the original loop finder methodology as described in [4] the QZ method was run over the entire circuit to discover all the poles for a given circuit. The poles which did not lie in the region of interest were subsequently rejected. This method proves to be rather wasteful since the number of poles we are actually interested in is actually a very small subset of the total size of the system. Also QZ method[5] has $O(N^3)$ time complexity and solves a dense matrix problem even though the matrices may be sparse thus imposing a heavy memory demand as well. So while a QZ based implementation works well over smaller circuit sizes and gives us accurate information about potentially unstable loops, it cannot be extended to very large circuit sizes.

An alternative method would be to only explore the region of interest of the s-plane and extract only the relevant pole information. This works best when the number of poles in our region of interest is a very small subset of the total size of the system. This is especially true for our target application where the individual macros that make up the large analog circuit have already been checked for stability. Thus, even if we are analyzing a 20,000 node system, the maximum number of unstable poles that we would have inadvertently added to the system would be closer to 10 or 20. Once we are guaranteed 100% coverage within the region of interest in the s-plane, we need not run iterations to compute the rest of the eigen values thus saving a lot of computational cycles.

A Lanczos or Arnoldi based algorithm centered around multiple expansion points is a logical choice to perform a region specific eigen value search. While several variations based around this basic theme exists, the rational Krylov algorithm as described by Axel Ruhe in [11, 12, 13] proves itself to be a good base algorithm towards this purpose.

B. The rational Krylov algorithm

The rational Krylov algorithm for eigen value computation and model reduction as described in [11, 12, 13] is an extention of the Lanczos or Arnoldi eigenvalue algorithm where several shifts(matrix factorizations) are performed in one run. It can be used to compute a selected set of eigenvalues to a nonsymmetric pencil

$$(A - \lambda B)\, x \;=\; 0 \tag{3.2}$$

The rational Krylov algorithm computes an orthogonal basis and a small Hessenberg pencil. The size of the Hessenberg pencil is very small when compared to the size of the system and is easily dealt with using standard eigen value decomposition methods involving similarity transformations. The eigensolution of the Hessenberg pencil in turn gives Ritz approximations to the solution of the original pencil.

The salient points of this algorithm that lend itself particularly well to our application are given below :

- Within a specified area of interest, the algorithm converges much faster than running Arnoldi iterations around a fixed point. This is because the closer the expansion point is to our pole, the faster that pole converges. Also running Arnoldi iterations around a fixed point has the disadvantage of rapidly increasing model size without capturing any new information. In fact, generating larger and larger models by running Arnoldi iterations around a fixed point may cause us to start converging upon bogus and/or duplicate eigen values and affect the accuracy of the poles we have already discovered.

- As described in [13], we can get a real Hessenberg pencil even in the case of complex shifts. This keeps the amount of complex arithmetic that needs to be performed to the bare minimum.

- As long as we can keep the size of the Hessenberg pencil limited, the eigen value approximations are easily obtained using the QZ method.

- The norm of the residual of an eigen value approximation is easily obtained from our Hessenberg pencil and serves as a good indicator of the accuracy of the eigenvalue with respect to the original system. It can be used both as a metric of convergence and to pick a good expansion point for subsequent shifts. Also, computing this norm value proves to be computationally less expensive than other methods based on a shift and invert Arnoldi methodology.

The detailed algorithm and how it works can be found in [13]. For sake of simplicity, we shall only describe the rational Krylov algorithm for complex expansion points here as it is the base algorithm used in this work. Since we are not interested in real poles, there is no reason for us to pick real expansion points.

1. Rational Krylov iteration for complex expansion points

The original rational Krylov algorithm described in [11] works for both real and complex shifts as long as the matrix pencil (A,B) is real. However Ruhe later extends his work in [13] whereby we do all the factorisation and solution operations on a shifted matrix $(A - \mu B)$ in complex arithmetic but subsequently split the resulting vector r into real and imaginary parts and deal with them seperately. The advantage of this of course is not having to perform complex orthogonolization or eigen decomposition on a complex matrix pencil.

The algorithm for complex shifts is described in Algorithm 1 where (A,B) is the matrix pencil we wish to solve.

(H,K) together make up a Hessenberg pencil whose eigen values can be easily computed. We will discuss the computation of the approximate eigen solution and its residual as well as the convergence criterion in more detail.

2. Approximate eigen solution and computation of residual

We find the approximate Eigen solution θ by solving the problem,

$$(K_{j,j} - \theta H_{j,j})\, s \;=\; 0 \tag{3.3}$$

Note that the subscript j refers to the number of columns in H or K and not step j. Recollect that for the rational Krylov iteration with complex expansion points, we add two columns to the Hessenberg pencil in place of one.

The approximative Eigen value is found by solving the generalised eigen value

Algorithm 1 RationalKrylovAlgorithm(A, B) [for complex shifts]

```
1: Choose starting vector $V(1)$
2: for j = 1,3,5.... until convergence do
3:     Choose shift $\mu_j$ and starting combination $r \leftarrow V_j t_j$
4:     Operate $r \leftarrow (A - \mu_j B)^{-1} B r$
5:     $r_1 \leftarrow real(r)$; $r_2 \leftarrow imag(r)$
6:     Orthogonalize $r_1 \leftarrow r_1 - V_j h_j$ where $h_j \leftarrow V_j^T r_1$
7:     Get new vector $v_{j+1} \leftarrow r_1 / h_{j+1,j}$ where $h_{j+1,j} \leftarrow ||r_1||$
8:     Orthogonalize $r_2 \leftarrow r_2 - V_{j+1} h_{j+1}$ where $h_{j+1} \leftarrow V_{j+1}^T r_2$
9:     Get new vector $v_{j+2} \leftarrow r_2 / h_{j+2,j+1}$ where $h_{j+2,j+1} \leftarrow ||r_2||$
10:    $M(j, j) \leftarrow real(\mu_j)$
11:    $M(j+1, j+1) \leftarrow real(\mu_j)$
12:    $M(j+1, j) \leftarrow -imag(\mu_j)$
13:    $M(j, j+1) \leftarrow imag(\mu_j)$
14:    Compute $K_{j+1,j+1} \leftarrow H_{j+1,j+1} M_{j+1,j+1} + T_{j+1,j+1}$
15:    Compute approximative eigen solution for the matrix pencil $(H, K)$
16:    Test for convergence and pick new $\mu_j$ if required
17: end for
```

problem using the QZ method. Note that the QZ method does not prove to be computationally expensive in this case as long as the size of the pencil (H,K) stays in control.

For a given solution (θ, s), we can take the vector

$$x \;=\; V_{j+1} H_{j+1,j} s \tag{3.4}$$

as the ritz approximation of the eigen vector in the original system.

It can also be shown that

$$\beta_{j,i} \;=\; (\mu_j - \theta_i h_{j+1,j})\, s_{j,i} \tag{3.5}$$

estimates the norm of the residual for the ith eigenvalue approximation θ_i This residual can be used to test for convergence and to select the next shift as mentioned below.

3. Check for convergence and selection of next shift

We use the previously computed norm of the residual $\beta_{j,i}$ as a convergence check. We flag an eigen value as converged when it is smaller than a preset value *tolconv* We pick a value of 0.5e-8 for *tolconv* as described in [13]. The approximative eigenvalue θ_i whose norm of residual at step j ($\beta_{j,i}$) is the minimum of all the eigenvalues with residuals above a value *tolshift* is chosen as the next shift. We use a value of 0.5e-4 for *tolshift* as described in [13]. The restriction of choosing a value greater than *tolshift* is mainly to avoid nearly singular shifted matrices. We keep the same shift for at most 5 steps in most cases. However, this number may increase dramatically when no eigenvalue with residual greater than tolshift is detected. This is a very real possibility since we pick only complex expansion points.

C. Shortcomings of rational Krylov algorithm for region based pole search

The rational Krylov algorithm proves to be a good choice for a base algorithm for reasons described in the previous section. However, when left to itself, rational Krylov algorithm discovers both real and complex poles and does not restrict itself within a specific area of our s-plane.

To remedy this problem and to avoid making unnecessary shifts, we introduced the concept of making complex shifts only in our application but even this methodology has its shortcomings as shall be described.

The norm of the residual computed from the Ritz approximation is mostly a pretty good indicator of which of the eigen values computed from the reduced Hessenberg pencil are a good approximation of the overall system. However, this is not true in all cases. Most importantly the residual value cannot be trusted in the following two cases

1. Large Hessenberg pencils.
2. Very large area of interest when the poles are randomly distributed and widely spaced.

An additional case to be considered is when we pick the starting expansion point too close to an eigen value thus resulting in a nearly singular matrix. Which means that even if we are to take care of the two main shortcomings we have mentioned above, we still need some way of verifying our results since there is no safe value for our first shift that will work for any arbitrary matrix pencil.

One way to solve the Large Hessenberg pencil problem would be to prune the model from time to time keeping only our relevant Eigen information. This has been discussed in [11] and can keep the size of our Hessenberg pencil down but will not solve the instability faced over large areas. On the other hand, a restart based algorithm will solve the issue of a large area but by itself may not be able to solve the accidental instability we may face from choosing a wrong expansion point in the first place.

Lastly, the rational Krylov algorithm by itself does not guarantee complete convergence. That means that there is no guarantee that all the poles in the region of interest will be discovered.

Having said that, the merits of the rational Krylov algorithm cannot be discounted and it is clear that a more carefully designed top level algorithm is required to solve the problems mentioned here.

D. Top level pole discovery algorithm

The top level algorithm that we developed to try and make up for the problems mentioned above solves both these problems by the following three means :

1. Control overall size of Hessenberg matrix during the search.
2. Control size of region of search.
3. Check accuracy of results obtained from a search by running more searches.

Clearly, from the above list of requirements, a partitioning strategy of some nature is required. Before we focus on the top level algorithm, let us focus on modifying the rational Krylov algorithm so that it operates within a limited region of interest and the size of the Hessenberg pencil remains limited. The partitionining strategy will then be described under top level pole discovery algorithm.

1. Bounded region search using rational Krylov algorithm

The modifications made to the rational Krylov algorithm to perform a region based search are relatively straightforward. It involves introducing the concept of a bounding box and not making a shift that is not within the bounding box. We prefer to use a square box as it is most easily subdivided as will be required by our top level algorithm. Also, the rate of convergence of a given pole is determined by distance from the closest expansion point and hence a circular or square region where we start the algorithm from the center is likely to give us the best results.

The modified rational Krylov algorithm is given in Algorithm 2. As you can note only the convergence tests have been improved and along with the pole values we are returning a convergence flag REGION_CONVERGED.

As can be seen in algorithm 2, the algorithm can exit in two ways. Depending upon whether it managed to converge on all the poles before exiting or not, it may set a status flag indicating whether the region has converged or not.

1. The first exit criterion and especially the condition "YET_TO _CONVERGE is not set" is in effect the convergence criterion for the algorithm as a whole for a successful exit. The flag is set only when there are some eigen values with residuals between tolconv and tolshift during a particular iteration. If a residual is above tolshift, we will choose it as a shift and thus converge on the corresponding Eigen value within a finite number of iterations. Once a residual falls below *tolconv* we know it has converged. Thus at any given time, we know if there are any Eigen values that are about to converge if we iterate a few more times simply by examining the flag "YET_TO_CONVERGE". The other condition namely "number of iterations > miniters_in_current_bbox" exists only to ensure that we build a sufficiently large Hessenberg pencil to check whether

Algorithm 2 RationalKrylovAlgorithmInBbox($A, B, bbox$)

```
1: Choose starting vector $V(1)$
2: for j = 1,3,5.... until convergence do
3:   Choose shift $\mu_j$ and starting combination $r \leftarrow V_j t_j$
4:   Update (H,K) {As given in algorithm 1}
5:   Compute approximative eigen solution for the matrix pencil $(H, K)$
6:   for all Eigen values in the pencil (H,K) do
7:     if Eigen Value does not belong to bounding bbox then
8:       Skip to next eigen value
9:     else if Eigen Value has already been marked as converged then
10:      Skip to next eigen value
11:    else if Eigen Value has residual less than tolconv then
12:      Mark Eigen Value as converged
13:    else if Eigen Value has residual greater than tolshift then
14:      Keep track of lowest eigen value which will be used for next shift
15:    else if tolconv < residual of Eigen Value < tolshift then
16:      Set Flag YET_TO_CONVERGE
17:    end if
18:  end for
19:  if number of iterations > miniters_in_current_bbox & ! YET_TO_CONVERGE
     then
20:    Set Flag REGION_CONVERGED
21:    EXIT()
22:  else if number of iterations > maxiters_in_current_bbox then
23:    Set Flag REGION_NOT_CONVERGED
24:    EXIT()
25:  end if
26: end for
```

there is any Eigen value of interest in the bounding box. The minimum number of iterations required to successfully say that the region contains no Eigen values increases with increase in circuit size. While this value can be statically set over all the bounding boxes, we shall discuss a dynamic approach towards predicting this value under the performance optimization section.

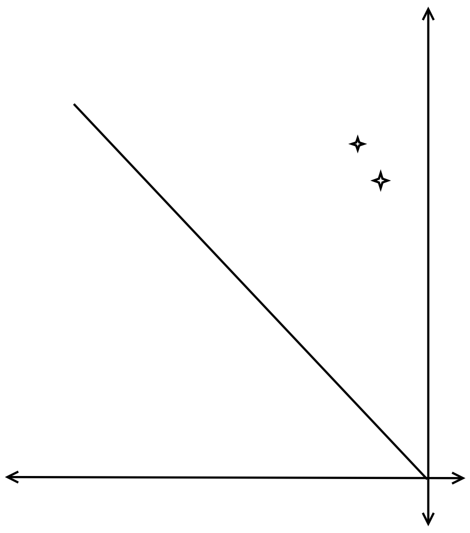
2. The second exit criterion "number of iterations > maxiters_in_ current_bbox" puts a cap on the size of our Hessenberg pencil by controlling the maximum number of iterations. If we have not managed to converge on all the poles within a finite number of iterations, we will conclude that we are not going to be able to correctly converge on the Eigen Values of interest or that doing so will be computationally very intensive and further action is required. Thus we explicitly flag the region as not being converged before exiting the algorithm. It is of note that we consider subdividing a region and running the rational Krylov algorithm within each one to be preferable to increasing the size of the Hessenberg pencil within a given partition.

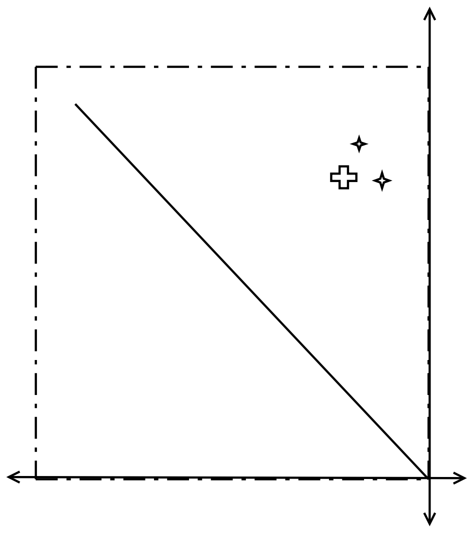
2. Top level control algorithm

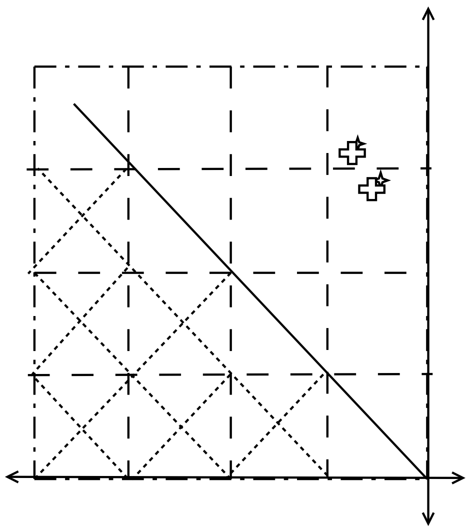
The top level pole discovery algorithm is easily described as given in algorithm 3. However for a detailed understanding of how it works, it is easier to examine an example such as that explained using Fig. 8. A star refers to real system poles and a cross to pole approximations obtained from the algorithm during any given step. As we can see, by successively repartitioning the s-plane, we converge closer and closer to our original system poles. In the next few paragraphs we shall proceed to explain how the algorithm works using the visual example given in Fig. 8 as well as to mention any caveats and implementation details.

Algorithm 3 PoleDiscovery($A, B, bbox, damptol, parent\_pole\_list$)

```
1: Get $pole\_list$ from parent partition as $parent\_pole\_list$
2: Run RationalKrylovAlgorithmInBbox($A, B, bbox$)
3: Store pole list obtained from Rational Krylov Algorithm as $pole\_list$
4: clear flag SUBDIVIDE_REGION
5: **if** Isset REGION_NOT_CONVERGED **then**
6:    clear $pole\_list$
7:    set flag SUBDIVIDE_REGION
8: **else if** Isset REGION_CONVERGED **then**
9:    **if** $pole\_list \not\subseteq parent\_pole\_list$ **then**
10:       set flag SUBDIVIDE_REGION
11:    **else**
12:       Add $pole\_list$ to global pole list
13:    **end if**
14: **end if**
15: **if** Isset SUBDIVIDE_REGION **then**
16:    Compute new number of partitions
17:    Compute bounding boxes for all partitions
18:    **for all** bounding boxes in region of interest as $NewBbox$ **do**
19:       PoleDiscovery(A,B,NewBbox,damptol,pole_list)
20:    **end for**
21: **end if**
```


(a) Damping tolerance line marking region of interest containing 2 poles in it


(b) Starting bbox and first pole approximation. Note that edge length of bbox has to be equal to frequency of interest


(c) Partitioning of starting bbox; rejection of bounding boxes not in region of interest; All but two regions found to have no poles

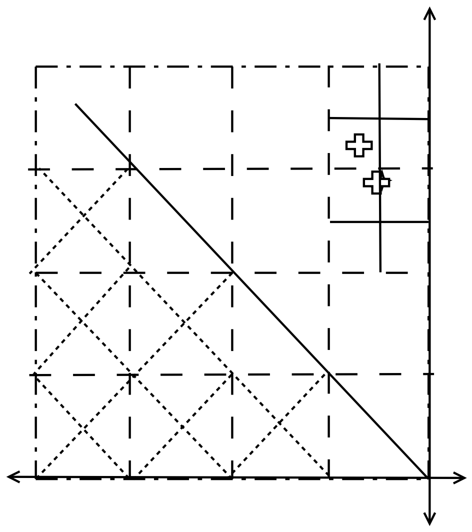

(d) Further partitioning for verification of pole values found in previous step; Poles found to be within 1e-4 relative accuracy of the approximation in the previous step and are accepted

Fig. 8. Sample run of top level pole discovery algorithm

The top level algorithm searches for poles by using a recursive partitioning strategy as demonstrated in Fig. 8. Fig. 8(a) demonstrates an actual s-plane showing that we have 2 poles in our region of interest. The region of interest is in turn defined by a subset of the region above the X axis demarcated by the Y-axis and a line which represents the locus of all points with a specified damping factor (mostly 0.7 for critical damping). This damping factor represents the damping tolerance or maximum damping factor of the poles that we are interested in.

We first start with a relatively large region of interest and run the modified rational Krylov algorithm over the large bounding box as shown in Fig. 8(b). The edge length of the box has to be equal to the frequency of interest. If our algorithm exits after setting the REGION_NOT_CONVERGED flag as is expected, then we reject any pole information that might have been extracted from the rational Krylov algorithm and proceed to the next step. If our algorithm exits after setting the REGION_CONVERGED flag, we keep the pole information (in the figure only one pole approximating two poles) and proceed to the next step.

The next step is to subdivide our bounding box into smaller boxes on which we may run the same algorithm on as given in Fig. 8(c). Note that there might be cases in which the algorithm exits after setting REGION_CONVERGED yet no Eigen values have been discovered. While this means that our algorithm is malfunctioning if it happens in the bounding box in Fig. 8(b), if it happens in the smaller boxes shown in Fig. 8(c) it means that there are no Eigen values within those bounding boxes.

If on the other hand, we do find some poles within the region as is the case for 2 of the boxes in Fig. 8(c), we will compare it to the poles we obtained from the parent bounding box. In this particular case, we find that they are not the same pole which is why we proceed to the next step and subdivide the two boxes again in figure Fig. 8(d).

In figure Fig. 8(d), the 2 boxes containing poles in Fig. 8(c) have been subdivided into 8 smaller boxes (each box has been divided into 2x2). In this step all the boxes but 2 exit with REGION_CONVERGED but return no Eigen Value. For the two boxes that do return Eigen Values, we compare the obtained Eigen Values against the Eigen values from Fig. 8(c). Finding them to be one and the same, we decide that our poles have converged. If they had not been one and the same, we would have to subdivide the corresponding bounding box again.

This comparison with the parent bounding box and ensuring that the Eigen Values are one and the same, are to take into account any errors caused by the algorithmic instabilities mentioned earlier. As long as we obtain the same Eigen Values from the subregions as we do from the parent bounding box, we can conclude the Eigen Values are in fact correct. Also, dividing a box into smaller boxes ensures greater coverage and we make sure that the parent bounding box was not heavily biased by a particular set of poles.

Note that when we say "one and the same", in reality the eigen values are not really exactly the same but may differ by relative error tolerance *errortol* which can be specified by the user. The relative error tolerance or the relative accuracy can be described as :

$$\text{relative accuracy} = \frac{\text{distance between poles under test}}{\text{distance of one pole from origin}} \tag{3.6}$$

A relative error of 0.01 thus roughly corresponds to 2 decimal digits of accuracy. As a result, as we can see the level of accuracy of the actual pole values are in our control.

Please note that the relative error metric is important in more ways than just specifying whether two poles will appear to be one and the same or not. It is also used in determining whether we have already examined a pole in the parent bounding box. As a result, a high value of errortol (which is the parameter by which we specify the maximum allowable relative error tolerance) may also potentially make the algorithm unstable and/or report a lot of bogus poles which are not actually present in the circuit. The algorithm is always unstable for errotol greater than 0.01. Depending upon the nature of the circuit, the algorithm might report bogus poles even at an errortol of 0.01. Hence, we may wish to reduce our errotol even lower. An errortol of 1e-4 has proven to be more than sufficient for all our test circuits so far.

Also, the number of partitions at each level are chosen in such a way as to maximise the coverage in the minimum number of runs. Since the poles are normally widely distributed we first start off by dividing a bounding box into a large number of regions perhaps a 8x8 matrix. This way we can reject large regions that do not contain any poles early on. But once we start discovering poles, smaller number of partitions will suffice so we can break it down into 4x4 and eventually 2x2 regions simply for verifying the already obtained pole values.

One special case that we skipped over while explaining Fig. 8 was a case in which we will exit with REGION_CONVERGED but no Eigen values in Fig. 8(b). Since the starting bounding box pretty much refers to the entire complex s-plane, it is unlikely that there will be no poles of interest. Such a situation normally means that our algorithm has malfunctioned because our miniters_in_current_bbox is not sufficiently high to start predicting which Eigen values are about to converge.

It has been observed that while a good value for miniters_in_current_bbox grows with the size of the circuit, it does depend upon the nature of the circuit and the pole distribution as well. Hence we have had to adopt a dynamic strategy for determining a good starting value for miniters_in_current_bbox. This strategy will be described in the “Performance optimization of top level algorithm” section.

E. Performance optimization of top level algorithm

The rational Krylov algorithm by itself offers very little scope for optimization. However, the rational Krylov algorithm does accept various parameters via which the runtime of one instance of the rational Krylov algorithm (and thus the cost of examining one partition) can be kept under control. Our top level control algorithm that is responsible for calling the rational Krylov algorithm and passing the necessary parameters to it thus provides us significant scope for optimization.

Please note that in many cases, optimizing for performance may come at the cost of accuracy. We examine a few methods to optimize performance that do not sacrifice accuracy or at least provide the designer with a mechanism to specify what kind of performance-accuracy tradeoff they wish for.

The main ways of optimizing performance that we shall examine are :

1. Restrict search to region of interest

Since we run rational Krylov algorithm on each of our bounding boxes, reducing the number of bounding boxes we examine reduces the number of instances of the rational Krylov algorithm we are forced to create. A very simple method to do so is examine only those bounding boxes which are of interest to us.

To determine whether or not a bounding box is of interest, measures can be taken to ensure that at least a part of the bounding box we are about to examine falls within our region of interest on the s-plane. This is easily done simply by examining the corners of our bounding box.

If the damping factor of our Upper Right corner of the bounding box is greater than the damping tolerance, it means NO point in the bounding box is within our region of interest. The entire partition can thus be safely ignored and need not be analyzed.

Similarly, if the distance of the lower right corner from the origin (the magnitude) is above our maximum frequency of interest, then it again means no point in the bounding box is within the region of interest.

A user programmable minimum frequency bound can also be similarly provided by examining the upper left corner.

2. Control miniters_in_current_bbox

The value of miniters_in_current_bbox plays a significant role in making sure that we explore all regions of interest fully. Too low a value of this parameter in any given partition and we may exit a partition before we have run sufficient number of rational Krylov algorithm iterations to begin detecting poles in that region. Thus in effect, we may conclude that a region is empty while in fact it does contain poles.

On the other hand, keeping the value of this parameter uniformly high over all partitions leads to a significant waste of computational cycles. Scaling this parameter down as our partitions become smaller and smaller would help us save significant number of computational cycles.

Predetermining a safe value of this parameter however is not straightforward. While it does increase with size of the system and with the size of the partition being examined, these two trends are not uniform over all circuit types. The nature of the circuit and the pole distribution has significant effect on how soon or fast we can scale down the value of this parameter as we make smaller and smaller partitions.

Fortunately, the algorithm itself provides us with runtime information that we can use to accurately predict this parameter. Since the minimum number of iterations required to detect if a region contains a pole is expected to scale down with the size of the partition, we try and use the iteration at which a pole is first detected in a partition as the number of iterations information in our subpartitions. In reality, we observe that the iteration that a pole is first detected at in a subpartition is always less than or within a certain error bound of the iteration at which the first pole is detected in the parent partition. In other words, if we detected the existence of a pole in the nth iteration, then we are going to detect the existence of a pole in the n+error_bound iteration for its subpartitions.

Therefore we define miniters_in_current_bbox as :

$$\begin{aligned} \text{miniters} &= \text{pole detected at iteration (parent partition)} \\ &\quad +2 \times \text{sensitivity} \end{aligned} \tag{3.7}$$

where sensitivity is a user defined value that can vary anywhere between 1 and 5. A sensitivity value of 3 works well in most cases.

Note that the value of miniters for a subpartition can be easily determined from the pole_detected_at_iteration value of its parent partition. That leaves only the value of miniters for the first bbox (Fig. 8(b)) in question since it has no parent. Since this partition is too big to report pole information accurately anyway, we instead use this algorithm only to estimate the value of miniters for its subpartitions. This is easily done by assuming that the top level algorithm always contain poles and running the algorithm all the way upto maxiters_in_any_bbox till a pole is detected. Once a pole is detected, we simply record the iteration at which the pole was detected and exit the rational Krylov algorithm.

3. Control maxiters_in_current_bbox

For smaller circuits and smaller partitions, we can expect to detect poles within a region faster than in larger regions or larger circuits. If we keep iterating and discovering new poles, then it probably means that our algorithm has become unstable due to the factors mentioned earlier on in this thesis.

Since the value of maxiters_in_current_bbox defines the exit condition in the case we do run into an instability, scaling this value depending upon the circuit and/or partition size can lead to significant savings in computational time. But as with miniters_in_current_bbox, this parameter once again depends on the nature of the circuit and the pole distribution.

The safest way to vary this parameter is to vary it based upon the miniters_in_current_bbox as shown below :

$$\begin{aligned} \text{maxiters} &= \min(\ \text{miniters} \times 2, \text{miniters} + 20 \\ &\quad , \text{maxiters\_in\_any\_bbox}) \end{aligned} \tag{3.8}$$

The miniters*2 condition takes care of small partitions or small circuits (where the miniters has been determined to be less than 20), the second condition miniters + 20 takes care of larger partitions and/or circuits. The last condition maxiters_in_any_bbox sets a global cap on the highest allowable value of maxiters. The last condition keeps the size of the model we generate for QZ decomposition in check. A value of 50 for this parameter means we will never run QZ on a matrix of size > 100.

4. User controlled speed vs accuracy tradeoffs

Since the time taken for pole discovery depends significantly upon the size of the region to be explored, letting the end user(the designer) control the maximum and minimum frequencies as well as the damping factor could significantly speed up our algorithm by limiting the number and size of partitions that need to be examined.

By changing the values of sensitivity and maxiters_in_any_bbox too, the designer can try and reduce the runtime though in this case there may be accuracy tradeoffs. Please note that changing the value of maxiters_in_any_bbox can have a counterintuitive effect on runtime. While the smaller the value of this parameter the better, if we keep concluding that the algorithm is unstable for a particular bbox and keep dividing the regions, we may end up examining more regions than we originally anticipated and thus in fact increase the execution time. A value of 50 has been found to be safe for most applications so far.

Lastly, other accuracy metrices such as “errortol” can be also changed but yet again the effect of these parameters may be counterintuitive. Setting a high value of errortol let’s say 0.01 might actually make us explore more regions as we converge on bogus poles (poles that do not really exist in the real system). A value of 1e-4 has been found to be safe though higher values of upto 0.01 do work for most circuit types.

Of all the parameters, the ones that will be safest for the designer to change will be the minimum and maximum frequencies and the damping tolerance. Changing the sensitivity must be undertaken with utmost care and conservative values should be used to obtain the pole information before trying smaller values. The rest of the values should probably not be touched unless trying extremely large and/or complicated circuits for which the current bounds prove to be insufficient.

CHAPTER IV

RESIDUE COMPUTATION

As described previously, the residue computation stage is the second step of the overall loop finder algorithm.

In a regular pole-zero analysis, only one transfer function needs to be computed. However for the loop finder algorithm, we need the transfer functions corresponding to a large number of nodes. Thus unlike the pole information which remains constant throughout the circuit, the residue information needs to be computed on a per-node basis. As our circuit size grows, the residue computation step is thus also likely to be the most costly step.

The main improvements that can be made in the top level residue computation algorithm are as follows :

1. Since the cost of residue computation of the poles of interest at one node will be multiplied by the number of nodes we want to analyze, reducing the per-node residue computation cost will bring down the cost of residue computation as a whole. Various opportunities to do so exist as shall be described in the Background section.

2. Designers may often wish to simply identify the existence and/or locality of a loop or analyze only a certain portion of the entire circuit. In both these cases, residue information needs to be generated for only a certain subset of the nodes in the circuit. Thus our top level residue computation algorithm should be able to create second order approximations and extract residue information on a per-node basis.

A. Background

In the QZ based loop finder method[4], the eigen vector matrix is obtained as part of the pole discovery stage. This eigen vector matix in turn can be used to project the circuit response onto the eigen space. Thus with our pole values and eigen vector matrix, we can generate the residue information for all the nodes over all poles. However as mentioned earlier we need only a very small subset of this information and thus we simply pick out the residues that correspond to our poles of interest[4].

Two points are worthy of note here.

1. Due to our new pole discovery methodology, we no longer have a unified eigen vector matrix as we previously did and so this direct projection method is no longer applicable.

2. The eigen vector matrix is a large dense matrix and to compute the actual residues we need to perform a matrix solve using this large dense matrix. Thus any method exploiting the inherent sparseness of our circuit matrices can possibly be made to run faster than the QZ based residue computation strategy.

Given that we are already aware of the poles in our circuit, and that we wish for an algorithm which can generate residue information on a per-node basis, generating reduced order models for the node transfer function is the logical next choice. Also, given the fact that we have only a very limited number of poles of interest, our reduced order model too will only contain information corresponding to the second order systems of interest.

B. Model order reduction

With the rapid decrease of feature size and the corresponding increase in the effect of parasatics, the complexity of our integrated circuits has been growing at an increasing rate. The effect of RLC interconnects too have become more dominant than ever before. Various model order reduction techniques have been developed over time to aid with generating reduced macromodels to deal with this increasing complexity. While these macromodels were originally developed for use in the simulation of large electrical circuits, the theory and algorithms developed have been used for other application areas as well.

In general, for a time invariant electrical circuit that can be described by the system of linear equations mentioned in equation 2.12, we can generate a reduced model given by the following system of equations :

$$\begin{aligned} \hat{C}\hat{x}^{\cdot}(t) &= -\hat{G}\hat{x}(t) + \hat{B}u(t) \\ \hat{y}(t) &= \hat{L}^{T}\hat{x}(t) \end{aligned} \tag{4.1}$$

where $\hat{G}$, $\hat{C}$, $\hat{B}$, $\hat{L}$, $\hat{x}$, $\hat{y}$, are of rank $k \ll n$.

It is important to note that any model order reduction technique by itself produces a a low order rational approximation for a given entry in our overall system transfer function. Whether we use explicit moment matching methods or Krylov subspace methods, they guarantee the preservation of moments but not of the actual eigen values. In our case, to be able to map our reduced model circuit responses to our eigen values of interest, our reduced order transfer function should contain our poles of interest as well. Clearly our reduced model must be accurate not just around a fixed point(say DC), but over a whole range of frequencies corresponding to our pole values of interest. Reduction algorithms that address this concern include various complex frequency hopping algorithms[14] and the rational Krylov algorithm when applied to model order reduction[15].

Since our linearized network is effectively a multiport RLC network, we would like to preserve the passivity of this system as in [16]. As in PRIMA, the reduced order system can be easily obtained using congruence transformations of our original system matrices as given in equation 4.2.

$$\begin{aligned} \hat{G} &= V'\,G\,V \\ \hat{C} &= V'\,C\,V \\ \hat{B} &= V'\,B \\ \hat{L} &= V'\,L \end{aligned} \tag{4.2}$$

Clearly what we require is a PRIMA like methodology that is capable of satisfying accuracy requirements at more than one expansion point. This need has already been addressed in the block rational Arnoldi algorithm for multipoint passive model order reduction for multiport RLC networks[17]. We shall first examine the general concept behind multipoint rational approximations.

1. Multipoint rational approximation

In a multipoint rational approximation, we are given m distinct points $s_1, s_2, ...s_m$ and m integers $n_1, n_2, ...n_m$ and we are asked to find the driving point impedance such that

$$\frac{d^{k-1}\hat{Z}}{ds^{k-1}}(s) = \frac{d^{k-1}\hat{Z}}{ds^{k-1}}(s) \tag{4.3}$$

where

$$\begin{aligned} 1 &\leq k \leq n_i \\ 1 &\leq i \leq m \end{aligned} \tag{4.4}$$

In other words, we wish to generate a reduced order transfer function

$$\hat{Z}(s) = \frac{b_{q-1}s^{q-1} + ... + b_1 s + b_0}{a_q s^q + ... + a_1 s + 1} \tag{4.5}$$

such that for every interpolation point $s_i, i = 1...m$, the coefficients $a_1...a_q$ and $b_0...b_{q-1}$ are chosen in such a way that the moments around the interpolation points of the approximate transfer function $\hat{Z}(s)$ match the moments of the transfer function $Z(s)$.

That is for each point s_i, there is a Krylov subspace $\mathcal{K}_{n_i}$ spanned by the columns of the matrices

$$\mathcal{K}_{n_i} = span\left(R_i, A_i R_i, A_i^2 R_i, ...A_i^{n_i-1} R_i\right) \tag{4.6}$$

where

$$\begin{aligned} R_i &= (G + s_i C)^{-1} B \\ A_i &= (G + s_i C)^{-1} C \end{aligned} \tag{4.7}$$

Our overall projection matrix is thus simply a union of our Krylov subspaces generated around each expansion point.

$$span(V) = \cup_{i=1}^{m} \mathcal{K}_{n_i} \tag{4.8}$$

It is to be noted that in the methodology described above we talk about matching a different number of moments n_i at each interpolation point. For our application, since we are only interested in the residue values for our poles of interest, this value n_i can be the same $\forall i$. Then the total order of the final system $q = m \times n$ where n is the order of expansion around each of the m interpolation points. The value of n should be sufficiently high to allow the transfer function to accurately capture our eigen value of interest. If we choose our interpolation points as our pole values of interest, the required value of n to effectively do so is not high as well.

Thus by generating reduced models around our pole values, we can approximate the response caused by those pole values via it's moments. With a sufficiently high order of expansion, the dominant eigen information present in the reduced model will be that which we seek along with some other eigen information to approximate the rest of the transfer function. The residue corresponding to the second order system that approximates our pole value within this reduced transfer function will give us the residue of our pole in the full system.

2. Block rational Arnoldi algorithm

The Block rational Arnoldi method presented in [17] is an adaptation of the rational Krylov algorithm that we use for our pole discovery stage. We shall make some performance improvements based upon other work as we go on. But the base methodology is described here.

Assuming that we are given G, C, B and all the expansions points μ_j, for each μ_j we can obtain the corresponding Krylov subspcace $\mathcal{K}_j$ as given in algorithm 4.

The final projection matrix V can be obtained as an orthonormal combination

Algorithm 4 BlockRationalArnoldiReal(G, C, B, μ_j)

```
1: $r \leftarrow -(G+\mu_j C)^{-1} B$
2: $r \leftarrow r/norm\,(r)$
3: $V_j\,(:,1) \leftarrow r$
4: for i=2:q do
5:     $V_j\,(i) \leftarrow -(G+\mu_j C)^{-1}\ C\ r$
6:     $V_j \leftarrow orthonormalize\,(V_j)$
7:     $r \leftarrow V_j(i)$
8: end for
9: return $V_j$
```

of all the V_j projection matrices we obtain above.

$$\begin{aligned} V &= [V_1, V_2, ...V_m] \\ V &= orthonormalize\,(V) \end{aligned} \tag{4.9}$$

To ensure that we obtain an orthonormal basis which is a requirement for performing congruence transformation, we will peform some sort of orthonormalization step as well. In the block rational Arnoldi method[17], this is done via block orthogonalisation as part of the basic algorithm instead of performing an orthonormalization step at the end.

The moment matching properties of this Algorithm hold due to the fact that V is a full rank matrix and that the union of Krylov subspaces generated at the various expansion points satisfy the inclusion formula $\cup_{i=1}^{m} \mathcal{K}_{n_i} \subseteq span\,(V)$. The sufficiency of these conditions for multipoint moment matching has been explored in [18].

Note that the additional cost of performing a multipoint expansion is simply LU factorisation at each new expansion point. Also, note that while this algorithm has been described for real interpolation points, it works equally well whether the selected shift μ_j is real or complex. However if we are to use complex expansion points, then our final projection matrix will be complex as well giving us complex $\hat{G}$, $\hat{C}$, $\hat{B}$ and $\hat{L}$ matrices. Preforming eigen decomposition for and generating residue information from such complex matrices will thus be more complex as well.

Borrowing the complex shift mechanism from [13] to obtain a real projection and applying it to our projection based model order reduction, we get the resulting algorithm as given below.

Algorithm 5 BlockRationalArnoldiComplex(G, C, B, μ_j)

```
1: $r \leftarrow -\left(G + \mu_j C\right)^{-1} B$
2: $r \leftarrow r/norm\left(r\right)$
3: $r1 \leftarrow real\left(r\right)$
4: $r2 \leftarrow imag\left(r\right)$
5: $V_j\left(1\right) \leftarrow r1$
6: $V_j\left(2\right) \leftarrow r2$
7: $V_j \leftarrow orthonormalize\left(V_j\right)$
8: $r \leftarrow r2$
9: for i=2:q do
10:     $r \leftarrow -\left(G + \mu_j C\right)^{-1}\ C\ r$
11:     $V_j(2i-1) \leftarrow real\left(r\right)$
12:     $V_j(2i) \leftarrow imag\left(r\right)$
13:     $V_j \leftarrow orthonormalize\left(V_j\right)$
14:     $r \leftarrow V_j\left(2i\right)$
15: end for
16: return $V_j$
```

Once we have our projection matrix, the reduced model can be generated by using congruence transformations as listed in equation 4.2.

C. Implementation aspects

While discussing the block rational Arnoldi algorithm, we went into detail as to how to compute the Krylov subspaces for each individual expansion point but not for the region of interest as a whole.

One of the methodologies to obtain the residue information for our second order approximations is to simply take a qth order expansion at each pole value of interest at a given node and use our projection matrix V_j as the complete projection matrix. We thus try to approximate the entire system as a second order system centered around that pole value. However, in doing so, the effect of surrounding and faraway poles on such a model is not insignificant. Increasing the order of expansion does not help beyond a particular limit either as we end up capturing more bogus poles and the associated subspace information hurts rather than helps our residue computation.

To really capture the residue information as accurately as possible we need to generate the reduced model as a function of all our poles of interest and not just one single pole. The larger subspace allows us to capture the original transfer function more accurately without having to generate very high order expansions around each point. Also, since the effect of a large majority of the complex poles have already been taken into account, we do not have disturbances from nearby poles or cluster of poles as we earlier faced. The more the number of second order systems we use to approximate the overall response, the more accurate our individual residue values are going to be.

Algorithm 6 given below allows us to do just that taking into account all the captured eigen values and not just the one closest to the pole we want to find the residue for. Also since we have all the eigen value information in a single projection matrix we will need to run QZ only once per node to obtain the residue information for the second order transfer functions.

A small point of note is that if we have a high number of poles in our region of interest, we may wish to split up our eigenvalues into clusters and use the algorithm that we are about to describe for each cluster. In this too, the effect of other pole clusters might affect our results somewhat but not as much as the effect on individual poles. This aspect is not discussed in more detail in this research work since all the practical test cases we have used so far have not required it.

Algorithm 6 GenerateProjectionMatrix($G, C, B, \mu_1, \mu_2 ... \mu_n$)

```
1: $V \leftarrow [\,]$
2: for all Eigen Values of interest as $\mu_j$ do
3:     Precompute LU factors for $(G + \mu_j C)$
4:     $r \leftarrow (G + \mu_j C)^{-1}\ B$
5:     $AddToProjectionMatrix\,(V, real\,(r))$
6:     $AddToProjectionMatrix\,(V, imag\,(r))$
7:     for i=2:q do
8:         $r \leftarrow V\,(:, -1)$
9:         $r \leftarrow -\,(G + \mu_j C)^{-1}\ C\ r$
10:        $AddToProjectionMatrix\,(V, real\,(r))$
11:        $AddToProjectionMatrix\,(V, imag\,(r))$
12:    end for
13: end for
14: $Orthonormalize\,(V)$
```

A couple of implementation aspects are of note here. One is that we need to perform LU factorization only once per expansion point. The second is that we perform Gram-Schmidt orthonormalization only once at the end of generating the complete projection matrix. At every other point we simply orthonormalize the last added column with respect to the previous columns as be described in the *AddToProjectionMatrix* method.

Algorithm 7 AddToProjectionMatrix(V,r)

```
1: if V != [ ] then
2:     $r \leftarrow r - V\ V'\ r$
3: end if
4: $r \leftarrow r/norm\,(r)$
5: $V \leftarrow [\ V\ r\ ]$
```

Computing the eigenvalues and the corresponding eigen vector information can be done directly using the QZ method owing to the small size of the system. This eigen information can in turn be used to project the circuit response of the reduced system into the eigen space to extract the corresponding residues.

D. Impedance calculation and dominant pole extraction

Once we have the residue information and pole information for all our poles of interest, computing the impedances and identifying the dominant poles is no different from the original loop finder algorithm as described in [4]. The only thing worthy of note here is that the impedance calculation and dominant pole extraction is also done on a per node basis. Since the residue information we generate using our reduced order models are only used in impedance computation, at the end of the per node impedance computation step, we only need to retain the impedance values of the dominant poles in each node. As a result the memory overhead is not too high as there is no requirement to store the response of each second order system for each node.

CHAPTER V

EXPERIMENTAL RESULTS

The base implementation is the same as what has been described in the previous two chapters. However the reference algorithm and the configuration parameters that we use by default for our methodology are mentioned in the following section.

A. Benchmark and implementation details

A variation of the original QZ based methodology described in [4] implemented in MATLAB has been used as the reference implementation. As described in section II.E, a significant performance improvement has been made before using as a benchmark to compare our methods against. The target implementation we are comparing it against has been implemented in both C and MATLAB. The C implementation proves to be a good comparison for the pole discovery stage since the pole discovery stage in QZ method is a LAPACK function call. The residue computation stage on the other hand is better compared with the MATLAB implementation since the residue computation of the QZ methodology is implemented in MATLAB. Unless otherwise mentioned, the target implementation results presented in the following section shall belong to the C implementation unless explicitly mentioned otherwise.

Unless explicitly mentioned, the maximum order used for model order reduction is 3 at every expansion point in both the MATLAB and C implementation. Thus the order of the final model being generated and solved to compute the residues is (number of poles of interest $\times 3 \times 2$). The last multiplication factor of 2 exists because every complex expansion point adds 2 columns to the projection matrix instead of 1. Also our maximum frequency of interest f_{max} and the damping tolerance ζ_{th} are set to 10Ghz and 0.7 respectively unless mentioned otherwise. Lastly an errortol of

1e-4 and a sensitivity of 5 (conservative pole discovery) has been used in the pole discovery stage.

B. Results

All circuits are based on real test cases provided by Texas Instruments. All tests were run on the same test machine albeit under differing loads at times. The machine has 8GB of RAM which proves to be a bottleneck for the QZ method after a certain number of equations as shall be described later.

1. Accuracy

The pole discovery algorithm provides for complete coverage within our region of interest. All the poles in our region of interest for all circuit test cases were discovered accurately. Repeated poles of course were reported as one single pole unlike in the QZ method which could report the repeated poles seperately. Our values differ by at most three decimal points of accuracy when all natural frequencies are expressed in Mhz and when our errotol is set to 1e-4. This is the worst case scenario and for most cases it is accurate to four decimal points. The additional check imposed by the last subdivision enables us to converge very accurately onto our pole value.

The accuracy of the residue values and thus the extracted loops depends upon the order q of expansion around each pole value. For values 3 and above we get consistently accurate values within an error bound of 10% of the value obtained from QZ. We choose to keep the q value at 3 even though increasing the order might yield better results especially in larger circuits. This is because as we increase the order per expansion point, our overall model size for residue computation increases based on the number of poles of interest as per the formula (number of poles of interest $\times$

order $\times 2$)

2. Speed

As can be seen in the following tables, for a lower number of equations, direct QZ decomposition definitely wins out. However, QZ has cubic complexity and the cost of pole discovery at least increases much faster for QZ than for our modified rational Krylov Algorithm.

Please note that the QZ method is a MATLAB internal function and makes a direct call to LAPACK thus giving it an edge over the rational Krylov Algorithm which is implemented in MATLAB. So the real comparision that should be made here is between the pole discovery time via QZ and the rational Krylov Algorithm implemented in C which is what is presented in Table I.

Table I. Pole discovery times for various circuit configurations

No. Of Equations	Damping Tolerance	Pole Discovery Time		Speed up
		Krylov	QZ	
556	0.9	3.0s	0.38s	0.13x
2901	0.7	26.47s	114.58s (1.9min)	4.33x
5202	0.7	41.136s	901.63s (15min)	22x
5569	0.7	162.49s (2.71min)	623.28s (10.4min)	3.84x
6361	0.7	627.75s (10.46min)	676.12s (11.26min)	1.1x

While we observe significant speed ups for some of the test circuits, it is also of note that the circuit type (and thus the resulting pole distribution) also has a significant effect on speedups. In particular, the circuit with 6361 equations belongs to a different family from the rest and does not follow our expected trend. This is why we cannot rely on performance improvements purely from the pole discovery stage alone.

Table II details the performance improvement we gain from the residue computation stage by generating reduced models instead of using the QZ based methodology. Note that the major part of the savings in this step arises from the fact that we deal with sparce matrices for MOR instead of the dense eigen vector matrix that we use in the QZ methodology. Also of note is that even a minor performance improvement on a per node basis can lead to a significant improvement for the algorithm as a whole as the residue information needs to be generated for every node.

The residue computation stage via the QZ methodology has been implemented in MATLAB. Hence along with the results for our target implementation in C we are also including the results for the MATLAB implementation of our residue computation methodology.

Note that the MATLAB version of our target implementation is not the most optimized version. The time required for the MATLAB implementation can be further reduced by utilizing column orthonormalization in place of block orthonormalization. (This has been implemented in the C code)

Thus the speedup value we are providing for the residue computation portion is with respect to the C code. While this does not prove to be an accurate comparison, the real speedup will be somewhere between the speedup from the MATLAB and C implementations.

As can be seen from Table II, leveraging the inherent sparseness of our circuit

Table II. Residue computation time for various circuit configurations

No. Of Equations	No. of Nodes	No. of Poles	Execution Time			Speedup
			MOR (C)	MOR (MATLAB)	QZ (MATLAB)	MOR (C)
556	289	3	0.7s	2.09s	3.52s	5x
2901	1310	4	24.08s	401.29s (6.7 min)	543.6s (9.06 min)	22.57x
5202	1700	7	160.63s (2.68 min)	954.08s (15.90 min)	2219.49s (37 min)	13.8x
5569	2526	7	244.75s (4.1 min)	1646.96s (27.45 min)	4234.83s (1.18 hrs)	17.3x
6361	2026	5	126.45s (2.1 min)	1445.95s (24.1 min)	3399.49s (56.66 min)	26.8x

matrices allows us to gain significant performance improvement over the QZ based methodology. In the QZ method, we do dense linear solves after LU Factorization of the eigen vector matrix. For the MOR method, generating the reduced model and the eigen decomposition of the reduced model dominates the cost of residue computation. However the reduced model is almost constant in size as compared to the number of nodes and so the cost of eigen decomposition of the reduced model can be neglected. Cost of generating reduced model is thus dominated by cost of sparse linear solves and so in the long run ends up proving faster.

An interesting point that arises from the above two results is which step of our automatic stability analysis algorithm benefits the most from the performance improvements. Table III lists a comparison of pole discovery and residue computation times of some real circuits. The last two testcases in the table also belong to the class of very large circuits that we wish to run automatic stability for.

Table III. Comparison of pole discovery and residue computation times

No. Of Equations	Damping Tolerance	No. of Nodes	No. of Poles	Time elapsed		
				Pole Discovery	Residue Computation	Total Time
556	0.9	289	3	3.0s	0.7s	3.7s
2901	0.7	1310	4	26.47s	24.08s	50.55s
5202	0.7	1700	7	41.136s	160.627s	3.4 min
5569	0.7	2526	7	2.71 min	4.08 min	6.79 min
6361	0.7	2026	5	10.46 min	2.11 min	12.57 min
17447	0.7	4773	8	13min	55.4 min	1.14hrs
39723	0.7	26675	12	54min	31.9 hrs	32.8rs

Observe that the residue computation costs dominate over the pole discovery costs for both the QZ and MOR based methods except for smaller circuit sizes. This has to do with the fact that the residue computation has to be done at every node in the circuit or at least every node of interest in the circuit (provided that the designer is only interested in a small subset of the nodes).

The total execution time taken by our two methodologies is listed in Table IV. Note that the speedup factor given here is not accurate since the QZ based methodology is implemented in MATLAB while the new approach exploiting rational Krylov algorithm and model order reduction has been implemented in C. However the figures do indicate the potential improvement that can be obtained over currently existing loop finding technology. The "-" entries correspond to the circuit matrices which the QZ based loop finder could not handle.

Table IV. Total execution times for various circuit configurations

No. Of Equations	Damping Tolerance	No. of Nodes	No. of Poles	Time elapsed		Speedup
				QZ (MATLAB)	New approach (C)	Time
556	0.9	289	3	3.7s	3.9s	1.05x
2901	0.7	1310	4	50.55s	11 min	13x
5202	0.7	1700	7	3.4 min	52 min	15.3x
5569	0.7	2526	7	6.79 min	1.35 hrs	12x
6361	0.7	2026	5	12.57 min	1.13 hrs	5.4x
17447	0.7	4773	8	1.14hrs	-	-
39723	0.7	26675	12	32.8rs	-	-

It is important to note from the overall execution time that for smaller circuit sizes (less than 1000), a direct QZ based approach is a better choice. However as the circuit size grows, the modified rational Krylov algorithm followed by reduced model residue extraction proves to be the better choice. For extremely large circuits such as those which we are targetting for automatic stability analysis, our new methodology proves to be the only choice.

3. Memory footprint

Our rational Krylov algorithm followed by Model Order reduction has a low memory footprint since at any time we only have sparse matrices in memory. Additionally, we clean up after ourselves after every partition in the rational Krylov algorithm and every node in the residue computation stage and hence our total memory usage stays under control.

On the other hand, the QZ based methodology always deals with dense matrices. Both for the pole discovery stage (QZ) and for the residue computation stage (Dense matrix solve using the eigen vector matrix). For large circuit sizes of above 10,000 nodes, the memory footprint proves to be a very real bottleneck.

CHAPTER VI

CONCLUSION

As can be seen, the new approach exploiting the modified Rational Krylov Algorithm followed by residue extraction using model order reduction allows us to extend the loop finder methodology to larger circuits than was previously possible. Additionally, it also provides significant speedup over the previously used QZ method.

However, depending upon the type of circuit under investigation, the speedups may differ. A larger number of real world testcases are required to accurately benchmark the performance improvement we obtain by using one method over the other. Additionally, a lot of scope exists for further speed up of the algorithm by means of parallel programming. This is enabled by two factors. One is the fact that the algorithm involves analyzing several discrete partitions which can be analyzed independently of each other during the pole discovery stage. And the second is the residue computation stage where several nodes may be analyzed in parallel or the moment computation for various poles within a node can be done in a parallel fashion.

Lastly, the generalised pole discovery methodology described here can also be possibly extended to other applications requiring region based pole search.

REFERENCES

[1] R. D. Middlebrook, "Measurement of loop gain in feedback circuits," *International Journal of Electronics*, vol. 38, pp. 485–512, Apr. 1975.

[2] M. Tian, V. Visvanathan, J. Hantgan, and K. Kundert, "Striving for small-signal stability," *Circuits and Devices Magazine, IEEE*, vol. 17, pp. 31 –41, Jan. 2001.

[3] W. K. Chen, *Active Network and Feedback Amplifier Theory*, pp. 185–246. New York: McGraw Hill, 1980.

[4] G. P. Fang, R. Burt, and N. Dong, "Loop finder analysis for analog circuits," in *IEEE Custom Integrated Circuits Conference*, (San Jose, California, United States), Sept. 2010.

[5] B. S. Garbow, "Algorithm 535: The qz algorithm to solve the generalized eigenvalue problem for complex matrices," *ACM Trans. Math. Softw.*, vol. 4, pp. 404–410, Dec. 1978.

[6] E. Kuh and R. Rohrer, *Theory of Linear Active Networks*, pp. 520–605. San Fransisco, CA: Holden-Day, 1967.

[7] R. Mancini, *Op Amps for Everyone.* Dallas, TX: Texas Instruments, Aug. 2002.

[8] R. C. Dorf and R. H. Bishop, *Modern Control Systems.* Prentice Hall, 11 ed., 2007.

[9] R. Burt, "Stability: A method for evaluating stability of continuous-time close-loop systems," tech. rep., Texas Instruments, 2004.

[10] M. Milev and R. Burt, “A tool and methodology for ac-stability analysis of continuous-time closed-loop systems,” in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 3*, DATE ’05, pp. 204–208, 2005.

[11] A. Ruhe, “Rational krylov algorithms for nonsymmetric eigenvalue problems,” *Recent Advances in Iterative Methods, IMA Volumes in Mathematics and its Application*, vol. 60, pp. 149 – 164, 1993.

[12] A. Ruhe, “Rational krylov algorithms for nonsymmetric eigenvalue problems. ii. matrix pairs,” *Linear Algebra and its Applications*, vol. 197-198, pp. 283 – 295, 1994.

[13] A. Ruhe, “The rational krylov algorithm for nonsymmetric eigenvalue problems. iii: Complex shifts for real matrices,” *BIT Numerical Mathematics*, vol. 34, pp. 165–176, 1994.

[14] E. Chiprout and M. Nakhla, “Analysis of interconnect networks using complex frequency hopping (cfh),” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 186 –200, Feb. 1995.

[15] A. Ruhe and D. Skoogh, “Rational krylov algorithms for eigenvalue computation and model reduction,” in *Applied Parallel Computing Large Scale Scientific and Industrial Problems* (B. Kgstrm, J. Dongarra, E. Elmroth, and J. Wasniewski, eds.), vol. 1541 of *Lecture Notes in Computer Science*, pp. 491–502, Springer Berlin / Heidelberg, 1998.

[16] A. Odabasioglu, M. Celik, and L. T. Pileggi, “Prima: passive reduced-order interconnect macromodeling algorithm,” in *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, ICCAD ’97, pp. 58–65, Nov. 1997.

[17] I. Elfadel and D. Ling, “A block rational arnoldi algorithm for multipoint passive model-order reduction of multiport rlc networks,” in *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, ICCAD ’97, pp. 66 –71, Nov. 1997.

[18] E. Grimme, *Krylov Projection Methods for Model Reduction.* PhD thesis, University of Illinois at Urbana-Champaign, 1997.

VITA

Parijat Mukherjee received his Bachelor of Technology degree in Electronics and Communication Engineering from National Institute of Technology, Tiruchirappalli, India in 2007. From 2007 to 2008, he worked with the Custom Circuit Design team at IBM India Pvt Ltd. before entering Texas A&M University to pursue his Master of Science Degree in Computer Engineering in 2008. His reseach interests lie in the general areas of VLSI systems and electronic design automation in particular focussing on analog and mixed-signal CAD and parallel algorithms.

Mr. Parijat may be reached at WERC 303, 3259 TAMU, College Station, TX 77843, USA. His email id is parijatm@gmail.com.