High-Order Time-Domain Steady-State Simulation for Radio Frequency Applications

Nassir D. Abou Ziki

Master of Engineering

Department of Electrical and Computer Engineering

McGill University

Montreal, Quebec

2015-03-08

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Master of Engineering in Electrical Engineering

DEDICATION

To my parents, *amare et sapere vix deo conceditur*.

ACKNOWLEDGEMENTS

First and foremost, I want to thank Professor Roni Khazaka for his guidance and support, without his knowledge and advice this work would not have come to fruition. I also thank the circuit simulation group at Carleton University and mainly Professor Michel Nakhla and Mr. Mina Farhan for the valuable discussions that we had during the development of this thesis.

I thank my colleagues and desk neighbours in the MACS lab Marwan Kanaan, Gabi Sarkis, Muhammad Kabir, François Leduc-Primeaufor and Julian Santorelli for their continuous support and for making the journey towards the completion of this work enjoyable and enriching.

I would like to give special thanks to my parents, Daher Abou Ziki and Lydia Hassanieh as well as colleagues and best friends Marco Kassis and Raffi Toukhtarian for their endless support, advice and words of wisdom that helped me get through all the difficult times.

Finally, I thank my sister Jana Abou Ziki, my brother Maen Abou Ziki, my brother-in-law Firas Dweik and all my friends who always provided me with invaluable support in my life outside the lab.

ABSTRACT

Steady-state simulation is a crucial step in the design of Radio Frequency (RF) circuits. The most popular steady-state simulation methods include Harmonic Balance (HB) in the frequency domain and Shooting methods in the time domain. Both of these methods have their drawbacks; while HB can be computationally expensive due to the large number of unknowns that are required to be found, Shooting methods suffer computationally due to the necessity of computing the sensitivity matrix. This work focuses on alleviating the computational costs of Shooting methods. A high order Shooting-Newton method for steady-state simulation of Radio Frequency (RF) circuits is presented. The method is based on high order A- and L-stable Obreshkov formula (ObF). The proposed Shooting method can have arbitrarily high-order without losing the stability property. The high-order methods allow using bigger step sizes in numerical integration; the stability property guarantees that the accuracy of the numerical solution is not compromised. Consequently, the presented method allows for a considerable improvement in CPU cost when compared to the Backward Euler based approach. Speedups of more than 8 are reported.

ABRÉGÉ

La simulation en régime permanent est une étape cruciale de la conception de circuits à radio-fréquences. Parmi les méthodes couramment utilisées pour la simulation en régime permanent, on retrouve notamment la méthode d'équilibre harmonique dans le domaine fréquentiel et les méthodes de *shooting* dans le domaine temporel. Chacune de ces méthodes comporte des lacunes. La méthode d'équilibre harmonique peut demander une forte puissance de calcul à cause du grand nombre d'inconnus dans le système, tandis que les méthodes de *shooting* sont également complexes dû à la nécessité de calculer la matrice de sensitivité. Ce mémoire porte sur la réduction du coût de calcul des méthodes de *shooting*, et présente une méthode Shooting-Newton d'ordre supérieur pour la simulation en régime permanent de circuits radio-fréquences. La méthode proposée est fondée sur des formules "A-stable" et "L-stable" d'Obreshkov d'ordre supérieur. L'ordre de la méthode de *shooting* peut être arbitrairement grand sans perte de stabilité. Le recours à des méthodes d'ordre supérieur permet d'utiliser des pas plus grands lors de l'intégration numérique, et la stabilité de la méthode garantie la précision de la solution. Conséquemment, la méthode proposée permet une amélioration considérable du coût de calcul par rapport aux approches fondées sur la méthode d'Euler implicite. Les résultats présentés démontrent une amélioration du temps de calcul de plus de 8 fois.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Background and Motivation

To characterize a radio frequency (RF) circuit properties like distortion, power, frequency, noise, gain and impedance need to be measured. These properties are usually characterized at steady-state solution [1]. Hence, non-linear steady-state analysis is a crucially important step in the design process of RF circuits. Methods to perform the non-linear steady-state analysis exist in both the frequency domain and the time-domain.

Harmonic Balance (HB) is perhaps the most widely used frequency-domain steady-state simulation method. It was first first implemented by Baily [2] around 1968, which was then followed by several works on new methods to solve HB [3, 4]. In HB each of the state state variables is represented using a Fourier Series, whose coefficients are adjusted using an optimization algorithm such that the equations describing the system are satisfied with the least error. The major advantage in HB is that it avoids performing the computationally expensive numerical integration of the system equations. However, its main disadvantage is that the number of variables that need to be solved for is large. For instance, a system that has N state variables and each state variable requires $2M + 1$ Fourier coefficients, then this means that there are $N(2M + 1)$ variables to be solved for. This makes HB especially expensive and perhaps impractical of large systems.

In time-domain methods, finding the steady state solution can be done in several ways. The steady-state solution is defined as the solution that is asymptotically approached by the system after the effect of the initial conditions has died out. Hence, a rather straightforward way to obtain the steady-state solution is to perform a transient analysis long enough until the system reaches its steady-state. This method for finding the steady-state solution is far from ideal. While it could work well for circuits whose transients decay fast, it presents a lot of challenges for systems where the transients die out slowly, which means the numerical integration has to be done over a larger time interval resulting in expensive time and computational costs. Therefore, time-domain numerical methods were developed specifically to tackle this problem. The Shooting methods [5] are the most popular of these methods.

The main disadvantage in Shooting methods is due to computing the sensitivity matrix, which necessitates several numerical integrations over one period. Finding the steady-state response of a circuit can be formulated as a boundary-value problem (BVP). Shooting methods solve such BVPs by finding the initial conditions for a system of differential equations such that the solution after one period is equal to the initial conditions, i.e. $x(0) = x(T)$. This is done by solving the system's differential equations for one period starting from some consistent set of initial conditions, like the DC solution. Then, after one period, the initial conditions are iteratively updated using a Newton iteration step. This combination is usually referred to as Shooting-Newton algorithms, and they are an iterative layering on top of a transient analysis [6]. Therefore, as is the case with time-domain transient methods, the accuracy and efficiency of the Shooting methods are limited by the stability and the order of the integration method being used [7]. This is especially problematic in RF circuits where the presence of widely separated tones at the output will force the use of a very small time step size compared to the period of the system, and hence will render finding the steady-state solution computationally expensive. The work in this thesis focuses on mitigating the computational costs of time-domain steady-state analysis using Shooting methods.

1.2 Thesis Contribution and Organization

In this thesis we present a high-order Shooting-Newton method based on high-order stable integration methods. The presented method takes advantage of the high-order to allow fast and accurate simulations. Since stability is not breached at the higher orders, larger step sizes can be used without compromising the accuracy of the method. The presented algorithm is based on the recently developed A- and L-stable modified Obreshkov Formula (ObF) which allows integration methods with arbitrarily high order [8, 9, 10].

This thesis is organized into seven chapters. Following this chapter, chapter 2 presents a review of fundamental theoretical concepts that will be referred to throughout the thesis. Chapter 3 presents a review of various integration methods for solving differential equations. Chapter 4 gives a summary of some practical considerations to aid the implementation of the Obreshkov-based integration methods. Chapter 5 outlines the derivation of the high-order Shooting methods. In chapter 6, numerical examples are presented to verify the performance of the high-order integration methods for both transient and steady-state analyses. Finally, chapter 7 gives a summary of the contributions of this work.

CHAPTER 2
Theoretical Framework

This chapter serves as a presentation of important theoretical concepts that will be revisited throughout this work and that will be essential to present the algorithm proposed in this thesis. Time-domain simulation can be mathematically represented as an initial value problem in the case of transient analysis, or as a boundary value problem in the case of steady-state analysis. Therefore, this chapter begins with a general definition of these problems in sections 2.1 and 2.2. In the later two sections of this chapter the concepts of the order of an integration method and the stability of an integration methods are introduced. The order of the integration method determines how closely an integration method, given a certain time step at a certain iteration, can estimate the true solution of the problem at the next time step. In general, the higher the order of the method the better it is at estimating the solution. The concept is important in the context of circuit simulation because it offers a way to study the errors introduced by the numerical integration process. Finally, this chapter introduces the concept of stability of the integration method which is a required property in order to obtain an accurate solution. The method of characterizing the stability property of an integration method is presented and the different types of stability are discussed.

In general, problems involving numerical integration aim to solve a set of ordinary differential equations (ODEs). When performing an integration the result will have unknown integration constants.

Consider the following system of equations,

$$\begin{aligned} y' &= z \\ z' &= -y. \end{aligned} \tag{2.1}$$

The solution of the system is as follows,

$$\begin{aligned} y &= A\sin(t+\alpha) \\ z &= A\cos(t+\alpha). \end{aligned} \tag{2.2}$$

Notice that the solution has two unknown integration constants (A and α). A solution in this form is often referred to as the general solution of the system which describes a family of functions parametrized by the integration constants. To be able to arrive at a specific solution, the problem needs to be defined further by including constraints or conditions on the dependent variables (y and z in this case) at specific values of the independent variable t. If these conditions are given at the initial time t_0 then the problem being solved is called an initial value problem (IVP). However, if the conditions on the dependent variables are given at different instances of the independent variables then the problem is called a boundary value problem (BVP) [11]. Hence, for an IVP the integration starts at the initial point with all the solution information and marches with it in time, such a process is called local. On the other hand, for BVPs the the solution information is not locally known anywhere and thus the process of constructing a solution is global in time [12]. Consequently, finding solutions for BVPs presents more difficulties when compared to IVPs, some of these difficulties will be highlighted later in section 2.2 when the topic of BVPs is revisited.

2.1 The Initial Value Problem

The initial value problem is often represented by the following equation

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), t), \quad 0 \leq t \leq b \tag{2.3}$$

with $\boldsymbol{x}(0) = \boldsymbol{x_0}$. It is enough to require $\boldsymbol{f}(\boldsymbol{x}, t)$ to be sufficiently smooth to assure that equation (2.3) has a unique solution. Specifically, let $\boldsymbol{f}(\boldsymbol{x}, t)$ be continuous in a region $\mathcal{D} = \{0 \leq t \leq b, |\boldsymbol{x}| < \infty\}$ and Lipschitz continuous in $\boldsymbol{x}$, i.e., there exists a constant L such that for all $(t, \boldsymbol{x})$ and $(t, \hat{\boldsymbol{x}})$ in $\mathcal{D}$,

$$|\boldsymbol{f}(\boldsymbol{x}, t) - \boldsymbol{f}(\hat{\boldsymbol{x}}, t)| \leq L|\boldsymbol{x} - \hat{\boldsymbol{x}}|. \tag{2.4}$$

Then

1. For any $\boldsymbol{x_0}$ there exists a unique solution $\boldsymbol{x}(t)$ for the IVP (2.3) for $t \in [0, b]$. This solution is differentiable.
2. The solution $\boldsymbol{x}$ depends continuously on the initial data: if $\hat{\boldsymbol{x}}$ also satisfies the ODE but not the same initial values then

$$|\boldsymbol{x}(t) - \hat{\boldsymbol{x}}(t)| \leq e^{Lt}|\boldsymbol{x_0} - \hat{\boldsymbol{x}}_{\boldsymbol{0}}|. \tag{2.5}$$

3. If $\hat{\boldsymbol{x}}$ satisfies, more generally, a perturbed ODE

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{f}(\hat{\boldsymbol{x}}, t) + \boldsymbol{r}(\hat{\boldsymbol{x}}, t)$$

where $\boldsymbol{r}$ is bounded on $\mathcal{D}$, $||\boldsymbol{r}|| < M$, then

$$|\boldsymbol{x}(t) - \hat{\boldsymbol{x}}(t)| \leq e^{Lt}|\boldsymbol{x}_{\mathbf{0}} - \hat{\boldsymbol{x}}_{\mathbf{0}}| + \frac{M}{L}(e^{Lt} - 1). \tag{2.6}$$

Consequently, a unique solution exists for $t \in [0, b]$ that continuously depends on the initial conditions, hence the problem subject to the conditions outlined above is well-posed [12].

2.2 The Boundary Value Problem

As mentioned earlier, BVPs arise when the goal is to find a solution for a system of ODEs that is required to satisfy subsidiary conditions at two or more distinct instants of time. For instance, finding the solution of the equation

$$\dot{\boldsymbol{x}} \quad = \quad \boldsymbol{f}(\boldsymbol{x}(t), t), \tag{2.7}$$

over an interval $[0, T]$ is a two-point boundary problem if the solution is required to satisfy

$$\boldsymbol{g}(\boldsymbol{x}(0), \boldsymbol{x}(T)) \quad = \quad \mathbf{0}. \tag{2.8}$$

Boundary problems are of interest in circuit simulation because the problem of finding a steady-state solution whether it is periodic or quasi-periodic can be formulated as a boundary problem.

While the theory for existence and uniqueness of a solution to an IVP is well defined as described in section (2.1), the theory for existence and uniqueness of a solution for a BVP is more complicated and less thoroughly developed, sometimes it is not even possible to find a solution for a BVP. For example consider the following system,

$$\dot{\boldsymbol{x}} \;=\; \boldsymbol{x}(t) \tag{2.9}$$

with boundary constraint

$$\boldsymbol{x}(T) - \boldsymbol{x}_0\, e^t \;=\; 1. \tag{2.10}$$

The solution of equation (2.9) is given by

$$\boldsymbol{x}(t) \;=\; \boldsymbol{x}_0\, e^t. \tag{2.11}$$

Therefore, by substituting the solution in equation (2.11) at time $t = T$ into the constraint (2.10), it can be seen that the solution doesn't satisfy the boundary constraint. Hence this BVP has no solution .

Even though there is no general theory regarding the existence and uniqueness of a solution for a BVP, there are robust theorems developed to deal with individual problems. In the context of steady-state circuit simulation the solution is usually linearly constrained at two points. This allows for reducing the solution of the BVP to that of an initial value problem. Consider the system of N first order differential equations

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), t), \tag{2.12a}$$

subject to the most general linear two-point boundary conditions

$$A\boldsymbol{x}(0) + B\boldsymbol{x}(T) = c\,, \tag{2.12b}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^N$, $f : \mathbb{R}^{N+1} \to \mathbb{R}^N$, A, $B \in \mathbb{R}^{N\times N}$, and $c \in \mathbb{R}^N$.

The solution of the BVP presented in (2.12) reduces to the solution of the initial value problem

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), t), \tag{2.13a}$$

$$\boldsymbol{x}(0) = \boldsymbol{x}_0\,, \tag{2.13b}$$

where $\boldsymbol{x}_0$ is chosen to satisfy the implicit nonlinear equation

$$A\boldsymbol{x}_0 + B\boldsymbol{\phi}(\boldsymbol{x}_0, 0, T) - c = 0\,, \tag{2.14}$$

and $\boldsymbol{\phi}(\boldsymbol{x}_0, t_0, t)$ is the state transition function, which is the solution of the initial value problem (2.13) at time t starting from state $\boldsymbol{x}_0$ at time t_0. Hence, if $\boldsymbol{x}_0$ is the root of (2.14), then $\boldsymbol{x}(t) = \boldsymbol{\phi}(\boldsymbol{x}_0, 0, t)$ is the solution of the BVP (2.12) [1]. This result is best summarized in the following theorem.

Theorem 2.1 *Let the function $\boldsymbol{f}(\boldsymbol{x}(t), t)$ be continuous in t over the interval [0, T] for all $\boldsymbol{x}$ and Lipschitz continuous in $\boldsymbol{x}$, uniformly in t. Then the boundary value problem* (2.12) *has as many solutions there are distinct roots $\boldsymbol{x}_0^{(j)}$ of equation* (2.14)*. These solutions are*

$$\boldsymbol{x}^{(j)}(t) = \boldsymbol{\phi}(\boldsymbol{x}_0^{(j)}, 0, t),$$

the solutions of the initial value problem (2.13) *with initial state $\boldsymbol{x}(0) = \boldsymbol{x}_0^{(j)}$ [13], [1].*

2.3 Order and Local Error of Integration Methods

Solving an ODE using numerical integration methods basically provides an approximation to the exact solution. The approximation arises due to the presence of inherent errors in the numerical integration process such as truncation errors and round-off errors. The truncation error basically is the difference between the exact solution and the approximated one, it arises from the fact that the exact solution of the differential equation is not known and the approximated solution is reached by solving a different problem dictated by the integration method used. The round-off errors arise from the fact that the numbers cannot be expressed exactly, instead they are expressed in finite precision [11]. Consequently, characterizing the error and its accumulation during the numerical integration process has been a major concern in the study of numerical integration methods as it indicates a measure of the accuracy of the numerical method used [8].

A method is said to be of order p if

$$\boldsymbol{x}_n = \boldsymbol{x}(t_n) + Ch_n^{p+1}\frac{d^{p+1}}{dt^{p+1}}\boldsymbol{x}(t)\Big|_{t=t_n} + \mathcal{O}(h_n^{p+2}), \tag{2.15}$$

where $\boldsymbol{x}_n$ is the approximation of the exact solution $\boldsymbol{x}(t)$ generated by the integration method at time $t = t_n$, and $h_n = t_n - t_{n-1}$ is the time step. C is the error constant which is a characteristic of the integration method used. The local truncation error (LTE) of the integration method is thus given by the $(p+1^{th})$ term in equation (2.15), hence,

$$\boldsymbol{d}_n = Ch_n^{p+1}\frac{d^{p+1}}{dt^{p+1}}\boldsymbol{x}(t)\Big|_{t=t_n}. \tag{2.16}$$

The LTE as defined in (2.16) is an approximation of the error that the difference method is introducing as the solution evolves a single step forward assuming perfect knowledge of the exact solution in the previous step. Therefore, another measure of the error is devised which is defined as the difference between the exact solution $\boldsymbol{x}(t_n)$ and the approximated solution $\boldsymbol{x}_n$ at each time step [8]. This measure is called the local error and is given by,

$$\boldsymbol{l}_n = \boldsymbol{x}(t_n) - \boldsymbol{x}_n. \tag{2.17}$$

It is shown in [12] that for the numerical integration methods considered in this work that LTE is related to the local error by,

$$|\boldsymbol{d}_n| = |\boldsymbol{l}_n|(1 + \mathcal{O}(h_n)). \tag{2.18}$$

Hence the LTE given in (2.16) is asymptotically the same as the local error defined in (2.17).

2.4 Stability

In general the stability of a numerical method is an indication of its accuracy; a method is said to be stable if small changes in initial values result in bounded changes in the numerical approximations provided by the method. The stability of a numerical method is usually determined by characterizing its behavior in approximating the solution of the scalar test problem [11],

$$\begin{aligned} \dot{x} &= \lambda x, \\ x(0) &= x_0, \end{aligned} \tag{2.19}$$

where λ is a constant complex number representing the eigen value of the system's matrix and $\Re e(\lambda) < 0$. The solution of the scalar test problem is

$$x(t_n) = x_0 e^{\lambda t_n}. \tag{2.20}$$

The solution is exponentially decaying for $\Re e(\lambda) < 0$, hence it is expected that the sequence of approximations of the solution (2.20) provided by the numerical method to satisfy,

$$|x_n| \leq |x_{n-1}|, \quad n = 1, 2, 3, \cdots \tag{2.21}$$

which is known as the absolute stability requirement [8]. For a numerical method the region of absolute stability is defined as the region in the complex plane where $q = h\lambda$ in this region results in an approximation of the solution of the scalar test problem (2.19) satisfying the absolute stability requirement (2.21). This means that inorder to approximate the naturally stable system (2.19), h needs to be chosen such that $h\lambda$ lies in the region of absolute stability. Figure 2–1 shows the absolute stability region of the forward Euler method.

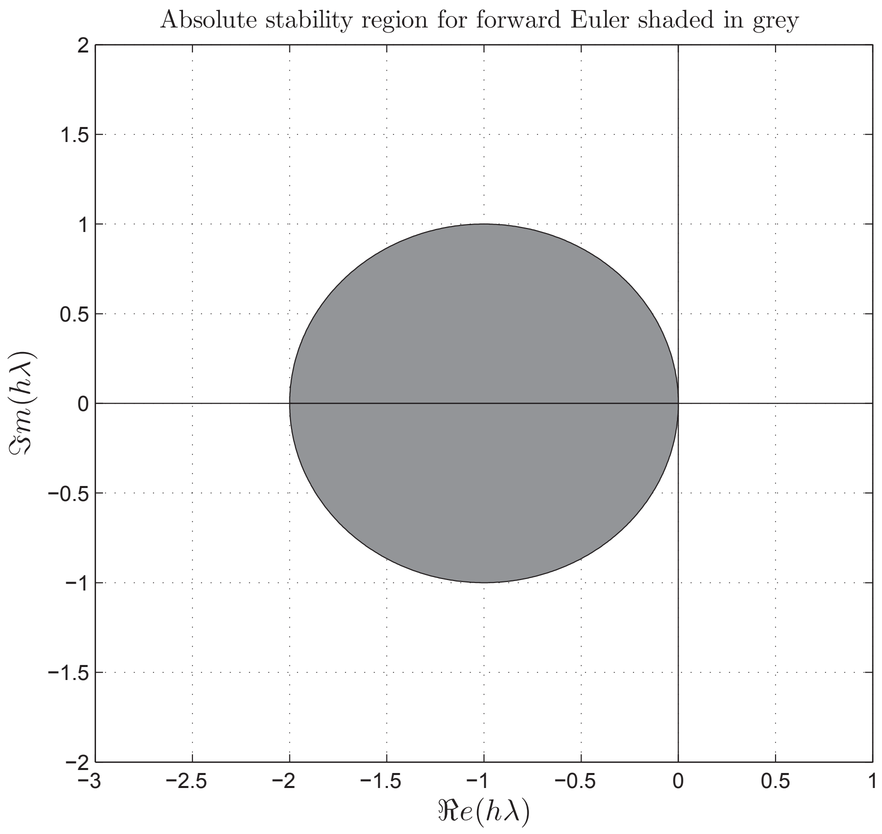

Figure 2–1: Stability region for forward Euler method

2.4.1 A-stability of Integration Methods

Ideally, the step size h requirements should be dictated by approximation accuracy not by the stability region. An integration method is said to be $A-$-stable if its region of absolute stability contains the entire left plane of the complex plane. Hence, an integration method is A-stable if it leads to a bounded solution for the scalar test problem (2.19) for any step size h for all $\Re e(\lambda) < 0$. For example,the Backward Euler and the trapezoidal rule are both A-stable integration since their regions of stability include the entire left half of the complex plane as shown in figures 2–2 and 2–3 below.

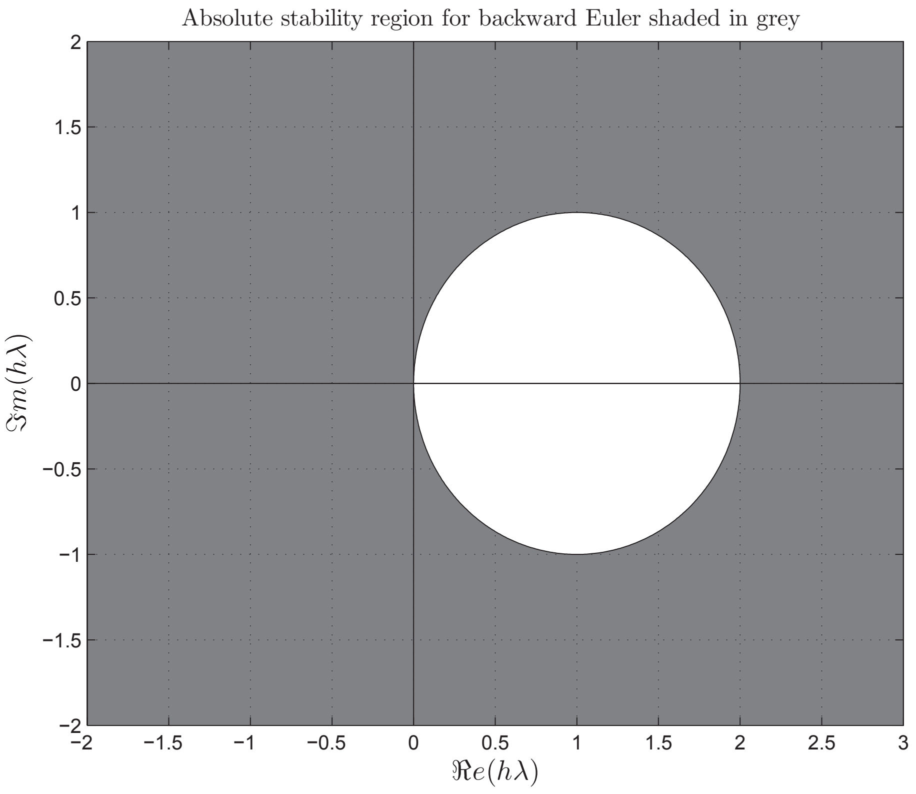

Figure 2–2: Stability region for backward Euler method

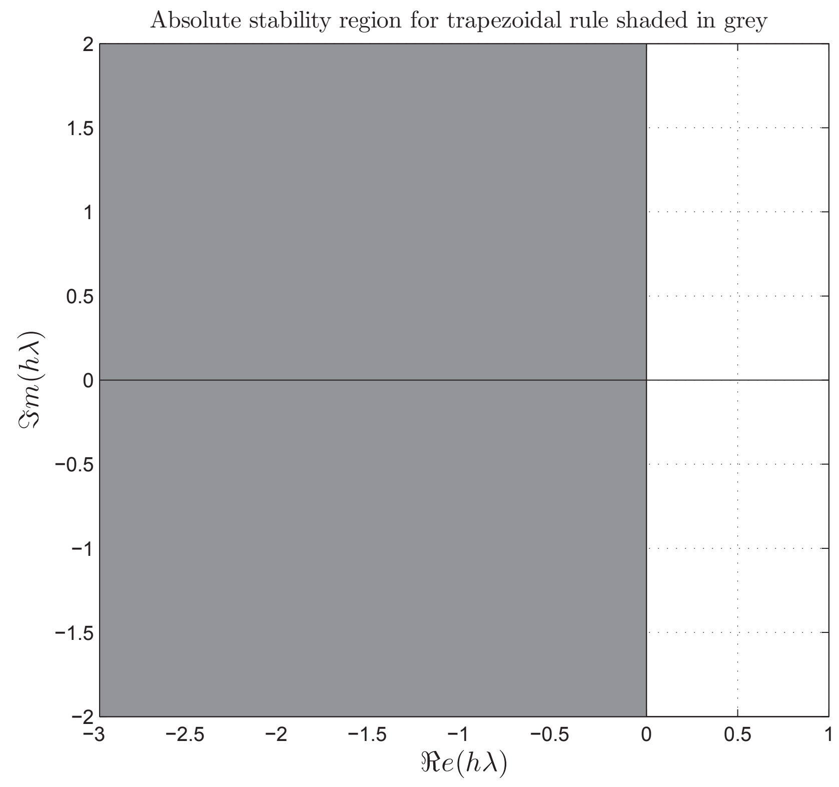

Figure 2–3: Stability region for trapezoidal rule

2.4.2 L-stability of Integration Methods

A main shortcoming of A-stable methods is that they are not suitable for solving stiff systems. There is no general robust definition of a stiff system, however, a system usually arises when the underlying physical system has components with widely deferring time constants [14]. This could lead to a highly oscillatory transient component in the solution of the system, which forces the adoption of excessively small step size to obtain accurate approximations even though the method is A-stable.

Consequently, the concept of L-stability was developed. A method is said to be L-stable if it is A-stable and has stiff decay [11], [15], i.e. when it is applied to the scalar test equation (2.19) the following condition is satisfied,

$$\text{as } \Re e(h\lambda) \to -\infty, \text{ then } \frac{x_{n+1}}{x_n} \to 0. \tag{2.22}$$

L-stable methods would cause cause rapidly decaying components to decay rapidly in the numerical approximation as well, hence it would not be necessary to use an excessively small step size [11].

2.4.3 $A(\alpha)$-stability of Integration Methods

Constructing A-stable or L-stable methods with high order is a difficult endeavor, hence depending on the problem at hand it is possible to relax the stability requirement. This leads to the rise of $A(\alpha)$-stable methods which are stable over a limited area of the left half side of the $(h\lambda)$-complex plane. A method is called $A(\alpha)$-stable if its region of absolute stability contains all $h\lambda$ such that,

$$\Re e(h\lambda) \leq 0 \quad \text{and} \quad -\tan(\alpha)\,|\Re e(h\lambda)| \leq \Im m(h\lambda) \leq \tan(\alpha)\,|\Re e(h\lambda)|$$

where $\alpha \in [0, \pi]$ [16]. Notice that for $\alpha = \pi/2$, $A(\pi/2)$-stability results in the definition of A-stability but for all other $\alpha \in [0, \pi]$ the stability requirements are relaxed.

CHAPTER 3
Review of Integration Methods

This chapter provides a review of most common integration methods as well as recently developed methods that are related to the work presented in this thesis, namely the stable high-order Obreshkov-based integration method. The chapter highlights the characteristics of these methods and their limitations, and hence clarifies the choice of using the Obreshkov-based integration method for steady state simulation as proposed later.

3.1 Linear Multistep Methods

A linear mulistep method (LMS) uses the the values and the derivatives at the previous k steps to advance to the next step at t_{n+k}, it is hence also referred to as a k-step linear method. The general form of a k-step linear method is given by the following formula,

$$\sum_{i=0}^{k} \alpha_i \boldsymbol{x}_{n+k-i} = h \sum_{i=0}^{k} \beta_i \boldsymbol{x}^{(1)}_{n+k-i}, \tag{3.1}$$

where $\boldsymbol{x}^{(1)}_n$ is the approximation of $\dfrac{d}{dt}\boldsymbol{x}(t)$ at $t = t_n$, the coefficients α_i and β_i are specific to the integration method [8], [14].

In order to derive a k-step LMS method with order p the coefficients α_i and β_i $(i = 0, 1 \cdots, k)$ have to be chosen such that the first $p + 1$ Taylor series coefficients of the following operator

$$\boldsymbol{\mathcal{L}}(\boldsymbol{x}(t)) = \sum_{i=0}^{k} \alpha_i \boldsymbol{x}(t_{n+k-i}) - h \sum_{i=0}^{k} \beta_i \boldsymbol{x}'(t_{n+k-i}), \tag{3.2}$$

vanish. An LMS is said to be explicit if approximating the solution $\boldsymbol{x}_{n+k}$ at t_{n+k} does not require the knowledge of the derivative $\boldsymbol{x}_{n+k}^{(1)}$, i.e. if β_0 is set to 0. Otherwise, the method is called implicit. The advantage of an explicit method is that the solution at the next step $\boldsymbol{x}_{n+k}$ at t_{n+k} can be computed directly from the k previous points and their derivatives. However, for an implicit method computing the solution at the next step requires solving a system of nonlinear equations.

3.1.1 Stability of LMS methods

To analyze the stability of LMS method the scalar test problem (2.19) is considered. Hence, $\dot{x} = \lambda x$ is substituted into (3.1) to obtain

$$\sum_{i=0}^{k} (\alpha_i - h\lambda\beta_i) x_{n+k-i} = 0. \tag{3.3}$$

It is shown in [8] that equation (3.3) simplifies to

$$\rho(z) - q\,\sigma(z) = 0, \tag{3.4}$$

where, $q = h\lambda$, $z = e^q$, $\rho(z) = \sum_{i=0}^{k} \alpha_i z^{k-i}$ and $\sigma(z) = \sum_{i=0}^{k} \beta_i z^{k-i}$. The polynomial in (3.4) is of degree k in the variable z, hence it must have k roots call them z_j $(j = 1, 2, \cdots, k)$. It is also shown in [8] that to obtain a stable solution it is required that,

$$|z_j| \leq 1. \tag{3.5}$$

Therefore, it is required that the roots of the polynomial (3.4) be on or inside the unit circle in the z domain, this condition is referred to as the root condition. Consequently, to find the region of absolute stability in the q-domain of an LMS method the roots of polynomial (3.4) have to be found in terms of q. Thus, the region of absolute stability would be all the values of q such that the roots z_j remain inside or on the unity circle in the z-domain. It should be noted that except for simple cases computing the roots z_j in terms of q could be very difficult. In such cases some simplified stability properties could be studied like 0-stability ($h \to 0$) and A_∞-stability ($h \to \infty$).

3.1.2 The Dahlquist Barrier

In 1963, Dahlquist [7] was able to establish important properties regarding the A-stability of LMS methods. He proved that an explicit k-step LMS method cannot be A-stable. Moreover, he also proved that the highest possible order for an implicit A-stable LMS method is 2, i.e. the trapezoidal rule is the highest order A-stable LMS method. The Dalhquist barrier illustrates the trade-off between stability and order.

3.1.3 Examples of LMS methods

Forward Euler (FE)

The Forward Euler method is a first order ($p = 1$) single step method ($k = 1$) obtained by setting $\beta_0 = 0$, hence it's explicit. From equation (3.1) it is found that the expression of FE is given by,

$$\alpha_0 \boldsymbol{x}_{n+1} + \alpha_1 \boldsymbol{x}_n = h\beta_1 \boldsymbol{x}'_n. \tag{3.6}$$

The coefficients α_0, α_1 and β_1 are obtained as outlined in section 3.1 by satisfying the condition set by equation (3.2) ($\alpha_0 = 1, \alpha_1 = -1$ and $\beta_1 = 1$). Hence the FE difference formula is,

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + h\boldsymbol{x}'_n. \tag{3.7}$$

By applying the scalar test equation $x' = \lambda x$ to (3.7) we obtain,

$$x_{n+1} = (1 + h\lambda)x_n. \tag{3.8}$$

The absolute stability condition requires that $|1 + q| \leq 1$ for $\Re e(\lambda) < 0$, where $q = h\lambda \in \mathbb{C}$. Let $q = u + iv$ where $u, v \in \mathbb{R}$, then the absolute stability condition implies $|1 + u + iv| \leq 1 \rightarrow (1 + u)^2 + v^2 \leq 1$. Therefore, the absolute stability region of FE is the unit circle in the q-plane centered at (-1,0) as shown in figure 2–1.

Backward Euler (BE)

BE is similar to FE in that it is a single-step first order method, however, BE is an implicit method i.e. $\beta_0 \neq 0$. It is obtained from equation (3.1) by setting $\beta_1 = 0$. The coefficients of BE are found in the same way as described before for FE to obtain the following formula,

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + h\boldsymbol{x}'_{n+1}. \tag{3.9}$$

When the scalar test problem is applied to BE we find,

$$x_{n+1} = \frac{1}{(1 - h\lambda)}x_n. \tag{3.10}$$

Hence, the condition of absolute stability requires $|1/(1-q)| \leq 1$ for $\Re e(\lambda) < 0$. Hence, $|1/(1-u-iv)| \leq 1 \rightarrow (1-u)^2 + v^2 \geq 1$.Therefore, the absolute stability region of BE is the unit circle in the q-plane centered at (1,0) as shown in figure 2–2. Notice that we also have $\dfrac{x_{n+1}}{x_n} = \dfrac{1}{1-q}\Big|_{q\to\infty} \rightarrow 0$, hence the L-stability requirement is satisfied which means BE is an L-stable method.

Trapezoidal rule (TR)

TR is a second order single step method which is given by,

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \frac{h}{2}(\boldsymbol{x}'_{n+1} + \boldsymbol{x}'_n). \tag{3.11}$$

TR is an implicit method, its region of stability is found by applying the scalar test problem in a similar fashion as seen before. The region of stability of TR is the whole left side of the complex q-plane as seen in figure 2–3, hence its an A-stable method.

Backward Difference Methods (BDF)

The BDF methods are obtained from equation (3.1) by setting $\beta_1 = \beta_2 = \cdots = \beta_k = 0$ to get the following formula,

$$\sum_{i=0}^{k} \alpha_i \boldsymbol{x}_{n+k-i} = h\beta_0 \boldsymbol{x}'_{n+k}. \tag{3.12}$$

The BDF methods are k-step methods with order $p = k$. The BDF method of first order reduces to the BE method and it is the only A- and L-stable BDF method. BDF methods of higher orders are $A(\alpha)$-stable up to 6^{th} order, and they are very well suited to handle stiff systems [11]. Figure 3–1 below shows the absolute stability region for BDF methods upto order 6.

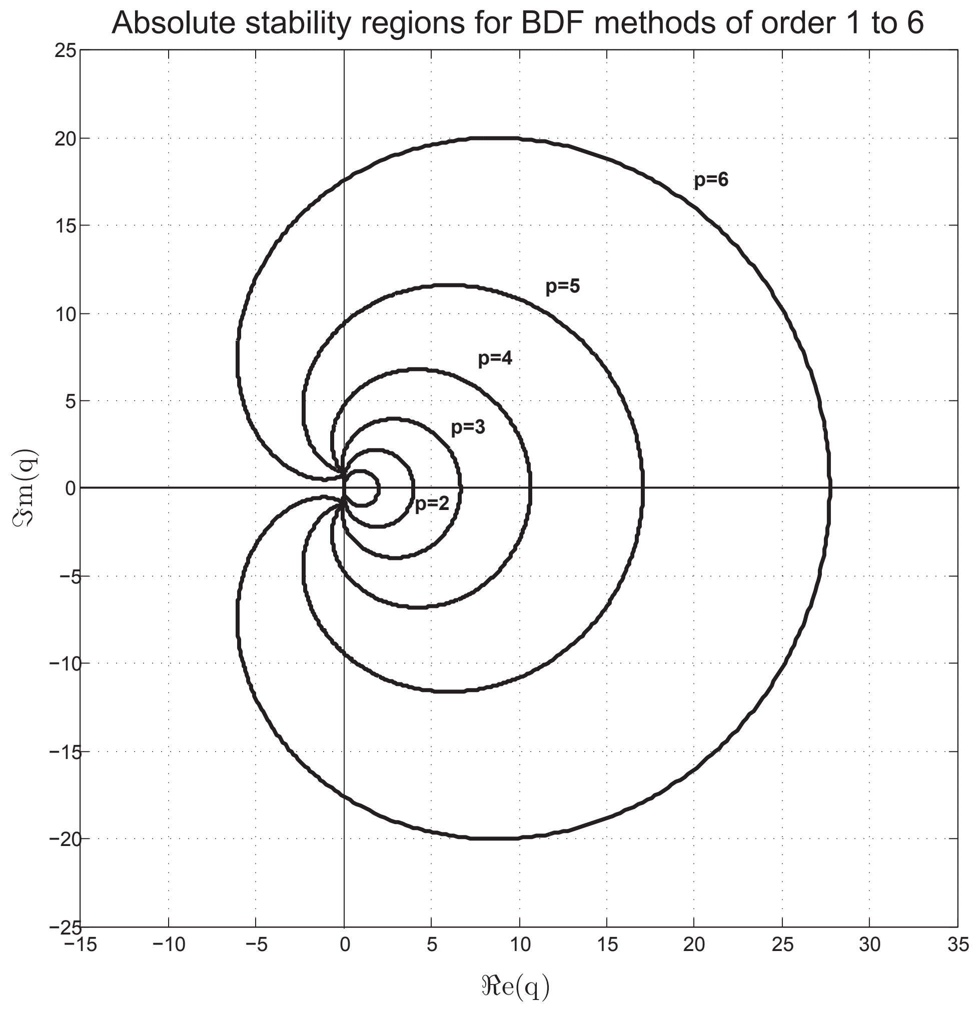

Figure 3–1: Stability region for BDF methods (stability region is exterior to the curves)

3.2 Single-step Multi-stage Methods

Single-step multi-stage methods don't rely on the values of the solution and its derivatives computed at the past points like LMS methods do. Instead, as their name suggests, they introduce extra calculations within a step to compute approximations to the solution and its derivatives at several off-step points. One of the advantages in these methods stems from the fact that they are single step, so they are trivial to implement with a variable step size algorithm as opposed to LMS methods.

The most prominent methods in this category are the Runge-Kutta (RK) methods [14], [16]. An s-stage Runge-Kutta methods is given by the following equations,

$$\begin{aligned} \boldsymbol{X_i} &= \boldsymbol{x_{n-1}} + h\sum_{j=0}^{s} a_{ij}\boldsymbol{f}(t_{n-1}+c_jh, \boldsymbol{X_j}), \quad 1 \le i \le s && (3.13) \\ \boldsymbol{x}_n &= \boldsymbol{x}_{n-1} + h\sum_{i=1}^{s} b_i\boldsymbol{f}(t_{n-1}+c_jh, \boldsymbol{X_i}). && (3.14) \end{aligned}$$

The stage values X_i's are intermediate approximations to the solution at times $t_{n-1}+c_ih$, these stage values are local to the step from t_{n-1} to t_n and the only approximation that the next step sees is $\boldsymbol{x}_n$. The coefficients of the method are chosen such that the values of $\boldsymbol{x}_n$ approaches $\boldsymbol{x}(t_n)$. The Runge-Kutta methods can be represented by a Butcher tableau [16] shown below,

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

or in a more concise form,

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

$\boldsymbol{A} \in \mathbb{R}^{s\times s}$ and $\boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^s$ where s is the number of stages. Unlike LMS methods, RK methods have no theoretical barrier that limits the stability of higher order methods, hence A-stable RK methods with arbitrarily high order are possible. The order of the RK method is limited by the number of stages where $q \leq s$. Still, the difficulty of forming A-stable high order RK is due to the fact that it becomes extremely difficult to form the appropriate Butcher tableau. Moreover, the performance of RK methods can suffer in very stiff systems with moderate step sizes such as those encountered in electronic circuit represented by a systems of differential algebraic equations. In these cases, the effective order of the method falls well below the its nominal order due to the fact that the stage order dominates the error. This has been the main reason that no serious effort has been made to ubiquitously adopt RK methods in circuit simulators and their use is limited for specific problems (such as overcoming the difficulties of BDF in oscillatory problems) [9].

3.3 General Linear Methods

The general linear methods (GLMs) are the generalization of LMS and RK methods. They were developed to circumvent the theoretical limitations of LMS methods while also overcoming the computational complexities in RK methods [17]. Still, they require powerful optimization techniques to construct the method. The general linear methods typically use r past points from previous time steps along with s to approximate the solution at the next step. The output approximations from step number $n-1$ that are available to to compute the next step n are denoted by $x_i^{[n-1]}$, where $i = 1, 2, \cdots, r$, the stage values are given by X_i, where $i = 1, 2, \cdots, s$ and the stage derivatives are given by F_i, where $i = 1, 2, \cdots, s$. Then the computations of the stages and outputs from the next stage n are governed by the following formulae,

$$\begin{aligned} \boldsymbol{X_i} &= \sum_{j=1}^{s} a_{ij} h \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{x}_j^{[n-1]}, \quad i = 1, 2, \cdots, s, && (3.15) \\ \boldsymbol{x}_i^{[n]} &= \sum_{j=1}^{s} b_{ij} h \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{x}_j^{[n-1]} \quad i = 1, 2, \cdots, r. && (3.16) \end{aligned}$$

where the coefficients a_{ij}, u_{ij}, b_{ij} and v_{ij} are the elements of the matrices $\boldsymbol{A} \in \mathbb{R}^{s\times s}$, $\boldsymbol{U} \in \mathbb{R}^{s\times r}$, $\boldsymbol{B} \in \mathbb{R}^{r\times s}$ and $\boldsymbol{V} \in \mathbb{R}^{r\times r}$ respectively. This coefficients are specific to the integration method. To simplify the notation the following vectors are defined,

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X_1} \\ \boldsymbol{X_2} \\ \vdots \\ \boldsymbol{X_s} \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} \boldsymbol{F_1} \\ \boldsymbol{F_2} \\ \vdots \\ \boldsymbol{F_s} \end{bmatrix}, \quad \boldsymbol{x}^{[n-1]} = \begin{bmatrix} \boldsymbol{x}_1^{[n-1]} \\ \boldsymbol{x}_2^{[n-1]} \\ \vdots \\ \boldsymbol{x}_r^{[n-1]} \end{bmatrix}, \text{ and } \boldsymbol{x}^{[n]} = \begin{bmatrix} \boldsymbol{x}_1^{[n]} \\ \boldsymbol{x}_2^{[n]} \\ \vdots \\ \boldsymbol{x}_r^{[n]} \end{bmatrix}.$$

Therefore, equations (3.15) and (3.16) can be rewritten as,

$$\begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{x}^{[n]} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} \otimes \boldsymbol{I}_N & \boldsymbol{U} \otimes \boldsymbol{I}_N \\ \boldsymbol{B} \otimes \boldsymbol{I}_N & \boldsymbol{V} \otimes \boldsymbol{I}_N \end{bmatrix} \begin{bmatrix} h\boldsymbol{F} \\ \boldsymbol{x}^{[n-1]} \end{bmatrix}. \tag{3.17}$$

$\boldsymbol{I}_N$ is the $N \times N$ identity matrix and $\otimes$ denotes the Kronecker product. The stability properties of GLMs are studied by substituting the scalar test problem in equation (3.17) where the stability matrix $\boldsymbol{M}(q) = \boldsymbol{V} + q\boldsymbol{B}(\boldsymbol{I} - q\boldsymbol{A})^{-1}\boldsymbol{U}$ is obtained. The stability function of GLMs is given by, $\boldsymbol{\Phi}(z, q) = \det(z\boldsymbol{I} - \boldsymbol{M}(q)$. A GLM is A-stable if all the z-roots of $\boldsymbol{\Phi}(z, q)$ lie inside the unit disk whenever $\Re e(q) \leq 0$, with only simple roots allowed on the unit circle [16]. In [18] it was proved that for an A-stable GLM with order p the inequality $p \leq 2s$ holds. Moreover the methods attaining the highest order ($p = 2s$) will have an error constant whose lower limit is given by,

$$C \geq (-1)^k \frac{s!s!}{(2s)!(2s+1)!}. \tag{3.18}$$

3.4 Multi Derivative Methods : The Obreshkov Formula

All the methods presented so far had used information either from previous step or stages or both to approximate the solution at the next step. In the 1940s Nikola Obreshkov proposed that information from a third dimension can be used to predict the solution at the next step, namely from using high order derivatives [19]. The Obreshkov methods were left untried due to difficulties involved in handling the high order derivatives [8], [9]. The generalized single step Obreshkov methods are defined by,

$$\sum_{i=0}^{k} \alpha_{i,k}(-1)^i h^i x_{n+1}^{(i)} = \sum_{i=0}^{k} \alpha_{i,k} h^i x_n^{(i)}. \tag{3.19}$$

The coefficients $\alpha_{i,k}$ are computed by substituting x_{n+1} and x_n in (3.19) by $x(t_{n+1})$ and $x(t_n)$ respectively then expanding the resulting equation using Taylor series around the point $t = t_n$, same powers of h are grouped to get,

$$\alpha_{i,k} = \frac{(2k-i)!}{(2k)!} \frac{(k)!}{i!(k-i)!}. \tag{3.20}$$

Obreshkov showed that the order of the method (3.19) is $2k$ i.e. $x_{n+1} = x(t_{n+1}) + \mathcal{O}(h^{2k})$. It was later noted that the coefficients (3.20) are those of the diagonal (k,k)-Padé approximant to the exponential function [20]. Hence, by applying the

Obreshkov formula to the scalar test problem (2.19) the following result is obtained

$$x_{n+1} = \mathcal{R}_{(k,k)}(q)x_n. \tag{3.21}$$

where $q = h\lambda$ and $\mathcal{R}_{(k,k)}(q)$ is the unique k^{th} diagonal approximant to the exponential function. Hence, for the Obreshkov method to be A-stable it is required that $|\mathcal{R}_{(k,k)}(q)| \leq 1$ for $\Re e(\lambda) < 0$. It was shown by Birkhoff and Varga [21] that, in fact, this condition is satisfied by all diagonal Padé approximants to the exponential function. Therefore the Obreshkov formula is A-stable. The Obreshkov method is not L-stable since $\lim_{\lambda \to -\infty} |\mathcal{R}_{(k,k)}(h\lambda)| = 1$ instead of 0 as required according to equation (2.22). Note that the Obreshkov based methods can be arbitrary high in order with out losing the A-stability property.

3.4.1 The Modified Obreshkov Formula

Recently, Gad et al. presented a criterion to formulate an L-stable Obreshkov method [8], [9]. They proved that it is possible to achieve L-stability by modifying the Obreshkov formula in the following fashion,

$$\sum_{i=0}^{k} \alpha_i h^i x_{n+1}^{(i)} = \sum_{i=0}^{l} \beta_i h^i x_n^{(i)}. \tag{3.22}$$

where α_i and β_i are the coefficients of the (l,k)-Padé approximant given by

$$\begin{aligned} \alpha_i &= (-1)^i \frac{(l+k-i)!}{(l+k)!} \frac{(k)!}{i!(k-i)!} \qquad i = 0, \cdots, k \\ \beta_i &= \frac{(l+k-i)!}{(l+k)!} \frac{(l)!}{i!(k-i)!} \qquad i = 0, \cdots, l. \end{aligned}$$

It was shown in [9] that the modified Obreshkov method is A-stable if and only if $l \in \{k-2, k-1, k\}$, and that is is L-stable if and only if $l \in \{k-2, k-1\}$. Notice that for

$l = k$ the modified Obreshkov formula (3.22) reduces to the original formula (3.19), thus the Obreshkov formula is a special case of the modified Obreshkov formula. The order of the modified Obreshkov formula is $k + l$ and error constant is given by,

$$C_{l,k} = (-1)^k \frac{l!k!}{(l+k)!(l+k+1)!}. \tag{3.23}$$

Note that the error constant of the Obreshkov formula $C_{k,k}$ is identical to the lowest possible error constant achieved by a GLM or an RK method with k-implicit stages (see equation (3.18)). Hence, the Obreshkov method obtains the most accurate results than any existing single-step or multi-step methods with equal number of unknowns [9]. In [9] and [8] the idea of generalized multi-step Obreshkov formulae is explored as well. It was found that although A-stable and possibly L-stable multi-step Obreshkov methods can be formulated, these methods wouldn't provide an advantage in terms of order achievable compared to the single step method. Moreover, it was proven that A-stable multi-step Obreshkov methods wouldn't provide any improvement in terms of accuracy over the single step Obreshkov method either, since their minimum error constant is identical to that of the A-stable Obreshkov formula i.e. $C_{k,k}$ (for $k = l$). Figure 3–2 below shows the stability regions for Obreshkov methods of orders 1, 3, 5 and 7, all these methods are L-stable.

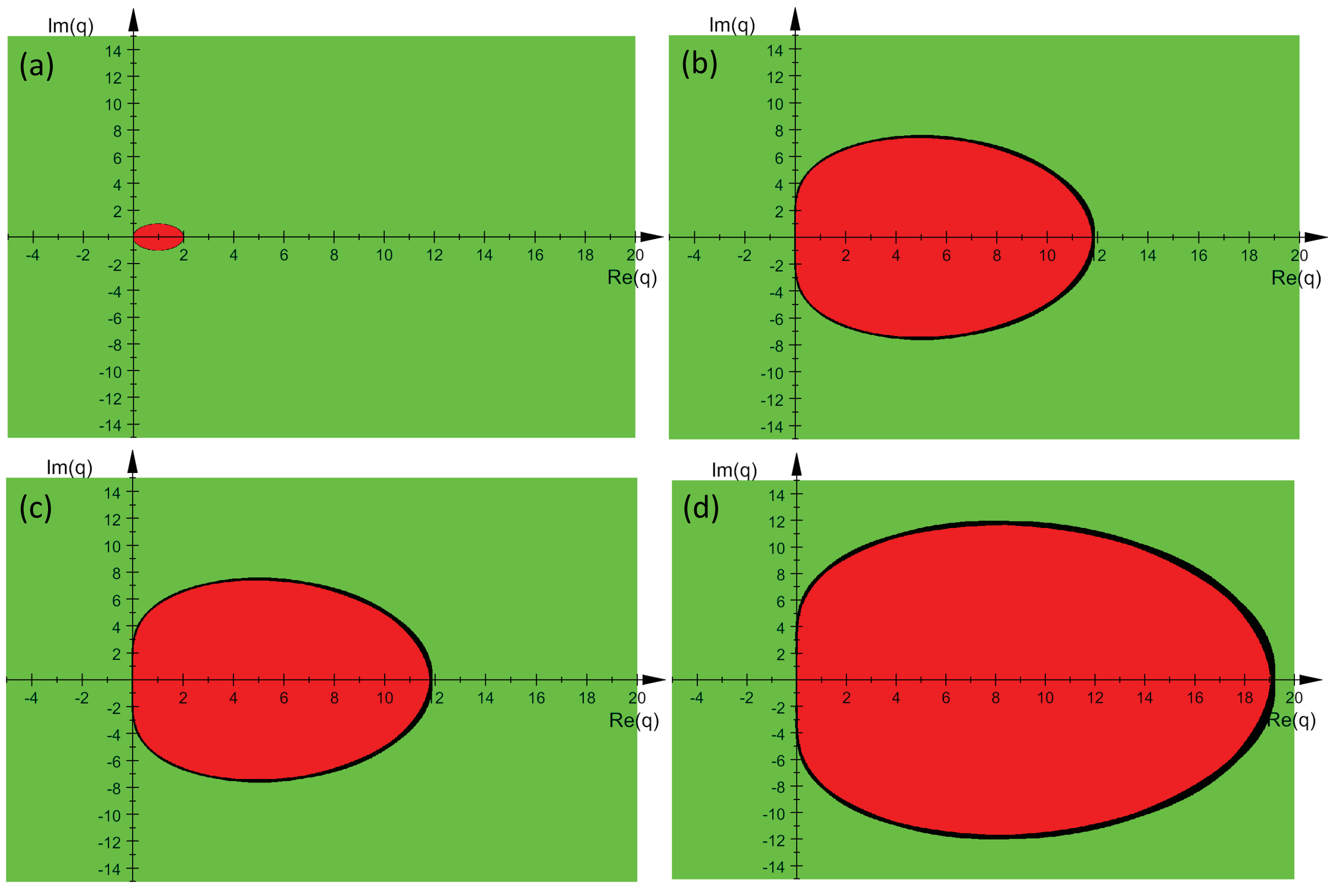

Figure 3–2: Absolute stability regions for different Obreshkov methods shaded in green: (a) Obreshkov 1^{st} order (BE), (b) Obreshkov 3^{rd} order, (c) Obreshkov 5^{th} order, (d) Obreshkov 7^{th} order.

3.4.2 Application to Circuits

The dynamics of a circuit are represented by the general modified nodal analysis (MNA) equation as follows [22],

$$\boldsymbol{G}\boldsymbol{x}(t) + \boldsymbol{C}\dot{\boldsymbol{x}}(t) + \boldsymbol{f}(\boldsymbol{x}(t)) = \boldsymbol{b}(t), \tag{3.24}$$

where $\boldsymbol{x}(t)$ is an $N \times 1$ vector of unknown nodal voltages and device currents, $\boldsymbol{G}$ and $\boldsymbol{C}$ are $N \times N$ matrices of memoryless elements (resistors) and memory elements (capacitors, inductors) respectively, $\boldsymbol{f}(\boldsymbol{x}(t))$ is an $N \times 1$ vector representing the nonlinear elements in the circuit (diodes, transistors, ...) and finally $\boldsymbol{b}(t)$ is an $N \times 1$ representing the forcing functions. To be able to solve the system of differential equation (3.24) numerically it is discretized in time, where starting from a known solution at time t_n a step h is made to get to the solution at time t_{n+1}. The discretized system is represented as follows,

$$\boldsymbol{G}\boldsymbol{x}_{n+1} + \boldsymbol{C}\boldsymbol{x}_{n+1}^{(1)} + \boldsymbol{f}_{n+1} = \boldsymbol{b}_{n+1}, \tag{3.25}$$

where $\boldsymbol{x}_{n+1}$ and $\boldsymbol{x}_{n+1}^{(1)}$ are the approximations of $\boldsymbol{x}(t_{n+1})$ and $\left.\frac{d}{dt}\boldsymbol{x}(t)\right|_{t=t_{n+1}}$ respectively, $\boldsymbol{f}_{n+1}$ is $\boldsymbol{f}(\boldsymbol{x}_{n+1})$ and $\boldsymbol{b}_{n+1}$ is $\boldsymbol{b}(t_{n+1})$. Discretizing the system (3.24) using the modified Obreshkov formula is outlined in [8] and [9] by discretizing (3.24) and its derivatives in the following fashion,

$$\begin{aligned}
\boldsymbol{G}\boldsymbol{x}_{n+1} + \frac{1}{h}\boldsymbol{C}(h\boldsymbol{x}_{n+1}^{(1)}) + \boldsymbol{f}_{n+1} &= \boldsymbol{b}_{n+1} \quad \text{rewriting (3.25)} \\
\boldsymbol{G}(h\boldsymbol{x}_{n+1}^{(1)}) + \frac{1}{h}\boldsymbol{C}(h^2\boldsymbol{x}_{n+1}^{(2)}) + h\boldsymbol{f}_{n+1}^{(1)} &= h\boldsymbol{b}_{n+1}^{(1)} \\
\boldsymbol{G}(h^2\boldsymbol{x}_{n+1}^{(2)}) + \frac{1}{h}\boldsymbol{C}(h^3\boldsymbol{x}_{n+1}^{(3)}) + h^2\boldsymbol{f}_{n+1}^{(2)} &= h^2\boldsymbol{b}_{n+1}^{(2)} \\
&\vdots \\
\boldsymbol{G}(h^{k-1}\boldsymbol{x}_{n+1}^{(k-1)}) + \frac{1}{h}\boldsymbol{C}(h^k\boldsymbol{x}_{n+1}^{(k)}) + h^{k-1}\boldsymbol{f}_{n+1}^{(k-1)} &= h^{k-1}\boldsymbol{b}_{n+1}^{(k-1)} \\
\alpha_0\boldsymbol{x}_{n+1} + \alpha_1(h\boldsymbol{x}_{n+1}^{(1)}) + \cdots + \alpha_k(h^k\boldsymbol{x}_{n+1}^{(k)}) &= \sum_{i=0}^{l}\beta_i h^i \boldsymbol{x}_n^{(i)}
\end{aligned}$$

The last equation is the modified Obreshkov formula (3.22). The discretized equations shown above can be represented in the augmented system,

$$\begin{bmatrix} \boldsymbol{G} & \boldsymbol{C}/h & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{G} & \boldsymbol{C}/h & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{G} & \boldsymbol{C}/h \\ \alpha_0 \boldsymbol{I} & \alpha_1 \boldsymbol{I} & \cdots & \alpha_{k-1}\boldsymbol{I} & \alpha_k \boldsymbol{I} \end{bmatrix} \underbrace{\begin{bmatrix} \boldsymbol{x}_{n+1} \\ h\boldsymbol{x}_{n+1}^{(1)} \\ \vdots \\ h^{k-1}\boldsymbol{x}_{n+1}^{(k-1)} \\ h^k \boldsymbol{x}_{n+1}^{(k)} \end{bmatrix}}_{\boldsymbol{\xi_{n+1}}} + \underbrace{\begin{bmatrix} \boldsymbol{f}_{n+1} \\ h\boldsymbol{f}_{n+1}^{(1)} \\ \vdots \\ h^{k-1}\boldsymbol{f}_{n+1}^{(k-1)} \\ \boldsymbol{0} \end{bmatrix}}_{\tilde{\boldsymbol{\rho}}_{n+1}} = \underbrace{\begin{bmatrix} \boldsymbol{b}_{n+1} \\ h\boldsymbol{b}_{n+1}^{(1)} \\ \vdots \\ h^{k-1}\boldsymbol{b}_{n+1}^{(k-1)} \\ \sum_{i=0}^{l} \beta_i h^i \boldsymbol{x}_n^{(i)} \end{bmatrix}}_{\tilde{\boldsymbol{b}}_{n+1}} \tag{3.26}$$

where $\boldsymbol{x}_{n+1}^{(k)}$ is the approximation of $\frac{d^k}{dt^k}\boldsymbol{x}(t)\big|_{t=t_{n+1}}$, $\boldsymbol{f}_{n+1}^{(k)} = \frac{d^k}{dt^k}\boldsymbol{f}(\boldsymbol{x}(t))\big|_{t=t_{n+1}}$, $\boldsymbol{b}_{n+1}^{(k)} = \frac{d^k}{dt^k}\boldsymbol{b}(t)\big|_{t=t_{n+1}}$ and $\boldsymbol{I}$ is an $N \times N$ identity matrix. The system (3.26) can be rewritten as,

$$(\tilde{\boldsymbol{G}} + \tilde{\boldsymbol{C}})\boldsymbol{\xi}_{n+1} + \tilde{\boldsymbol{\rho}}_{n+1} = \tilde{\boldsymbol{b}}_{n+1}. \tag{3.27}$$

The matrices $\tilde{\boldsymbol{G}}$ and $\tilde{\boldsymbol{C}} \in \mathbb{R}^{(k+1)N \times (k+1)N}$ and are given by,

$$\tilde{\boldsymbol{G}} = \begin{bmatrix} \boldsymbol{G} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{G} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{G} & \boldsymbol{0} \\ \alpha_0 \boldsymbol{I} & \alpha_1 \boldsymbol{I} & \cdots & & \alpha_k \boldsymbol{I} \end{bmatrix} \text{ and } \tilde{\boldsymbol{C}} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{C}/h & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{C}/h & \dots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{C}/h \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & & \boldsymbol{0} \end{bmatrix}. \tag{3.28}$$

The vectors $\boldsymbol{\xi}_{n+1}$, $\tilde{\boldsymbol{\rho}}_{n+1}$ and $\tilde{\boldsymbol{b}}_{n+1} \in \mathbb{R}^{(k+1)N\times 1}$ are the derivative-augmented unknowns vector, nonlinear vector and independent sources vector respectively as shown in (3.26).

Notice that for linear equations the nonlinear vector $\tilde{\boldsymbol{\rho}}_{n+1} = \boldsymbol{0}$, hence, (3.27) reduces to,

$$(\tilde{\boldsymbol{G}} + \tilde{\boldsymbol{C}})\boldsymbol{\xi}_{n+1} = \tilde{\boldsymbol{b}}_{n+1}. \tag{3.29}$$

The linear system (3.29) can be solved directly by direct inversion,

$$\boldsymbol{\xi}_{n+1} = (\tilde{\boldsymbol{G}} + \tilde{\boldsymbol{C}})^{-1}\tilde{\boldsymbol{b}}_{n+1}.$$

For the more general nonlinear case Newton iteration is necessary. Since the Obreshkov methods are A- or L-stable with arbitrarily high order it is expected that the increase in computation cost due to solving a bigger system will be outweighed by the ability to utilize big step sizes without compromising accuracy.

3.4.3 Difficulties in Modified Obreshkov Methods

There are a number of difficulties in integration methods based on the modified Obreshkov formula, mostly related to the presence of the high order derivative. These difficulties however were adequately addressed in [8], [9] and [10].

The first difficulty arises from the need to have at least l high oreder dervatives at $t = 0$, these derivatives usually aren't readily available and need to be calculates. In [8], [9] two methods to deal with this problem were formulated. The first, derived a recursive formula relating the $(k+1)$th derivative to the kth derivative. Using the recursive formula, theoretically, one should be able to compute the high order derivatives starting from the zeroth derivative (initial conditions). However, in practice this method is problematic because it introduces numerical errors that grow exponentially with the order of the derivative. Hence, another more robust method was suggested. The second method is based on the fact that the modified Obreshkov methods needs l derivatives at t_n to approximate k derivatives at t_{n+1} with $k-2 \leq l \leq k$. Hence it is possible to start the integration at low order; 1st order ($l=0, k=1$), for example, where no derivatives are required at t_0 and then step in time which will generate k derivatives (in this case the first derivative) automatically. The order of the method is then ramped gradually till the desired order is achieved. The order ramping is done at small step sizes at first, later when the order of the method increases the step size is increased without compromising accuracy. This method is summarized in a ramping algorithm detailed in [8].

A second difficulty arises when dealing with nonlinear systems where it's necessary to find the high order derivatives $\boldsymbol{f}_{n+1}^{(i)}$ (for $i = 1, \cdots, k-1$) in order to obtain the augmented nonlinear vector $\tilde{\boldsymbol{\rho}}_{\boldsymbol{n+1}}$ given a trial solution $\boldsymbol{\xi}_{n+1}$. Moreover, the Jacobian matrix of the augmented system $\dfrac{\partial \tilde{\boldsymbol{\rho}}_{n+1}}{\partial \boldsymbol{\xi}_{n+1}}$ needs to be computed as well. In [8], [9], a systematic efficient approach that represents the nonlinear functions in a rooted-tree structure is proposed to numerically carry out these computations while being compatible for integration with a circuit simulator. The advantage of the rooted-tree representation is that it allows for fast computation of the augmented nonlinear vector and the augmented Jacobian matrix while adding little computational cost since the tree structure for each nonlinear function has to be formed once off-line and saved for use during the integration process.

Furthermore, in [10] several optimization techniques were presented aimed at improving the Obreshkov-based high order methods formulated in [8], [9]. It was shown that the size od the augmented system can be shrunk from $(k+1)N \times (k+1)N$ to $kN \times kN$ while preserving the sparsity pattern and the lower Hessenberg structure of the augmented Jacobian matrix. Moreover, a more efficient block LU factorization algorithm is presented that takes advantage of the sparsity pattern of the augmented Jacobian matrix to speed up the integration process. Finally, a method to reduce the cost of Newton-Raphson iteration is suggested by using a low order method (such as trapezoidal method) to predict the approximated solution at the next step then correct the prediction using Obreshkov-based method. This was shown to speed up the convergence of Newton-Raphson iteration and hence improve the performance.

CHAPTER 4
Practical Considerations in the Implementation of Obreshkov-Based Methods

This chapter serves to aid anyone interested either in recreating the work presented in this thesis as well as the relevant literature. After having gone though the process of implementing a working Obreshkov based integration method, a lot of insight was gained on the hurdles that might delay or setback a first-time implementation of an Obreshkov-based numerical integration method. Therefore, in this chapter we seek to share with the reader this gained insight in the hope to provide a deeper understanding of the underlying issues and the limitations of certain techniques that are proposed in the literature. This chapter does not offer any fundamentally novel content and the literature cited [8, 9, 10] cover most of the points that will be discussed here and thus it could be skipped without loss of continuity.

4.1 Algorithm Initialization

In an Obreshkov-based method the solution and l high order derivatives from the previous step are used to approximate the solution and k of its derivatives at the next step. An obvious issue arises at the beginning of the integration process where usually the only information available is the solution at time $t = 0$ or in circuit simulation context the DC-solution of the circuit. Hence, a method had to be devised to compute need l high-order derivatives to be able to proceed with the integration process. In [8, 9], two methods are proposed, we will discuss each of them and provide a general guide on how they can be implemented.

4.1.1 Computing the High-Order Derivatives Using the Recursive Formula

The first method for finding the required high order derivatives is based on a general recursive formula that can be obtained from the discretized MNA equation (3.25) by considering its i^{th} derivative at time $t = 0$,

$$\boldsymbol{G}(h^{i-1}\boldsymbol{x}_0^{(i-1)}) + \frac{1}{h}\boldsymbol{C}(h^i\boldsymbol{x}_0^{(i)}) + h^{i-1}\boldsymbol{f}_0^{(i-1)} \quad = \quad h^{i-1}\boldsymbol{b}_0^{(i-1)}. \tag{4.1}$$

Thus, assuming $\boldsymbol{C}$ is invertible we can write,

$$h^i\boldsymbol{x}_0^{(i)} \quad = \quad h\boldsymbol{C}^{-1}(h^{i-1}\boldsymbol{b}_0^{(i-1)} - \boldsymbol{G}(h^{i-1}\boldsymbol{x}_0^{(i-1)}) - h^{i-1}\boldsymbol{f}_0^{(i-1)}). \tag{4.2}$$

Using equation (4.2) it is possible to build up the augmented unkown vector $\boldsymbol{\xi_0}$ up to the desired order. However, if the matrix $\boldsymbol{C}$ is not invertible it will not be possible to retrieve the high order derivatives using equation (4.2). Instead, the system can be partitioned into differntiial and algebraic set of equations given by,

$$\begin{bmatrix} \boldsymbol{C_{11}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} h^i\boldsymbol{x}_{0,U}^{(i)} \\ h^i\boldsymbol{x}_{0,L}^{(i)} \end{bmatrix} = h \begin{bmatrix} h^{i-1}\boldsymbol{b}_{0,U}^{(i-1)} \\ h^{i-1}\boldsymbol{b}_{0,L}^{(i-1)} \end{bmatrix} - \\ h \begin{bmatrix} \boldsymbol{G_{11}} & \boldsymbol{G_{12}} \\ \boldsymbol{G_{21}} & \boldsymbol{G_{22}} \end{bmatrix} \begin{bmatrix} h^{i-1}\boldsymbol{x}_{0,U}^{(i-1)} \\ h^{i-1}\boldsymbol{x}_{0,L}^{(i-1)} \end{bmatrix} - h \begin{bmatrix} h^{i-1}\boldsymbol{f}_{0,U}^{(i-1)} \\ h^{i-1}\boldsymbol{f}_{0,L}^{(i-1)} \end{bmatrix}, \tag{4.3}$$

such that $\boldsymbol{C_{11}}$ and $\boldsymbol{G_{22}}$ are invertible. Thus, $h^i\boldsymbol{x}_{0,U}^{(i)}$ can be obtained from the upper part of (4.3) by inverting $\boldsymbol{C_{11}}$ as shown below, since it is dependent on the already availabe lower order derivatives,

$$h^i \boldsymbol{x}_{0,U}^{(i)} = h\boldsymbol{C}_{\mathbf{11}}^{\mathbf{-1}}(h^{i-1}\boldsymbol{b}_{\boldsymbol{0,U}}^{\boldsymbol{(i-1)}} - \boldsymbol{G}_{\mathbf{11}}(h^{i-1}\boldsymbol{x}_{0,U}^{(i-1)} - \boldsymbol{G}_{\mathbf{12}}(h^{i-1}\boldsymbol{x}_{0,L}^{(i-1)}) - h^{i-1}\boldsymbol{f}_{0,U}^{(i-1)}. \quad (4.4)$$

From the lower part of (4.3) a nonlinear algebraic equation to compute $h^i\boldsymbol{x}_{0,L}^{(i)}$ can be written as follows,

$$\boldsymbol{G}_{\mathbf{22}}(h^i\boldsymbol{x}_{0,L}^{(i)}) + h^i\boldsymbol{f}_{0,L}^{(i)} = h^i\boldsymbol{b}_{0,L}^{(i)} - \boldsymbol{G}_{\mathbf{21}}(h^i\boldsymbol{x}_{0,U}^{(i)}), \quad (4.5)$$

where $h^i\boldsymbol{x}_{0,U}^{(i)}$ has already been computed from (4.4). The main difficulty in equation (4.5) is that finding the $\boldsymbol{x}_{0,L}^{(i)}$ requires knowing $\boldsymbol{f}_{0,L}^{(i)}$ which is not readily available. In [8], it is shown that $\boldsymbol{f}_{n+1}^{(i)}$ can be expressed as a sum of a product of the block entries of the Jacobian matrix $\partial\boldsymbol{\rho}/\partial\boldsymbol{\xi}\Big|_{t=t_{n+1}}$ and the derivatives of $\boldsymbol{x}_{n+1}$ which in turn are the block entries of the augmented vector $\boldsymbol{\xi}_{n+1}$ as follows,

$$\begin{aligned} h^i\boldsymbol{f}_{n+1}^{(i)} &= h^i\sum_{j=0}^{i-1}\binom{i-1}{j}\boldsymbol{R}_{n+1}^{(i-1-j)}\boldsymbol{x}_{n+1}^{(j+1)} \\ &= \boldsymbol{R}_{n+1}^{(0)}h^i\boldsymbol{x}_{n+1}^{(i)} + h^i\sum_{j=0}^{i-2}\binom{i-1}{j}\boldsymbol{R}_{n+1}^{(i-1-j)}\boldsymbol{x}_{n+1}^{(j+1)}, \end{aligned} \quad (4.6)$$

where $\boldsymbol{R}_{n+1}^{(u)} = \dfrac{\partial^u}{\partial t^u}\left(\dfrac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}}\right)$. Therefore, equation (4.5) can be simplified to the following,

$$(\boldsymbol{G}_{\mathbf{22}} + \tilde{\boldsymbol{J}}_{\boldsymbol{0,L}})(h^i\boldsymbol{x}_{0,L}^{(i)}) = h^i\boldsymbol{b}_{0,L}^{(i)} - \boldsymbol{G}_{\mathbf{21}}(h^i\boldsymbol{x}_{0,U}^{(i)}) - \boldsymbol{\gamma}_{0,L}, \quad (4.7)$$

where $\tilde{\boldsymbol{J}}_{\boldsymbol{0,L}} = \partial\boldsymbol{f}_{0,L}/\partial\boldsymbol{x}_{0,L}$ and $\boldsymbol{\gamma}_{0,L} = h^i\sum_{j=0}^{i-2}\binom{i-1}{j}\boldsymbol{R}_{0,L}^{(i-1-j)}\boldsymbol{x}_{0,L}^{(j+1)}$ which does not depend on $\boldsymbol{x}^{(i)}$.

To partition the system (4.2) into its invertible and non-invertible parts, we formulated a heuristic approach which yields the required structure of the system. We start by applying singular value decomposition to the singular $\boldsymbol{C}$ matrix to get,

$$\boldsymbol{C} = \boldsymbol{U}\boldsymbol{C}_{new}\boldsymbol{V}, \tag{4.8}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are invertible matrices and $\boldsymbol{C}_{new}$ has the required structure shown in (4.3). By left multiplying (4.1) by $\boldsymbol{U}^{-1}$ and using the fact that the identity matrix $\boldsymbol{I} = \boldsymbol{V}^{-1}\boldsymbol{V}$ it follows that,

$$\begin{aligned}\boldsymbol{U}^{-1}\boldsymbol{G}\boldsymbol{V}^{-1}(h^{i-1}\boldsymbol{V}\boldsymbol{x}_0^{(i-1)}) + \frac{1}{h}\boldsymbol{U}^{-1}\boldsymbol{C}\boldsymbol{V}^{-1}(h^{i}\boldsymbol{V}\boldsymbol{x}_0^{(i)}) + h^{i-1}\boldsymbol{U}^{-1}\boldsymbol{f}_0^{(i-1)} &= h^{i-1}\boldsymbol{U}^{-1}\boldsymbol{b}_0^{(i-1)} \\ \boldsymbol{G}_{new}(h^{i-1}\boldsymbol{x}_{0,new}^{(i-1)}) + \frac{1}{h}\boldsymbol{C}_{new}(h^{i}\boldsymbol{x}_{0,new}^{(i)}) + h^{i-1}\boldsymbol{f}_{0,new}^{(i-1)} &= h^{i-1}\boldsymbol{b}_{0,new}^{(i-1)}\,,\end{aligned} \tag{4.9}$$

where $\boldsymbol{G}_{new} = \boldsymbol{U}^{-1}\boldsymbol{G}\boldsymbol{V}^{-1}$, $\boldsymbol{C}_{new} = \boldsymbol{U}^{-1}\boldsymbol{C}\boldsymbol{V}^{-1}$, $\boldsymbol{x}_{0,new} = \boldsymbol{V}\boldsymbol{x}_0$, $\boldsymbol{f}_{0,new} = \boldsymbol{U}^{-1}\boldsymbol{f}_0$ and $\boldsymbol{b}_{0,new}^{(i-1)} = \boldsymbol{U}^{-1}\boldsymbol{b}_0$. The resulting new system (4.9) has the partitioned structure shown in (4.3), the upper and lower parts of the $\boldsymbol{x}_{0,new}^{(i)}$ can then be calculated using (4.4) and (4.7). The required solution for the original system is then retrieved simply by using $\boldsymbol{x}_0^{(i)} = \boldsymbol{V}^{-1}\boldsymbol{x}_{0,new}^{(i)}$.

4.1.2 Computing the High-Order Derivatives Using Order Ramping

An alternative way to compute the high-order derivatives would be to use the inherent property of the L-stable Obreshkov formula where only l derivatives are needed at the n^{th} step to generate k derivatives at the $(n+1)^{th}$ step, where $k-2 \leq l \leq k-1$. Therefore, for a method of an order $p = k+l$ in order to obtain the respective augmented initial vector of the integration is started from the DC solution at $t = 0$ with a low order method say $k = 2$ and $l = 0$. Since $l = 0$ this means that the only necessary information at $t = t_0$ is the DC solution $\boldsymbol{x}_0$, at the next step the first two derivatives are now available hence a higher order method can be used, say $k = 4$ and $l = 2$, to obtain four derivatives at the third step. This order ramping process can be continued until the desired order of integration is reached. It is important to note that since the lower order methods have a higher local truncation error than the higher order methods a very small step size must be used during the ramping process. The step size can then be increased gradually as the order is increased.

After implementing both methods, it was found that calculating the high order derivatives using the recursive formulas introduces huge numerical errors to the derivatives due to the inversion of the $\boldsymbol{C}$ or $\boldsymbol{C}_{11}$ matrix. The numerical error could be small for the first or second derivative but it gets exponentially amplified for increasingly high-order derivatives. The amplification factor gets bigger in the presence of parasitic elements with small capacitances or inductances. Therefore, The first one or two derivatives can be calculated using the recursive formula but computing the higher order derivatives has to be done using the order ramping process while choosing a small step size for lower order steps to keep the local truncation error small. As a starting point, we suggest to focus on implementing the initialization algorithm using the order ramping process.

4.2 Error Estimation

In [8, 9] an error estimation method is proposed to control the step size in order to keep the error below a specified threshold. After implementing the high-order Obreshkov based integration method with the error control mechanism it was found that the method used to estimate the error might exhibit some limitations. This section sheds light on the possible source of these shortcomings, for details about the theoretical derivations and implementation of the error estimation method refer to [8].

The proposed error estimation technique for a numerical integration method of order p relies on estimating the h-scaled p^{th} derivative at two consecutive time steps. Then, the $(p+1)^{th}$ is estimated as the difference of the two p^{th} derivatives divided by the step size. The local truncation error which is proportional to the $(p+1)^{th}$ derivative is then calculated using by multiplying the $(p+1)^{th}$ derivative by a scaling coefficient specific to the integration method being used called the error constant. Therefore, the accuracy of the estimated error is dependent on the accuracy of the estimated p^{th} derivative.

To evaluate the accuracy of error estimation process, the h-scaled p^{th} derivative of a known sinusoidal function was computed using the proposed method and then compared to the exact value at a specified time instant t_n. The sinusoidal function was chosen to have a frequency of 1MHz so that it represents a practical time varying function in the context of circuit simulation. The step size was varied and the value of the h-scaled p^{th} derivative at an arbitrarily chosen time $t = 2s$ was plotted as a function of the step size h using a log-log scale as shown in figure 4–1.

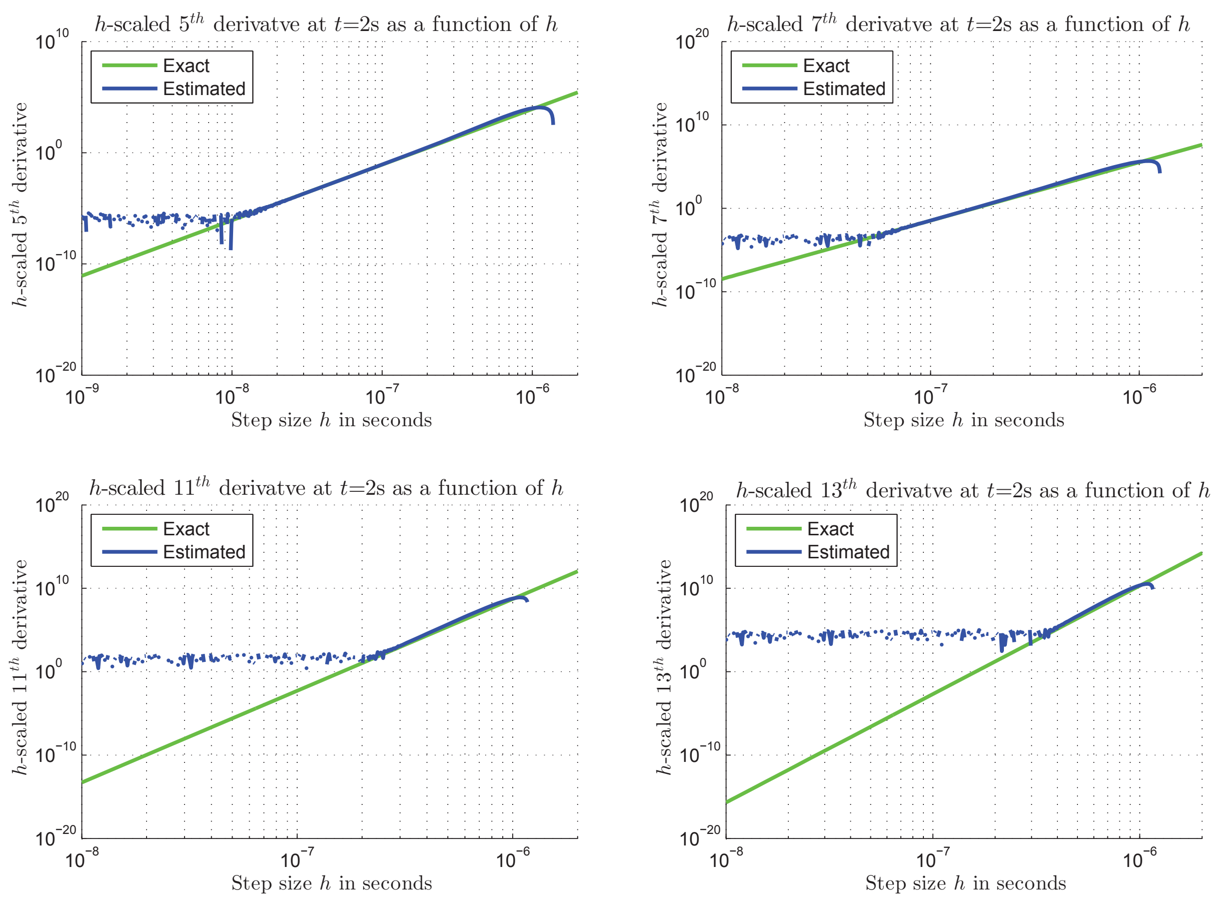

Figure 4–1: The variation of the h-scaled derivative of the function $y(t) = \sin(2\pi \times 10^6 t)$ at $t_n = 2s$ as a function of the step size h.

Since the data is being plotted on a log-log scale the negative values of the h-scaled derivatives are ignored. It can be seen from figure 4–1, that the estimated value of the p^{th} derivative matches the exact value for a certain range step sizes. It is also noted that as the order of the derivative being estimated increases the range of step sizes that allow accurate estimation of the higher order derivative shrinks. For example for the 5^{th} order derivative that range appear to be $h \in [10^{-8}, 10^{-6}]$, while for the 13^{th} order derivative that range shrinks to about $h \in [4 \times 10^{-7}, 10^{-6}]$. Moreover, it seems to be that the largest possible step size that allows for an accurate estimation of the high order derivative has a value of about one period, in this case $h = 10^{-6}$. Consequently, the proposed error estimation method will provide a good measure of the error but only for a certain range of step sizes at each order. It is important to note the limitations that this method acquires when it is being used.

CHAPTER 5
High-Order Steady-State Simulation

In this chapter, a detailed derivation of the high-order Shooting method is presented. The method is a Shooting-Newton algorithm based on the A- and L-stable Obreshkov-based integration methods. This chapter is divided into two sections, the first offers a review of the construction of the Shooting-Newton method. The second details the derivation of the high-order Shooting-Newton method.

5.1 The Shooting Method

Shooting methods are designed to solve boundary value problems (see section 2.2), this section describes the general theoretical framework of the shooting methods. Consider the following system of differential equation,

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), t), \tag{5.1a}$$

subject to the most general linear two-point boundary conditions

$$A\boldsymbol{x}(0) + B\boldsymbol{x}(T) = c\,. \tag{5.1b}$$

The goal of the shooting method is to solve for $\boldsymbol{x}_0$ that satisfies equation (5.1b) given that $\boldsymbol{x}(t)$ is subject of the dynamics defined by equation (5.1a). We can rewrite equation (5.1b) as the following variational formula for $\boldsymbol{x}_0$,

$$\boldsymbol{\Psi}(\boldsymbol{x}_0) = A\boldsymbol{x}(0) + B\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T) - c = 0\,, \tag{5.2}$$

where $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T)$ is the transition function of $\boldsymbol{x}(t)$ that defines its state between $t = t_0$ and $t = T$ starting from $\boldsymbol{x}(0) = \boldsymbol{x}_0$. In other words, $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T)$ is the solution of the IVP given by equation (5.1a) with initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0$. The nonlinear variational equation in $\boldsymbol{x}_0$ (5.2) can be solved using Newton-Raphson, which necessitates finding $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T)$ by solving the IVP (5.1a) between from $t = 0$ to $t = T$, as well as computing the following Jacobian matrix,

$$\frac{d\boldsymbol{\Psi}(\boldsymbol{x}_0)}{d\boldsymbol{x}_0} = A + B\frac{d\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T)}{d\boldsymbol{x}_0} \,. \tag{5.3}$$

The Jacobian in (5.3) is computed by deriving equation (5.1a) with respect to the initial condition $\boldsymbol{x}_0$ to get,

$$\underbrace{\frac{d}{dt}\left(\frac{d\boldsymbol{x}}{d\boldsymbol{x}_0}\right)}_{\dot{\boldsymbol{z}}} = \underbrace{\frac{d\boldsymbol{f}(\boldsymbol{x}, t)}{d\boldsymbol{x}}}_{\boldsymbol{J}(t)}\underbrace{\frac{d\boldsymbol{x}}{d\boldsymbol{x}_0}}_{\boldsymbol{z}}, \tag{5.4}$$

where $\boldsymbol{z}(t)$ is known as the sensitivity matrix and $\boldsymbol{J}(t)$ is the Jacobian matrix generated during the integration process of equation (5.1a). Notice that the equation (5.4) is a linear differential equation and hence its solution can be obtained at a given time t directly after $\boldsymbol{J}(t)$ had been computed when solving equation (5.1a) at the same time t. At time $t = T$, we will have $\boldsymbol{z}(T) = \dfrac{d\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T)}{d\boldsymbol{x}_0}$ and hence the Jacobian matrix in equation (5.8) van be easily computed. Therefore, by integrating the IVP (5.1a) from $t = 0$ to $t = T$ all the information needed to update $\boldsymbol{x_0}$ using Newton-Rapshon will be available,

$$\boldsymbol{x}_0^{new} = \boldsymbol{x}_0^{old} - \underbrace{\left.\frac{d\boldsymbol{\Psi}(\boldsymbol{x}_0)}{d\boldsymbol{x}_0}^{-1}\right|_{\boldsymbol{x}_0 = \boldsymbol{x}_0^{guess}} \boldsymbol{\Psi}(\boldsymbol{x}_0^{guess})}_{\Delta\boldsymbol{x}_0} \,. \tag{5.5}$$

After $\boldsymbol{x}_0$ is updated, the whole process is iterated until $|\Delta \boldsymbol{x}_0|$ falls within the defined error tolerance level.

5.2 High-Order Shooting Method

In the context of circuit simulation, shooting methods are essential in time-domain steady-state simulation. While a transient analysis solves an IVP, a shooting method is an iterative procedure layered on top of transient analysis that is designed to solve boundary value problems [6] (BVPs). Since the shooting method is not concerned in finding the transient behaviour of a given system, it is thus desirable to be able to perform the iterative transient procedure as fast as possible without violating the accuracy of the numerical solution. In the light of this argument, the benefit of deriving a high-order shooting algorithm becomes apparent. A stable high-order shooting algorithm will allow bigger step sizes in the numerical integration without compromising the accuracy of the solution, and thus make it possible to faster find an accurate steady-state solution.

The goal in time-domain steady-state analysis is to find the set of initial conditions that will start the numerical integration process at the steady-state solution of the system. This requirement is represented as the solution of the boundary value problem given by,

$$\boldsymbol{\Psi}(\boldsymbol{x_0}) = \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T) - \boldsymbol{x_0} = 0. \tag{5.6}$$

The transition function of $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T)$ is obtained by solving the initial value problem given by,

$$\boldsymbol{G}\boldsymbol{x}(t) + \boldsymbol{C}\dot{\boldsymbol{x}}(t) + \boldsymbol{f}(\boldsymbol{x}, t) - \boldsymbol{b}(t) = 0. \tag{5.7}$$

The nonlinear variational equation in $\boldsymbol{x}_0$ (5.6) can be solved using Newton-Raphson, which necessitates finding the transition function $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T)$ by solving the initial value problem (3.24) between $t = 0$ to $t = T$, as well as computing the following Jacobian matrix,

$$\frac{d\boldsymbol{\Psi}(\boldsymbol{x}_0)}{d\boldsymbol{x}_0} = \frac{d\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{x}_0, t_0, T)}{d\boldsymbol{x}_0} - \boldsymbol{I}\,. \tag{5.8}$$

To find $d\boldsymbol{\phi}(\boldsymbol{x})/d\boldsymbol{x}_0$ the IVP given in equation (5.7) is derived with respect to $\boldsymbol{x_0}$ to obtain,

$$\boldsymbol{G}\boldsymbol{Z}(t) + \boldsymbol{C}\dot{\boldsymbol{Z}}(t) + \underbrace{\boldsymbol{A}(t)\boldsymbol{Z}(t)}_{\boldsymbol{F}(\boldsymbol{Z})} = 0, \tag{5.9}$$

where $\boldsymbol{Z}(t) = d\boldsymbol{x}(t)/d\boldsymbol{x}_0 \in \mathbb{R}^{N\times N}$ is the sensitivity matrix, and $\boldsymbol{A}(t) = d\boldsymbol{f}(\boldsymbol{x}, t)/d\boldsymbol{x} \in \mathbb{R}^{N\times N}$.

To obtain the high-order shooting algorithm, equations (5.7) and (5.9) are discretized using the modified Obreshkov Formula to obtain (3.26) and (5.10) shown below,

$$(\tilde{\boldsymbol{G}} + \tilde{\boldsymbol{C}})\boldsymbol{\xi}_{n+1} + \tilde{\boldsymbol{\rho}}_{n+1} = \tilde{\boldsymbol{b}}_{n+1} \tag{3.26}$$

$$(\tilde{\boldsymbol{G}} + \tilde{\boldsymbol{C}})\tilde{\boldsymbol{Z}}_{n+1} + \tilde{\boldsymbol{F}}_{n+1} = \tilde{\boldsymbol{B}}_{n+1}, \tag{5.10}$$

where $\tilde{\boldsymbol{Z}}_{\boldsymbol{n+1}}$, $\tilde{\boldsymbol{F}}_{\boldsymbol{n+1}}$ and $\tilde{\boldsymbol{B}}_{\boldsymbol{n+1}} \in \mathbb{R}^{(k+1)N\times N}$ such that,

$$\tilde{\boldsymbol{Z}}_{\boldsymbol{n+1}} = \begin{bmatrix} {\boldsymbol{Z}_{n+1}}^T & h{\boldsymbol{Z}_{n+1}^{(1)}}^T & \ldots & h^k {\boldsymbol{Z}_{n+1}^{(k)}}^T \end{bmatrix}^T, \tag{5.11}$$

$$\tilde{\boldsymbol{B}}_{\boldsymbol{n+1}} = \begin{bmatrix} \boldsymbol{0}^T & \boldsymbol{0}^T & \ldots & \sum_{i=0}^{l} \beta_i h^i {\boldsymbol{Z}_n^{(i)}}^T \end{bmatrix}^T, \tag{5.12}$$

$$\tilde{\boldsymbol{F}}_{\boldsymbol{n+1}} = \begin{bmatrix} \boldsymbol{F}(\boldsymbol{Z}_{n+1}) & = & \boldsymbol{A}_{n+1}\boldsymbol{Z}_{n+1} \\ h\boldsymbol{F}(\boldsymbol{Z}_{n+1})^{(1)} & = & h\Big(\boldsymbol{A}^{(1)}_{n+1}\boldsymbol{Z}_{n+1} + \boldsymbol{A}_{n+1}\boldsymbol{Z}^{(1)}_{n+1}\Big) \\ h^2\boldsymbol{F}(\boldsymbol{Z}_{n+1})^{(2)} & = & h^2\Big(\boldsymbol{A}^{(2)}_{n+1}\boldsymbol{Z}_{n+1} + 2\boldsymbol{A}^{(1)}_{n+1}\boldsymbol{Z}^{(1)}_{n+1}\boldsymbol{A}_{n+1}\boldsymbol{Z}^{(2)}_{n+1}\Big) \\ \vdots & & \\ h^{k-1}\boldsymbol{F}(\boldsymbol{Z}_{n+1})^{(k-1)} & = & h^{k-1}\Big(\binom{k-1}{0}\boldsymbol{A}^{(k-1)}_{n+1}\boldsymbol{Z}_{n+1} + \binom{k-1}{1}\boldsymbol{A}^{(k-2)}_{n+1}\boldsymbol{Z}^{(1)}_{n+1} + \cdots + \binom{k-1}{k-1}\boldsymbol{A}_{n+1}\boldsymbol{Z}^{(k-1)}_{n+1}\Big) \\ & & \boldsymbol{0} \end{bmatrix}. \tag{5.13}$$

The equation (5.13) can be simplified to,

$$\tilde{\boldsymbol{F}}_{\boldsymbol{n+1}} = \boldsymbol{J}_{\boldsymbol{n+1}}\tilde{\boldsymbol{Z}}_{\boldsymbol{n+1}}, \tag{5.14}$$

where $\boldsymbol{J}_{\boldsymbol{n+1}}$ is given by,

$$\boldsymbol{J}_{n+1} = \begin{bmatrix} \boldsymbol{A}_{n+1} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ h\boldsymbol{A}^{(1)}_{n+1} & \boldsymbol{A}_{n+1} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ h^2\boldsymbol{A}^{(2)}_{n+1} & 2h\boldsymbol{A}^{(1)}_{n+1} & \boldsymbol{A}_{n+1} & \ddots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ h^{k-1}\binom{k-1}{0}\boldsymbol{A}^{(k-1)}_{n+1} & h^{k-2}\binom{k-1}{1}\boldsymbol{A}^{(k-2)}_{n+1} & \dots & h\binom{k-1}{k-2}\boldsymbol{A}^{(1)}_{n+1} & \boldsymbol{A}_{n+1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}. \tag{5.15}$$

Therefore, (5.10) can be re-written as,

$$(\tilde{\boldsymbol{G}} + \tilde{\boldsymbol{C}} + \boldsymbol{J}_{n+1})\tilde{\boldsymbol{Z}}_{n+1} = \tilde{\boldsymbol{B}}_{n+1}. \tag{5.16}$$

The matrix $\boldsymbol{J}_{n+1}$ described above in (5.15) is formed from the entries of first N-column block of the Jacobian matrix $d\tilde{\boldsymbol{\rho}}_{n+1}/d\boldsymbol{\xi}_{n+1}$ of system (3.26) which is available after every time step update. Hence, systems (3.26) and (5.16) are solved consecutively at each time step. Once $\boldsymbol{J}_{n+1}$ is available, $\tilde{\boldsymbol{Z}}_{n+1}$ is calculated by one matrix inversion. The sensitivity matrix $\boldsymbol{Z}_{n+1}$ at $t = t_{n+1}$ is given by the first $N \times N$ block of $\tilde{\boldsymbol{Z}}_{n+1}$. When $t = T$ the initial conditions $\boldsymbol{x_0}$ is updated by the following Newton-Raphson step,

$$\begin{aligned} \boldsymbol{x}_0^{new} &= \boldsymbol{x}_0^{old} - \underbrace{\left.\frac{d\boldsymbol{\Psi}(\boldsymbol{x}_0)}{d\boldsymbol{x}_0}^{-1}\right|_{\boldsymbol{x}_0=\boldsymbol{x}_0^{old}} \boldsymbol{\Psi}(\boldsymbol{x}_0^{old})}_{\Delta\boldsymbol{x}_0} \\ &= \boldsymbol{x}_0^{old} - \big(\boldsymbol{Z}_T - \boldsymbol{I}\big)^{-1}\big(\boldsymbol{x}_T - \boldsymbol{x}_0^{old}\big). \end{aligned} \tag{5.17}$$

The process is repeated until $\Delta\boldsymbol{x}_0$ falls below the required convergence tolerance level (e_{conv}).

CHAPTER 6
Numerical Results

In this chapter we present some numerical examples that demonstrate the performance of the implementation of the Obreshkov based integration methods, as well as outline the advantages of the proposed method for high-order steady-state simulation. The algorithms used were implemented in Matlab®.

6.1 High-Order Transient Analysis

In this section numerical example is presented to verify that the implementation of the high-order Obreshkov method yields results consistent with what has been described in the literature. Except for the algorithm initialization required to build up the augmented unknowns vector for high-order Obreshkov-based integration meth-

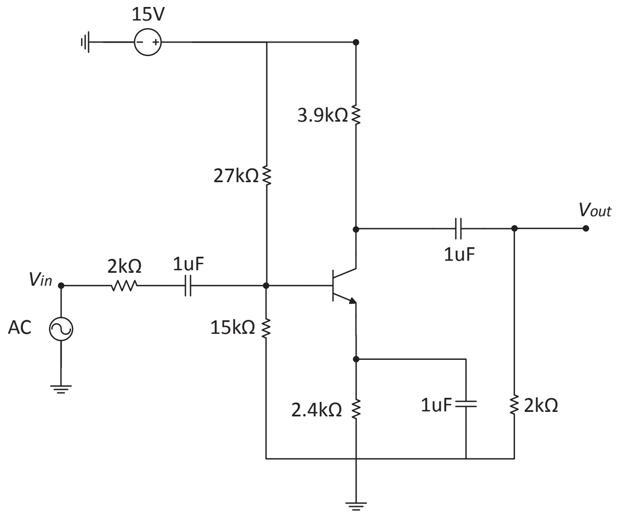

Figure 6–1: Common-emitter Amplifier.

ods, simulations are done using a constant step size. The test circuit shown in figure 6–1 describes a common emitter amplifier. The amplifier has a gain of 30dB at frequencies greater than 10kHz. At the input of the amplifier a sine wave was applied with an amplitude of 0.01V and frequency of 10kHz. The size of the system given by the MNA equation describing the circuit is $N = 13$.

Since a closed form expression for the solution is not possible, the error measurement was done by integrating the system describing the amplifier circuit using Backward Euler (BE) at a very small step size (h = 1ns, which is equivalent to 100,000 points per period) and the result was used as the reference solution. The circuit was then simulated using different step sizes, the relative error was computed at each time point by calculating the deviation of the solution from the reference solution. Figure 6–2 shows a magnified portion of the solution to demonstrate how the error was computed. The green graph represents the reference solution and the blue and black graphs are obtained using BE with step sizes $h = 0.1\mu s$ and $1\mu s$ respectively leading to a maximum relative error of 0.17% and 1.7%. The starred data point is obtained using an 11^{th} order Obreshkov method with a step size $h = 58\mu s$ resulting in a maximum relative error of 0.15%.

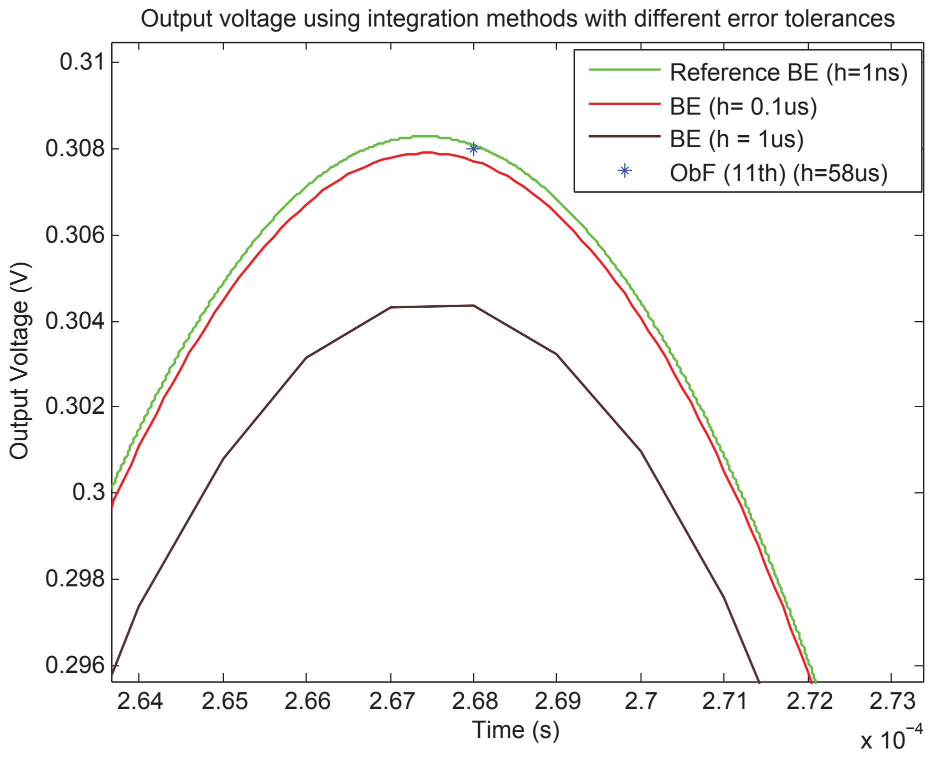

Figure 6–2: Error computation criteria.

Finally, the output of the common emitter amplifier is shown in figure 6–3. The solid line represents the solution obtained using BE at a step size $h = 0.05\mu s$ and the starred data points represent the solution points obtained 11^{th} order Obreshkov method with a step size $h = 58\mu s$, both methods have a maximum relative error of about 0.15%. Note that for the solution obtained by using the 11^{th} order Obreshkov the first few steps are taken with a gradually increasing step size as required by the initialization process using order ramping.

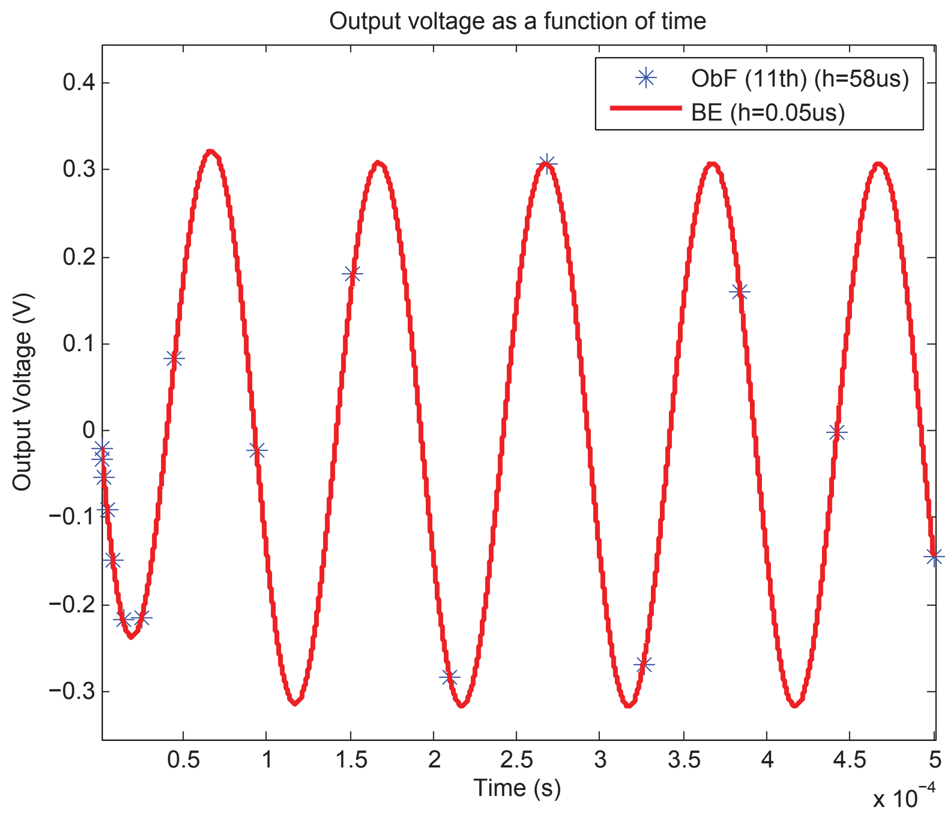

Figure 6–3: The output voltage of the CE amplifier.

6.2 High-Order Steady State Analysis

To evaluate the performance of the high-order shooting method we consider a two-stage low-noise amplifier (LNA) designed to have a gain of 17dB at 1GHz. The amplifier, shown in figure 6–4, also has internally regulated voltage supply to stabilize the gain against temperature variation [23]. The LNA is one of the early stages in an RF mixer circuit and it usually receives signals with different frequencies at its input. In the test scenario, two tones were applied at the input; F_1 = 1GHz, and F_2 = 1.01GHz. Hence the downmixed signal would be at 10MHz which sets the

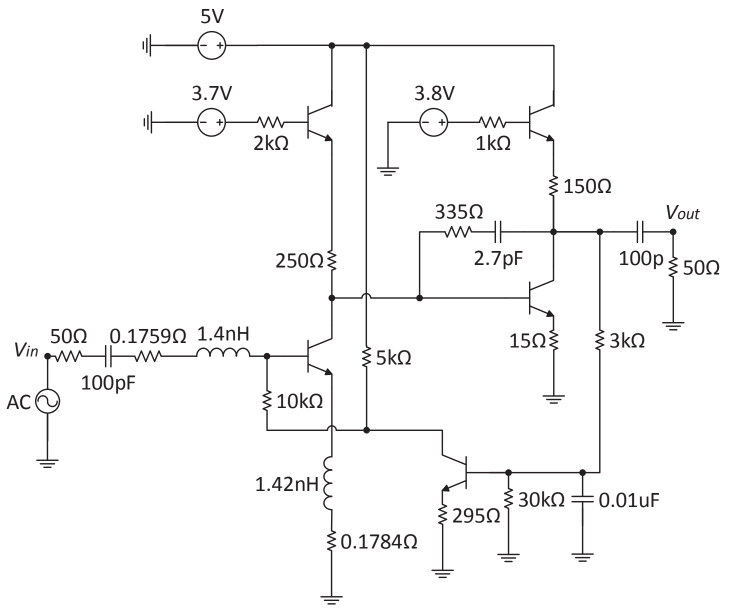

Figure 6–4: Two stage LNA.

period of the system. The size of the system given by the MNA equation describing the circuit is $N = 48$.

The steady state response of the circuit was found using several high-order shooting methods whose order was varied from 2 through 9. The order of these methods is given by $(2k-2)$ or $(2k-1)$. The CPU time consumed by each method was recorded and compared to that of a shooting algorithm based on the L-stable BE integration method. Aside from the algorithm initialization required by high-order Obreshkov-based methods, a constant step size is used in all the simulations.

Since a closed-form expression of the solution is not possible the error was measured in the same way as described in the previous section. A reference solution was computed using BE with very small step size ($h = 10^{-6}$ ns, which is equivalent to

100,000 points per period). The relative error was computed at each time point by calculating the deviation of the solution from the reference solution. The step size for each integration method was chosen so that the maximum relative error to the reference solution remained constant across the different integration methods. To verify that the obtained steady-state solutions have the same accuracy, a Fourier Transform was applied to obtain the intermodulation spectra, which were then compared. A sample is shown in Figure 6–5. The time domain steady state solution is shown in Figure 6–6.

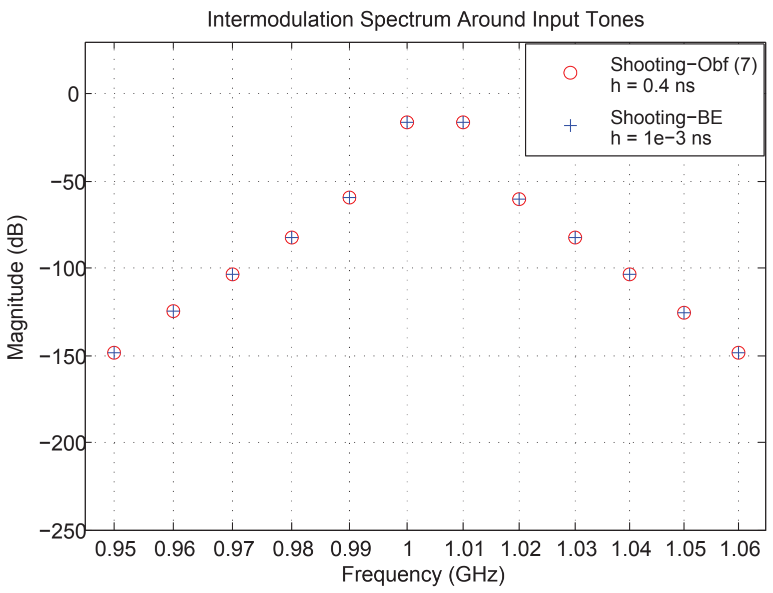

Figure 6–5: The intermodulation spectra at the steady-state solution using BE and ObF 7th order both with the same error tolerance of 0.12%.

Table I summarizes the results of the High-order Obreshkov based shooting method where k is the number of high order derivatives used and h is the step size. The number of Newton iterations required for convergence is also indicated.

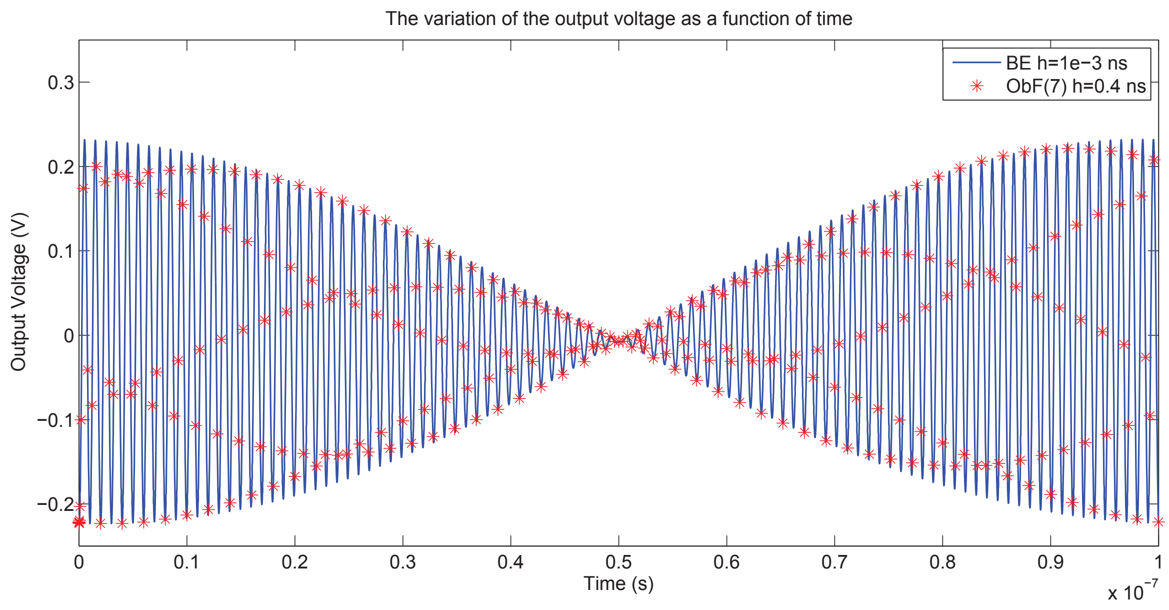

Figure 6–6: Steady-state output of LNA using using BE and ObF 7th order both with the same error tolerance of 0.12%.

Note that a higher order method requires a larger step size and thus a faster CPU time. However, this is balanced to some extent by the larger corresponding system of equations whose size is given by $(k + 1)N \times (k + 1)N$ where N is the size of the original MNA equations.

Table 6–1: High Order Shooting Method Results

k	Method (order)	h (ns)	# iterations	CPU time (s)	speedup
	BE (1st)	1e-3	5	1013.6	1
2	ObF (2nd)	1.2e-2	5	1027.2	0.987
	ObF (3rd)	0.09	5	193.3	5.244
3	ObF (4th)	0.125	5	190.1	5.333
	ObF (5th)	0.25	6	141.9	7.143
4	ObF (6th)	0.3	6	148.8	6.811
	ObF (7th)	0.4	6	125.5	8.076
5	ObF (8th)	0.48	6	148.3	6.835
	ObF (9th)	0.65	7	133.1	7.618

As can be seen from the results the speedup increases as the order of the method increases. However, the extra computation overhead due to the number of derivatives and size of equations can diminish the benefit gained from being able to use a larger step size. This trade-off between the computation overhead and the larger step size allowed can clearly be observed when going from 5^{th} to 6^{th} order ObF and from 7^{th} to 8^{th} order ObF for example. A parallel scheme for the Obreshkov formula was proposes in [24] and addresses this issue for regular transient analysis. However, even a serial implementation in Matlab can still achieve a speedup of up to 8 times as shown in Table 1.

CHAPTER 7
Conclusion

In this thesis a high-order Shooting-Newton method based on the A- and L-stable Obreshkov formula was proposed. The higher order methods allow faster steady-state simulations since a larger step size can be used in the numerical integration without compromising the accuracy of the method. Consequently, a significant reduction in CPU computation time can be obtained for a given level of accuracy. A practical RF circuit was presented as a test scenario and it was shown in the results that speedups of more than 8 are possible. There is still plenty of room for improvement in the performance of the high-order Shooting-Newton method. The implementation presented in this work does not make use of the improved implementation techniques outlined in [8]. Furthermore, the advantage of using a high-order Shooting-Newton algorithm to find the steady-state solution can be even more pronounced if parallelization schemes are utilized.

References

[1] W. J. S.-V. A. Kundert, Kenneth S., *Steady-State Methods for Simulating Analog and Microwave Circuits*. Boston, MA: Springer US, 1990.

[2] E. M. Baily, *Steady-State Harmonic Analysis of Nonlinear Networks*. PhD thesis, 1968.

[3] M. Nakhla and J. Vlach, "A piecewise harmonic balance technique for determination of periodic response of nonlinear systems," *Circuits and Systems, IEEE Transactions on*, vol. 23, no. 2, pp. 85–91, 1976.

[4] S. Egami, "Nonlinear, linear analysis and computer-aided design of resistive mixers," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 22, no. 3, pp. 270–275, 1974.

[5] T. J. A. Jr. and T. N. Trick, "Steady-state analysis of nonlinear circuits with periodic inputs," *Proceedings of the IEEE*, vol. 60, no. 1, pp. 108–114, 1972.

[6] K. Kundert, "Simulation methods for rf integrated circuits," in *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pp. 752–765, IEEE Computer Society, 1997.

[7] G. G. Dahlquist, "A special stability problem for linear multistep methods," *BIT Numerical Mathematics*, vol. 3, no. 1, pp. 27–43, 1963.

[8] Y. Zhou, *Stable High Order Methods for Circuit Simulation*. PhD thesis, Carleton University (Canada), 2011.

[9] E. Gad, M. Nakhla, R. Achar, and Y. Zhou, "A-stable and l-stable high-order integration methods for solving stiff differential equations," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 9, pp. 1359–1372, 2009.

[10] Y. Zhou, E. Gad, M. S. Nakhla, and R. Achar, "Structural characterization and efficient implementation techniques for a-stable high-order integration methods," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 1, pp. 101–108, 2012.

[11] C. W. Gear, *Numerical initial value problems in ordinary differential equations.* Englewood Cliffs, N.J.: Prentice-Hall, 1971. ID: 159668.

[12] P. L. R. Ascher, U. M., *Computer methods for ordinary differential equations and differential-algebraic equations.* Philadelphia: Society for Industrial and Applied Mathematics, 1998. ID: 39007159.

[13] H. B. Keller, *Numerical methods for two-point boundary-value problems.* Waltham, Mass.: Blaisdell, 1968. ID: 168716.

[14] K. Burrage, *Parallel and sequential methods for ordinary differential equations.* Oxford; New York: Clarendon Press ; Oxford University Press, 1995. ID: 34553689.

[15] J. D. Lambert, *Computational methods in ordinary differential equations,.* London; New York: Wiley, 1973. ID: 613517.

[16] J. C. Butcher, *Numerical methods for ordinary differential equations.* Chichester, England; Hoboken, NJ: Wiley, 2008. ID: 244251188.

[17] J. Butcher, "General linear methods," *Acta Numerica*, vol. 15, pp. 157–256, 2006.

[18] A. Iserles and S. Nrsett, *Order stars.* Chapman & Hall, 1991.

[19] N. Obreshkov, "Sur les quadrature mecanique," *(Bulgarian, French Summary) Akad. Nauk.*, vol. 65, pp. 191–289, 1942.

[20] B. Ehle, "High order a-stable methods for the numerical solution of systems of d.e.'s," *BIT Numerical Mathematics*, vol. 8, no. 4, pp. 276–278, 1968.

[21] G. Birkhoff and R. S. Varga, "Discretization errors for well-set cauchy problems.," *Journal of Mathematics and Physics*, vol. 44, pp. 1–23, 1965.

[22] S. K. Vlach, Jiri., *Computer methods for circuit analysis and design.* New York: Van Nostrand Reinhold, 1983.

[23] R. G. Meyer and W. D. Mack, “A 1-ghz bicmos rf front-end ic,” *Solid-State Circuits, IEEE Journal of*, vol. 29, no. 3, pp. 350–355, 1994.

[24] M. A. Farhan, E. Gad, M. S. Nakhla, and R. Achar, “Parallel simulation of large linear circuits with nonlinear terminations using high-order stable methods,” *Components, Packaging and Manufacturing Technology, IEEE Transactions on*, vol. 4, no. 7, pp. 1201–1211, 2014.