Delay Analysis of Digital Circuits Using Prony's Method

by

Jingyi Fu

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the the requirements

For the M.A.Sc. degree in

Electrical and Computer Engineering

School of Information Technology and Engineering

Faculty of Engineering

University of Ottawa

Abstract

This thesis describes possible applications of Prony's method in timing analysis of digital circuits. Such applications include predicting the future shape of the waveform in DTA(Dynamic Timing Analysis) and delay look-up table in STA(Static Timing Analysis).

Given some equally spaced output values, the traditional Prony's method can be used to extract poles and residues of a linear system, *i.e.* to characterize a waveform using an exponential function. In this thesis, not only values but also equally spaced derivatives are tested. Still using same idea of the traditional Prony's method, poles and residues can also be extracted with those values and derivatives. The resultant poles and residues will be used to predict the output waveform in DTA analysis. The benefits brought by the using of derivatives include less simulation steps and less CPU time consuming than the regular constant step simulation.

As a matter of fact, the Prony's method can precisely approximate a complicated waveform. Such property can be applied for STA analysis. The Prony's approximation can be used to precisely record an output waveform, which is used as an entry of the look-up table of STA. Since the accuracy of STA analysis relies on the accuracy of the input and output waveform in the look-up table, the accuracy of the Prony's approach is promising.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, prof. Emad Gad. Without his help this work would not have come to this completion.

I would like to extend my sincere thanks to prof. Michelle Nakhla for his support during the work of this thesis.

I would like to thank Mrs. Yinghong Zhou for providing digital circuits' output data to work with.

I have also to mention my classmates with whom I spent very good time understanding lots of new materials presented in class.

Although coming at the end, but first in mind is my wife, Teng Guo, who gave emotional and social supports for me and pushed me forward during hard time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The fast pace of developments in the micro electronics industry have pushed the operation speed and the integration level to very high unprecedented levels. Such developments have also made stringent demands on the CAD tools to keep pace with those developments.

One of the fundamental and basic activities in digital circuits is the switching between high and low levels. Characterization of a digital cell, through measuring basic things such as delay and slew rate is essential in identifying the critical path in a digital circuit, that is a path with the longest delay, between a set of inputs and a set of outputs in digital block. Such a task is usually known as timing analysis.

Ideally, timing analysis is carried out through formulating the problem in the form of nonlinear differential equations and using some form of a numerical method to trace the waveforms from the digital input to the output. This approach is typically known

as Dynamic Timing Analysis (DTA). Unfortunately, given the large number of inputs in modern digital circuits and the sheer size of the digital circuit, such an approach can easily become cumbersome, especially, that this simulation has to be repeated many times for different combinations of the input signals.

Static Timing Analysis (STA) is a different approach that aims at avoiding these difficulties. STA is carried out by characterizing each cell individually for different input and loading conditions, and then using this information to propagate the delay and slew from cell to cell in the digital path. The information pertinent to each cell is stored in a lockup table that provides the delay and slew at the output corresponding to the delay and slew at the input and the output load capacitance.

It should be obvious from the above description that DTA places large emphasis on the accuracy, whereas STA places strong emphasis on the speed of simulation. The main objective of this thesis is to investigate new approaches that can bridge the gap between STA and DTA.

1.2 Objective of the Thesis

This thesis introduces a new approach to the problem of timing analysis in digital circuits. The main approach presented in this thesis builds on the Prony's method [11] to develop schemes that can be used in the task of timing analysis. Traditionally, the Prony's method has been used in electromagnetic analysis to identify the propagation modes of linear systems [5]. To the best of our knowledge, this is the first work seeking to gear those methods towards the task of timing analysis in nonlinear circuits.

1.3 Contributions

The following are the main contributions developed in this thesis.

1.3.1 Expanding the scope of Prony's method

Traditionally, Prony's method is used to identify the propagation modes in a linear dynamical system given its transient response to an impulse function. On the other hand, the main objective in this thesis has a predictive nature, in that we ask about the delay in propagation from the input to the output, given as little information as possible. To be able to use the Prony's method to predict the future of a digital pulse given its past history requires expanding the basic idea of Prony's method so that it can use few points on the waveform to predict its future shape.

The contribution of this thesis in that regard, is the expansion of the Prony's method so that it can use high-order derivatives in addition to points on the digital waveform to construct. This method should enable using early points on the waveform to predict a significant portion of the future waveform.

1.3.2 Using Prony's method in STA

The thesis also demonstrates how the information extracted by the Prony's method can be used to derive a new approach to STA. In the contribution, we show how a digital cell can be characterized using the poles and residues obtained from the Prony's method.

1.4 Thesis Organization

Chapter 2 provides necessary information on timing analysis. In Chapter 3, methods of solving differential equations are explained, including the new method in [10]. Chapter 4 introduces the traditional Prony's method and its application in electrical engineering. Chapter 5 expands Prony's method to using derivatives. Numerical results are included in Chapter 6. Chapter 7 concludes the whole thesis and propose the possible future work.

Chapter 2

Review of Timing Analysis in Digital Circuits

This chapter provides a brief review of the concept of timing analysis in digital circuits as well as the main approaches used to perform it. Section 2.1 covers the static approach to timing analysis or STA, whereas Section 2.2 provides an insight into the dynamic approach, or the DTA.

2.1 Static Timing Analysis

The term static here implies that the timing analysis is performed in a manner that is independent of the input pattern, where the objective is to find the worst-case delay of the circuit over all possible combinations of the inputs [2].

2.1.1 Using Graphs To Represent Digital Circuits

A digital circuit can be classified as either being a combinational or a sequential circuit. A combinational circuit can be represented using the notion of a directed graph, $G = (V, E)$, where V are the set of vertices and E represents the set of edges of the graph. The vertices V in the graph represent the inputs and outputs of the logic circuit, whereas the edges, E, represent the connection between the vertices. In general there are two sets of edges: the first set connects each input of the gate to the output, whereas the other set represents the connections from the outputs of each gate to the inputs of its fanouts. Attached to each edge in the graph is a pair of numbers that represent the rise/fall delay associated with the connection. These delays are normalized with respect to a delay of a given gate representing the unit gate such as the minimum-sized inverter. The delays associated with the first set of edges provide the propagation delay within each gate, and the delays associated with the second set of edges represent the interconnect delays between the digital gates [23]. The following is an example used to illustrate using directed graphs to represent the simple digital circuit shown in Figure 2.1.

The example in Figure 2.1 presents a combinational circuit with three NAND Gates, $G1$, $G2$ and $G3$. The timing graph representing this circuit is shown in Figure 2.2.

The numbers within the gate represent, respectively, the rise/fall propagation delay within the gate. In this example, we assume that there is no delay arising from the interconnect, for simplicity. Thus, in this example, the rise/fall at the inputs of $G3$ are the same rise/fall introduced by the previous gate, $G1$. $G3$ itself is a NAND

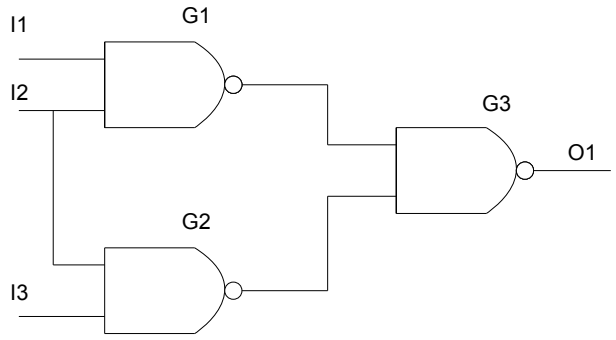

Figure 2.1: Example of a combinational circuit

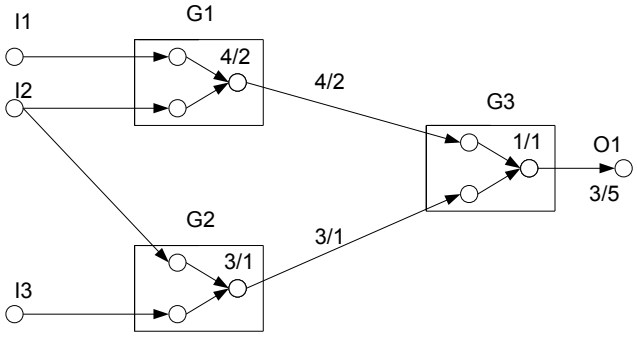

Figure 2.2: Example timing graph

gate with a rise/fall delay of 1/1. Thus, the fall delay of the path of $G1$ and $G3$ is $4 + 1 = 5$ and its rise delay is $2 + 1 = 3$. Similarly, the rise/fall delay of the path $G2$ and $G3$ is 2/4. Hence, this makes clear that the worst case rise/fall delay for entire path is 3/5. The worst-case delay typically determine the fastest speed for the operation.

A sequential circuit contains both combinational elements and sequential elements such as flip flops and latches, and can be represented as a set of combinational blocks between latches. Sequential elements are usually left unrepresented, whereas a timing graph is built for each of the combinational blocks [23].

2.1.2 Delay Look-up Tables

The information used in graph-based representation of digital circuits usually depend on the input stimulus of the logic gate and its output load [4] [8]. Hence, instead of using a single rise/fall pair to represent a given edge in a timing diagram, one needs a look-up table, whose entries provide the rise/fall delay corresponding to a given input and output condition.

In general, there are three parameters used to characterize a digital gate response. The first two parameters are the rise and fall delays which represent the delay between 50% output propagation time, which is the time span between $50\%V_{DD}$ at the input and $50\%V_{DD}$ at the output.

The third parameter used in characterizing a digital cell is the output slew, which is defined as the time difference between the points at which the output attains 10% and 90% of its final settling.

As mentioned earlier, these parameters depend on the input and output loading of a given cell. Consequently, they are stored in the form of a look-up table generated by detailed transistor-level SPICE simulation. For example, generating the look-up table for an inverter, the inverter is embedded in the circuit shown in Figure 2.3. Subsequently the circuit is simulated repetitively for different values of the load capacitance, C_L and different shapes of the input signal, whereby the above-mentioned parameters are extracted from the output waveform.

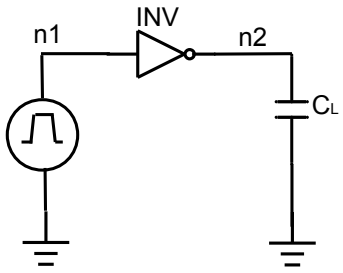

Figure 2.3: Delay test circuit for an inverter

The most widely used standard input in STA is the saturated ramp defined by:

$$r(t)/V_{DD} = \begin{cases} 0, t \leq t_{50\%} - 0.5t_s \\ \frac{t - t_{50\%} + 0.5t_s}{t_s}, t_{50\%} - 0.5t_s \leq t \leq t_{50\%} + 0.5t_s \\ 1, t \geq t_{50\%} + 0.5t_s \end{cases} \tag{2.1}$$

where $t_{50\%}$ stands for the time at which the signal reaches $50\%V_{DD}$ and t_s represents the slew of the input signal. Note here that the shape of the input waveform is controlled through only one parameter, t_s. Figure 2.4 gives an example of a saturated ramp having $t_{50\%} = 0.1ns$ and $t_s = 0.2ns$ for $V_{DD} = 1.2V$.

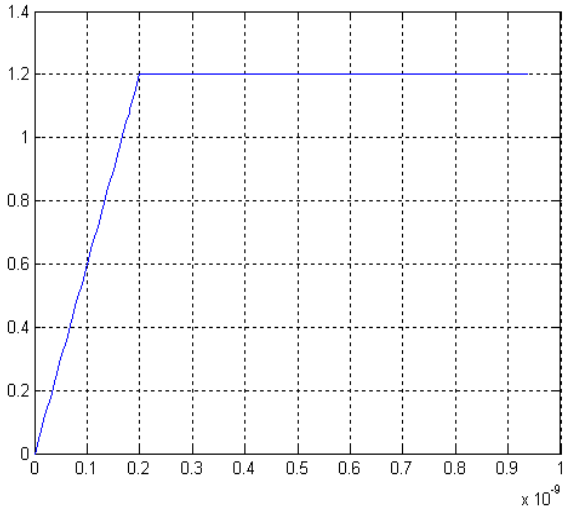

Figure 2.4: Example of a saturated ramp signal

With a delay look-up table, rise/fall delay and output slew for different input slew and load capacitance can be determined. Linear interpolation is used to determine the values between table entries.

The following example illustrates the delay calculation of an inverter characterized by tables 2.1- 2.3 for three values of load capacitances and three values for input slew.

Load Cap./Input Slew	0.05	0.2	0.5
0.01pF	0.03	0.18	0.33
0.5pF	0.06	0.36	0.66
2.0pF	0.09	0.72	1.32

Table 2.1: Rise Delay

Load Cap./Input Slew	0.05	0.2	0.5
0.01pF	0.02	0.16	0.30
0.5pF	0.04	0.32	0.60
2.0pF	0.08	0.64	1.20

Table 2.2: Fall Delay

Load Cap./Input Slew	0.05	0.2	0.5
0.01pF	0.01	0.09	0.15
0.5pF	0.03	0.27	0.45
2.0pF	0.06	0.54	1.90

Table 2.3: Fall Output Slew

Assume that in this example the designer is interested computing the following

- the fall delay for a rising input with slew=0.1
- the rise delay for a falling input with slew=0.147 and output load capacitance $C_L = 1.0pF$.

We examine these two cases in the following case studies

- CASE 1: Input signal is a rising edge with slew=0.1.

  The output is a falling edge. Check the Fall Delay table and linearly interpolate the table entries. The resultant fall delay=0.178. Check the Fall Output Slew table and linearly interpolate the table entries. The resultant fall output slew=0.147.

- CASE 2: Input signal is a falling edge with slew=0.12.

The output is a rising edge. Check the Rise Delay table and linearly interpolate the table entries. The resultant rise delay=0.261.

The load capacitance of each gate can be determined by computing the effective capacitance. If a logic gate's output is connected to RC interconnect, the effective load capacitance can be computed numerically [13] [20] [24] [25] [1]. If a logic gate's output is connected to another logic gate, the effective capacitance for any given gate can be obtained as follows: determine equivalent inverter for the gate, drive the inverter through a high resistance, measure the delay across the resistance, and compare it with an RC delay where C is the gate effective capacitance we are trying to compute [3].

2.1.3 Path STA in Combinational Circuits

Aided with the tables characterizing each cell individually, timing analysis for a given path consisting of successive cells can be carried out using one of several methods available in the literature. One of the most popular methods used in this task is the Critical Path Method (CPM), which is described here briefly.

The *critical path* is defined as the path between an input and an output with the maximum delay [15]. To locate the critcal path, we begin with the block whose output has the longest arrival time. Then the longest arriving input to this block is identified. Next the preceding block causing this transition is also identified. This process is repeated recursively until a primary input is reached.

The CPM proceeds from the primary inputs to the primary outputs in topological order. It computes the worst-case rise and fall arrival time at each intermediate node,

and eventually at the output of a circuit. Then the overall worst-case rise and fall arrival time for the whole circuit can be found [23].

2.1.4 Path STA in Sequential Circuits

In general, any sequential circuit can be represented as combinational logic and registers, which are driven by clock. Then the clock issue plays a dominant role for edge-triggered sequential circuits.

There are some special requirements for a combinational subcircuit in a sequential circuit. Let's denote the setup time, hold time, the maximum clock-to-output delay and minimum clock-to-output delay of any arbitrary flip-flop k as T_{sk}, T_{hk}, Δ_k and δ_k respectively, and let's assume the clock repeats after every P units of time. The delay of a combinational path $i \rightarrow j$, noted as $d(i,k)$, should satisfy the following two inequalities [23]

$$d(i,j) \leq P - T_{sj} - \Delta_i \tag{2.2}$$

$$d(i,j) \geq T_{hj} - \delta_i \tag{2.3}$$

Moreover, due to differences in interconnect delays on the clock distribution network, the clock signals do not arrive at all of the registers at the same time. This is called a "skew" in the clock. The designer must ensure each input-output path of a combinational subcircuit has a delay less than the clock period with the presence of clock skew.

2.2 Dynamic Timing Analysis

It should be obvious by now that the basic idea in STA assumes that the characteristics or behaviour of an individual cell does not change dramatically when connected in a bigger path, and its dependence on the driving input and output load in a simple and almost linear manner. Clearly, this is an oversimplification since a logical gate is inherently a nonlinear whose devices have strong nonlinear behaviour.

DTA, on the other hand, does not proceed from that assumption. Instead, it starts with the whole path and describes it as a system of nonlinear differential equations [22]. The timing analysis information are subsequently extracted from the numerical solution of the system of differential equations. More on the numerical solution of differential equations will be presented in Chapter 3.

Clearly, the result of DTA is very accurate since there are no assumptions made about the behaviour of the cell. However, these results correspond to only one set of inputs and to obtain the worst/best case, the numerical simulations as such will have to be repeated for all possible configurations of input waveforms. As a result of the complexity of modern, it becomes almost impossible to go through all the combinations of the inputs. For example, in order to simulate a 32- bit adder, $2^{64} = 1.845 \times 10^{19}$ different input patterns need to be generated [21].

Thus, to make the idea of DTA practically possible, a method is needed to generate efficient input test vectors to ensure that the worst or best case can be covered by a selected few inputs.

There are no standard procedures to generate test vectors. Usually, test generation methods can be classified into two categories: *random* or *deterministic*, of

which random testing is dominant over deterministic, and is the origin of most of the simulation-based tools. [21] Random methods do not require information on circuit structure to minimize the test set. The basic random methods generate stimuli with equal probability of appearance of $0s$ and $1s$.

Although a Dynamic Timing Analysis usually takes a long time to complete, it can not be replaced by Static Timing Analysis, because Dynamic Timing Analysis is more suitable in asynchronous circuits than Static Timing Analysis. Also Dynimic Timing Analysis can check the functionality of the design.

This thesis is trying to find a method to decrease the number of simulations, so that Dynamic Timing Analysis can be speeded up.

2.3 Summary

This chapter described briefly the underlying ideas of STA and DTA. It was mentioned that while DTA can provide accurate analysis, it remains largely impractical for large integrate circuits. It also shown that STA is based on characterizing each cell individually under certain input and output conditions, and assuming that these characteristics will not vary significantly when connected to other cells. As such, STA provides a fast analysis at the expense of accuracy.

Chapter 3

Numerical Solution of Circuit Differential Equations

The goal of this chapter is to provide some insight on the idea of DTA through presenting the mainstream and advanced numerical methods used in solving the differential equations representing the circuits to be analyzed for timing information.

3.1 Mathematical Circuit Formulation

Before a circuit simulation can be carried out, the circuit needs to be represented in a mathematical model. Modified Nodal Analysis (MNA) is the approach typically used to represent a linear or non-linear circuit in the mathematical domain. Using MNA, a nonlinear circuit can be represented in the time-domain in the form of a system of differential equations as follows

$$\mathcal{C}\frac{d\mathbf{x}(t)}{dt} + \mathcal{G}\mathbf{x}(t) + \mathbf{f}(\mathbf{x}(t)) = \mathcal{B}(t) \tag{3.1}$$

where the vector $\mathbf{x}$ stores all nodal voltages and currents of inductors and independent voltage sources, the matrix C stores capacitors, inductors and frequency-dependent members, the matrix G stores all conductors and frequency-independent members, $\mathbf{f}(\mathbf{x}(\mathrm{t}))$ is a vector that describes non-linear relations of non-linear components, and the vector B stores independent sources information.

Figure 3.1 presents a sample of circuit elements and their MNA stamps, which represents their contribution to the formulation in (3.1).

Element	Circuit Schematic	Stamp
Resistor		$\begin{array}{c} \begin{array}{cccc} . & i & \cdots & j \end{array} \\ \begin{array}{c} i \\ \vdots \\ j \end{array} \begin{bmatrix} g & \cdots & -g \\ \vdots & \ddots & \vdots \\ -g & \cdots & g \end{bmatrix} \\ \mathbf{G} \end{array}$
Capacitor		$\begin{array}{c} \begin{array}{cccc} . & i & \cdots & j \end{array} \\ \begin{array}{c} i \\ \vdots \\ j \end{array} \begin{bmatrix} C & \cdots & -C \\ \vdots & \ddots & \vdots \\ -C & \cdots & C \end{bmatrix} \\ \mathbf{C} \end{array}$
Inductor		$\begin{array}{c} \begin{array}{cccc} i & . & . & j \quad I_L \end{array} \\ \begin{array}{c} i \\ \vdots \\ j \\ I_L \end{array} \begin{bmatrix} 0 & \cdots & 0 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & -1 \\ 1 & \cdots & -1 & -sL \end{bmatrix} \\ \mathbf{C} \end{array}$
Current Source		$\begin{array}{c} \begin{array}{c} i \\ \vdots \\ j \end{array} \begin{bmatrix} J \\ \vdots \\ -J \end{bmatrix} \\ \mathbf{B} \end{array}$
Voltage Source		$\begin{array}{c} \begin{array}{cccc} i & . & . & j \quad I_E \end{array} \\ \begin{array}{c} i \\ \vdots \\ j \\ I_E \end{array} \begin{bmatrix} 0 & \cdots & 0 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & -1 \\ 1 & \cdots & -1 & 0 \end{bmatrix} \begin{bmatrix} . \\ . \\ . \\ E \end{bmatrix} \\ \mathbf{G} \qquad \mathbf{B} \end{array}$

Figure 3.1: Some basic eletrical components and their stamps

The following is an example of representing non-linear circuit in MNA formulation. The non-linear component in the circuit is a non-linear capacitor.

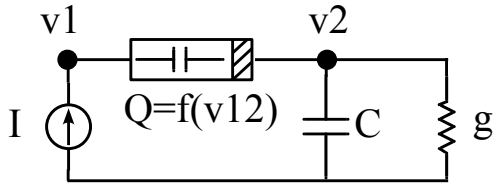

Figure 3.2: Non-liear circuit example

This circuit could be described using MNA formulation as follow:

$$\mathcal{G} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & g & 0 \\ 0 & 0 & -1 \end{bmatrix}, \mathcal{C} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & C & -1 \\ 0 & 0 & 0 \end{bmatrix}, \mathcal{B} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

$$\mathbf{u}(t) = [I],\ \mathbf{x}(t) = \begin{bmatrix} v_1(t) \\ v_2(t) \\ Q(t) \end{bmatrix}, \text{ and } \mathbf{f}(\mathbf{x}(t)) = \begin{bmatrix} 0 \\ 0 \\ f(v_1(t) - v_2(t)) \end{bmatrix}$$

3.2 Linear Multi-Step (LMS) Methods

Once the MNA formulation for a general circuit has been constructed as shown above, a numerical method can be used to solve for the vector of unknowns, $\mathbf{x}(t)$ at discrete time steps, t_0, t_1, ..., t_n.

One of the most popular methods is the Linear Multi-Step Methods (LMS). An LMS method seeks to approximate the unknown waveform $x(t)$ by a polynomial function in t, $\hat{x}(t)$. For example, $\hat{x}(t)$ could be chosen to be second degree polynomial in t

$$\hat{x}(t) = a_0 + a_1 t + a_2 t^2 \tag{3.2}$$

, where the coefficients a_0, a_1 and a_2 are to be determined given some information

about the original $x(t)$.

For example, if one knows about $x(t)$ and its derivatives at $t = 0,\ h,\ 2h$, then it becomes possible to construct the following system of equations

$$\begin{cases} x_0 = a_0 \\ x_0' = a_1 \\ x_1 = a_0 + a_1 h + a_2 h^2 \\ x_1' = a_1 + 2a_2 h \\ x_2' = a_1 + 4a_2 h \end{cases} \tag{3.3}$$

, where $x_0 = x(0)$, $x_1 = x(h)$, $x_2 = x(2h)$, $x_0' = x'(0)$, $x_1' = x'(h)$, $x_2' = x'(2h)$.

To find a_0, a_1 and a_2, one can use the first three equations to form the system

$$\begin{cases} x_0 = a_0 \\ x_1 = a_0 + a_1 h + a_2 h^2 \\ x_2' = a_1 + 4a_2 h \end{cases} \tag{3.4}$$

whose solution would produce the coefficients as follows

$$\begin{cases} a_0 = x_0 \\ a_1 = \frac{-4x_0 + 4x_1 - x_2' h}{3h} \\ a_2 = \frac{x_0 - x_1 + x_2' h}{3h^2} \end{cases} \tag{3.5}$$

Substituting for a_0, a_1 and a_2 in (3.5) into (3.2) and letting $t = 2h$, one gets the following formula for x_2

$$h\hat{x}_2' - \frac{3}{2}\hat{x}_2 + 2\hat{x}_1 - \frac{1}{2}\hat{x}_0 = 0 \tag{3.6}$$

In fact, (3.6) only represents a special instance of the general class of LMS methods whose general formulation takes on the following form

$$\sum_{j=0}^{k} a_j \hat{x}_{n+k-j} - h \sum_{j=0}^{k} b_j \hat{x}'_{n+k-j} = 0 \tag{3.7}$$

, where a_j and b_j, $j = 0$, ..., k are coefficients that are determined in a procedure similar to the sone described above. The general class of LMS methods is divided into further subclasses based on the particular values for a_i and b_i. Thus an LMS method is referred to as

- Explicit method (or predictor) if either a_0 or b_0 is equal to zero, or
- Implicit method (or corrector) if both a_0 and b_0 are equal to zero, or
- Backward differentiation formula (BDF) method if $b_1 = b_2 = ... = b_k = 0$ [26].

In addition to the above-mentioned classifications of LMS methods several other important characteristics are used to characterize LMS methods. The following subsections summarize those characteristics of Truncation Error, Order and Stability.

3.2.1 Local Truncation Error (LTE)

LTE refers to the error resulting from using a finite degree polynomial to approximate a general unknown waveform. To deduce the LTE arising from a given LMS formula, we substitute for $\hat{x}_{n+k-j}$ and $\hat{x}'_{n+k-j}$ with the actual waveform values, that is $x(t_{n+k} - jh)$ and $x'(t_{n+k} - jh)$. Clearly, since $\hat{x}(t)$ and $x(t)$ are different, then it is not expected that $x(t)$ will satisfy (3.7) perfectly. Instead, there will be an error term that can be collected on the right side

$$\sum_{j=0}^{k} a_j x(t_{n+k} - jh) - h \sum_{j=0}^{k} b_j x'(t_{n+k} - jh) = \epsilon \tag{3.8}$$

To find the error term more explicitly, we expand $x(t_{n+k} - jh)$ and $x'(t_{n+k} - jh)$ around t_{n+k} as follows

$$x(t_{n+k} - jh) = x(t_{n+k}) - \frac{jh}{1!} x'(t_{n+k}) + \frac{(jh)^2}{2!} x''(t_{n+k}) + \dots \tag{3.9}$$

$$x'(t_{n+k} - jh) = x'(t_{n+k}) - \frac{jh}{1!}x''(t_{n+k}) + \frac{(jh)^2}{2!}x'''(t_{n+k}) + \ldots \tag{3.10}$$

Substituting (3.9) and (3.10) into (3.8) results in

$$C_0 x(t_{n+k}) + hC_1 x'(t_{n+k}) + h^2 C_2 x''(t_{n+k}) + \ldots + h^m C_m x^{(m)}(t_{n+k}) + \ldots = \epsilon \tag{3.11}$$

, where

$$C_0 = a_0 + a_1 + a_2 + \ldots + a_k$$

$$C_1 = -\left[\tfrac{1}{1!}(a_1 + 2a_2 + \ldots + ka_k) + \tfrac{1}{0!}(b_0 + b_1 + \ldots + b_k)\right]$$

$$C_2 = \tfrac{1}{2!}(a_1 + 2^2 a_2 + \ldots + k^2 a_k) + \tfrac{1}{1!}(b_1 + 2b_2 + \ldots + kb_k)$$

$$C_3 = -\left[\tfrac{1}{3!}(a_1 + 2^3 a_2 + \ldots + k^3 a_k) + \tfrac{1}{2!}(b_1 + 2^2 b_2 + \ldots + k^2 b_k)\right]$$

$$\vdots$$

$$C_m = (-1)^m \left[\tfrac{1}{m!}(a_1 + 2^m a_2 + 3^m a_3 + \ldots + k^m a_k) + \tfrac{1}{(m-1)!}(b_1 + 2^{m-1} b_2 + 3^{m-1} b_3 + \ldots + k^{m-1} b_k)\right].$$

It can be shown that if a p^{th} order polynomial is used in approximating the unknown $x(t)$, then coefficient C_0 up to C_p vanish identically leaving only C_{p+1}, C_{p+2}, ... Hence, the approximation error becomes equal to

$$\epsilon = h^{p+1} C_{p+1} x^{(p+1)}(t_{n+k}) + \ldots . \tag{3.12}$$

The first term in the above series typically devotes the so-called LTE defined as follows

$$LTE = h^{p+1} C_{p+1} x^{(p+1)}(t_{n+k}) \tag{3.13}$$

where C_{p+1} is the LTE constant.

3.2.2 Order of LMS Methods

The order of an LMS is defined as the degree of the polynomial used to approximate $x(t)$. Thus if an order p method is used, then the degree of approximating polynomial, $\hat{x}(t)$ is p, i.e.

$$\hat{x}(t) = \sum_{i=0}^{p} a_i t^i \tag{3.14}$$

The coefficients of the polynomial can be found using information about the numerical values of the waveform, its derivatives (or a combination thereof) at several points in the past. In general, it is not necessary to compute the coefficient a_i explicitly, since eventually, one always ends up with a formula similar to the one in (3.7), where the coefficients a_i and b_i are computed *a priori* and used later for all problems.

3.2.3 Stability in the LMS Methods

Stability is another important property of a LMS formula. A numerical integration formula must also be stable in order to follow a stable output response. Stability is studied by placing a stability testing problem $x' = \lambda x$ into the LMS formula (3.7), where λ is a constant on the LHS of the complex plane. This results in

$$\sum_{j=0}^{k} (a_j - h\lambda b_j) x_{n+k-j} = 0 \tag{3.15}$$

The exact solution for testing function $x' = \lambda x$ is

$$x(t) = x(0)e^{\lambda t} \tag{3.16}$$

We know the solution of (3.15) also has a similar form of

$$x_{n+k-j} = x(0)e^{\lambda h(n+k-j)} \tag{3.17}$$

Substituting this solution to the equation (3.15), we get

$$x(0)e^{\lambda hn}\left(\sum_{j=0}^{k} a_j e^{\lambda h(k-j)} - \lambda h \sum_{j=0}^{k} b_j e^{\lambda h(k-j)}\right) = 0 \tag{3.18}$$

It can be simplified to

$$\sum_{j=0}^{k} a_j e^{\lambda h(k-j)} - \lambda h \sum_{j=0}^{k} b_j e^{\lambda h(k-j)} = 0 \tag{3.19}$$

The characteristic function for $e^{\lambda h}$ is

$$\sum_{j=0}^{k} a_j z^{k-j} - \lambda h \sum_{j=0}^{k} b_j z^{k-j} = 0 \tag{3.20}$$

It must have k roots since it is a polynomial of degree k. Because for a stable circuit λ is on the LHP and step size h is always a positive real number, we require all the roots of the characteristic function (3.20) stay inside the unit circle. For $h = 0$, the characteristic function (3.20) becomes:

$$\sum_{j=0}^{k} a_j z^{k-j} = 0 \tag{3.21}$$

The characteristic function (3.21) at $h = 0$ must have all roots inside the unit circle, and this is required by all LMS formula. This type of stability is called *zero stability*. For $h = \infty$, the characteristic function (3.20) becomes:

$$\sum_{j=0}^{k} b_j z^{k-j} = 0 \tag{3.22}$$

This type of stability is called A_∞ *stability*. For h is other number, we check the stability boundary, or let the root on the unit circle $z = e^{j\phi}$. The characteristic function (3.20) can be rewritten as:

$$q = \frac{\sum_{j=0}^{k} a_j z^{k-j}}{\sum_{j=0}^{k} b_j z^{k-j}} \tag{3.23}$$

, where $q = \lambda h$ is a complex number. We can draw the stability boundrary on q plane by inserting the results of equation (3.23). To choose some values on the unit circle, we start from $z = 0$ passing $z = j$ and end at $z = -1$. Corresponding to each value of chosen z, equation (3.23) gives a point on q complex plane. The stable region is on the left of the curve proceeding direction on the q plane. The following figures demonstrate the stable region of Forward Euler, Backward Euler, Trapezoidal and BDF. For all the stable λ on the LHP, we hope that the stable region on q plane can also cover all the LHP, so that a big step size of h is safe.

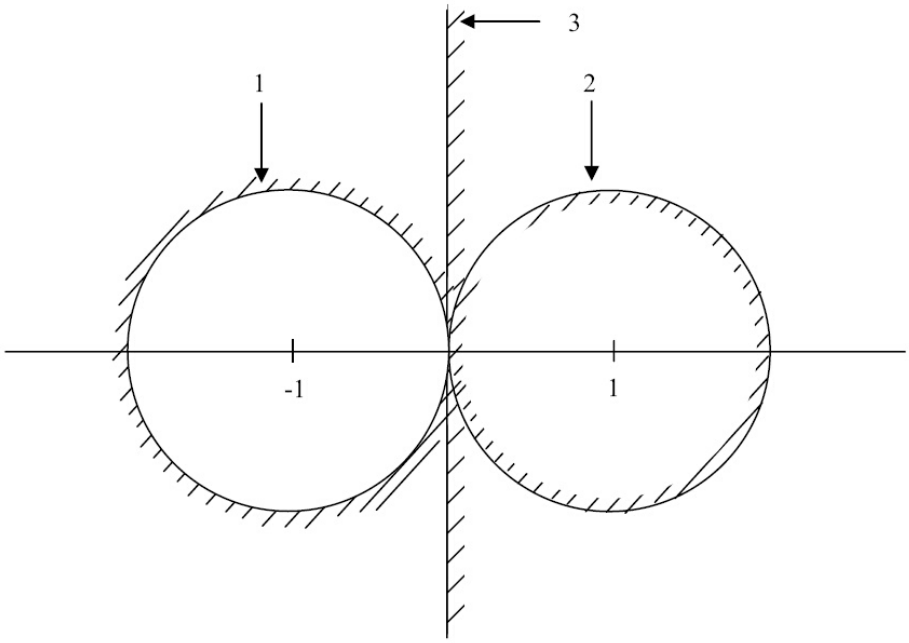

Figure 3.3: Stable region 1,Forward Euler, stable inside; 2, Backward Euler, stable outside; 3, Trapezoidal, stable LHS

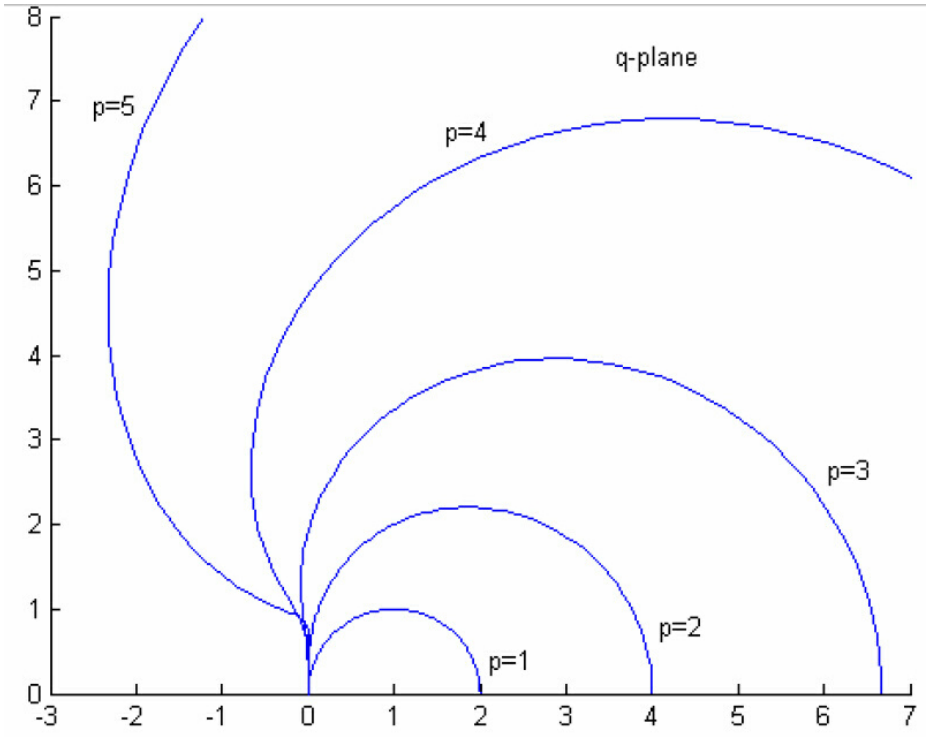

Figure 3.4: Stable region of BDF stable outside

3.2.4 Circuit Simulation Using LMS Methods

The previous subsections outlined the derivations and final form for the LMS methods. Here, we describe briefly how they can be used to perform the actual solution of the system of differential equations describing the MNA formulation. To make this presentation straightforward, we rewrite the BDF special form of the LMS

$$\sum_{i=0}^{k} a_j x_{n+k-j} - h b_0 x'_{n+k} = 0 \tag{3.24}$$

Notice here the above formula defines the derivative at t_{n-k} in terms of the previous points. Hence pre-multiplying by the matrix G on both sides of (3.1) and substituting from the MNA formulation

$$\sum_{j=0}^{k} a_j x_{n+k-j} + h b_0 [G x_{n+k} + f(x_{n+k}) - b(t_{n+k})] = 0 \tag{3.25}$$

Note that x_{n+k-j}, $j = 1, ..., n$ are assumed to be known. Hence, the above system is a system of nonlinear algebraic equations that can be solved using Newton iterative techniques for x_{n+k}.

3.3 Integration with High Order Derivatives

It is well established that LMS methods lose A-stability for orders higher than 2 [7]. A recent method was proposed in [10] to develop high-order methods that can maintain their A-stability. These methods incorporate the high-order derivatives to achieve the A-stability. Derivations of High-order methods can be done in a manner similar to the derivation of LMS outlined above. The final result, however, will be a formula similar to the LMS, that relates the values and derivatives at several successive points. The

general multistep high-order integration formula takes the following form

$$x_{n+k}^{(l)} = (-1)^l \frac{1}{h^l} \left(\sum_{j=0}^{k_1} a_j^C x_{n+k-j} - h \sum_{j=0}^{k_2} b_j^C x'_{n+k-j} + \ldots + (-1)^l h^l \sum_{j=0}^{k_m} c_j^C x_{n+k-j}^{(l)} \right) \tag{3.26}$$

, where

$$x_{n+k-j}^{(l)} = \frac{d^l}{dt^t} x(t)|_{t=t_{n+k-j}}$$

and h is the step size, i.e.

$$h = t_{m+1} - t_m$$

The coefficients a_j^c are determined using a procedure described in Appendix A.

3.3.1 The Obreshkov-based High-order Formula

A special class of the method in (3.26) is the single-step method defined by

$$\sum_{j=0}^{k} (-1)^j a_{j,k} h^j x_{n+1}^{(j)} = \sum_{j=0}^{k-p} a_{j,k-p} h^j x_n^{(j)} \tag{3.27}$$

This subclass enjoys the following advantages [10]

1. The coefficients $a_{j,k}$ can be written in a closed-form as shown next

$$a_{j,k} = (2k-j)! \frac{k!}{j!(k-j)!}; \tag{3.28}$$

2. It can be shown that this class of method is A-stable and L-stable if $p = 0$ or 1 and 2, respectively;

3. It can be shown that the LTE of this class can reach the theoretical minimum, where the coefficient of the truncation error is

$$C_{k-p,k} = (-1)^k \frac{(k-p)!k!}{(2k-p)!(2k-p+1)!}; \tag{3.29}$$

4. It can also be shown that the order of the Obreshkov-formula is

$$O((\lambda h)^{2k-p+1})$$

, which indicates that the Obreshkov-formula is a high order formula associated with high accuracy.

3.3.2 Circuit Simulation Using the Obreshkov Formula

The Obreshkov formula was used recently in [23] for the purpose of transient simulation of electronic circuits. This section summarizes the basic implementation procedures. We begin first by describing the implementation for linear circuits.

It was also emphasized that LMS-based method suffer a conflict between order and stability, which makes any method offering order high than 2 lose the A-stability property.

On the other hand, methods based on high-order derivatives can be made to be A-stable regardless of the order used. This has made it an attractive alternative for circuit simulation.

The implementation of high-order methods based on the Obreshkov formula also presented one more advantage that is of interest to this thesis, namely that of producing the high-order derivatives scaled by powers of the step size. Later, in this thesis, we will present a technique that shows how those scaled derivatives can be used to perform basic timing simulation using the idea of Prony's method.

3.3.2.1 Linear Circuit Simulation

For a linear circuit equaion (3.1) becomes

$$\mathcal{C}\frac{d\mathbf{x}(t)}{dt} + \mathcal{G}\mathbf{x}(t) = \mathcal{B}(t) \tag{3.30}$$

To apply Obreshkov's method, we consider the following augmented system:

$$\widetilde{\mathcal{C}}\xi_{n+1} + \widetilde{\mathcal{G}}\xi_{n+1} = \widetilde{\mathcal{B}}_{n+1} \tag{3.31}$$

The matrices $\widetilde{\mathcal{C}}$ and $\widetilde{\mathcal{G}} \in \mathbf{R}^{(k+1)N\times(k+1)N}$ in (3.31) are obtained from $\mathcal{C}$ and $\mathcal{G}$ as follows

$$\widetilde{\mathcal{C}} = \frac{1}{h}\begin{bmatrix} \mathbf{0} & \mathcal{C} & \dots & & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{C} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathcal{C} \\ \mathbf{0} & \mathbf{0} & & \dots & \mathbf{0} \end{bmatrix} \tag{3.32}$$

,

$$\widetilde{\mathcal{G}} = \begin{bmatrix} \mathcal{G} & \mathbf{0} & \dots & & \mathbf{0} \\ \mathbf{0} & \mathcal{G} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathcal{G} & \mathbf{0} \\ a_{0,k}\mathbf{I} & -a_{1,k}\mathbf{I} & & \dots & (-1)^k a_{k,k}\mathbf{I} \end{bmatrix} \tag{3.33}$$

where $\mathbf{I}$ is an identity matrix of size N. The vector ξ in (3.31) is defined by

$$\xi_{n+1} = \begin{bmatrix} \mathbf{x}_{n+1}^{(0)\,T} & h\mathbf{x}_{n+1}^{(1)\,T} & \dots & h^k\mathbf{x}_{n+1}^{(k)\,T} \end{bmatrix}^T \tag{3.34}$$

The source vector $\widetilde{\mathcal{B}}_{n+1}$ is defined in terms of the h-scaled derivatives of the source as follows

$$\widetilde{\mathcal{B}}_{n+1} = \begin{bmatrix} \mathcal{B}_{n+1}^{(0)\,T} & h\mathcal{B}_{n+1}^{(1)\,T} & \dots & h^{k-1}\mathcal{B}_{n+1}^{(k-1)T} & \sum_{j=0}^{k-p} a_{j,k-p}h^j\mathbf{x}_n^{(j)T} \end{bmatrix}^T \tag{3.35}$$

Similar to LMS, we also use prediction and correction to solve for the augmented system (3.31). First, we define a stage vector $\mathbf{z}_n$ as follows

$$\mathbf{z}_n = \begin{bmatrix} \mathbf{x}_n^{(0)T} & h\mathbf{x}_n^{(1)T} & \dots & h^k\mathbf{x}_n^{(k)T} & \mathbf{x}_{n-1}^{(0)\ T} & h\mathbf{x}_{n-1}^{(1)\ T} & \dots & h^{k-p}\mathbf{x}_{n-1}^{(k-p)T} \end{bmatrix}^T \tag{3.36}$$

And a predictor can be

$$\hat{\mathbf{z}}_{n+1} = \mathbf{P}\mathbf{z}_n \tag{3.37}$$

where $\mathbf{P}$ is a square with size of $2k - p + 2$

$$\mathbf{P} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{I}_P & \mathbf{0} \end{bmatrix} \tag{3.38}$$

and $\mathbf{I}_P$ is an identity matrix of sie k-p+1.

A corretion initiated by the error vector defined as follows

$$\Phi_{n+1} = \mathbf{M}\hat{\mathbf{z}}_{n+1} - \mathbf{u}_{n+1} \tag{3.39}$$

where

$$\mathbf{M} = \begin{bmatrix} \widetilde{\mathcal{G}} + \widetilde{\mathcal{C}} & -\mathcal{D} \end{bmatrix} \tag{3.40}$$

$$\mathbf{u}_{n+1} = \begin{bmatrix} \mathbf{B}_{n+1}^{(0)\ T} & \mathbf{B}_{n+1}^{(1)\ T} & \dots & \mathbf{B}_{n+1}^{(k-1)T} \end{bmatrix}^T \tag{3.41}$$

and $\mathcal{D}$ is a block matrix that has the following matrix

$\begin{bmatrix} a_{0,k-p}\mathbf{I}_P & a_{1,k-p}\mathbf{I}_P & \dots & a_{k-p,k-p}\mathbf{I}_P \end{bmatrix}$ as its last block-row and zeros otherwise. Then the error vector Φ_{n+1} is used in iteration

$$\hat{\mathbf{z}}_{n+1,new} \leftarrow \hat{\mathbf{z}}_{n+1,old} - \left(\widetilde{\mathcal{G}} + \widetilde{\mathcal{C}}\right)^{-1} \Phi_{n+1} \tag{3.42}$$

. Linear circuits converge within one iteration.

The truncation error is determined by

$$\epsilon_{n+1} = (2k-p)!C_{k-p,k} \left\| \mathbf{K}^T(\mathbf{y}_{n+1} - \mathbf{y}_n) \right\| \tag{3.43}$$

where $\mathbf{y}_{n+1}$ is defined as

$$\mathbf{y}_{n+1} = \left[\begin{array}{cccccc} \mathbf{x}_{n+1}^{(0)\,T} & h\mathbf{x}_{n+1}^{(1)\,T} & \dots & \frac{h^k}{k!}\mathbf{x}_{n+1}^{(k)\,T} & \dots & \frac{h^{2k-p+1}}{(2k-p+1)!}\mathbf{x}_{n+1}^{(2k-p+1)^T} \end{array} \right]^T \tag{3.44}$$

, and $\mathbf{K}$ is a selector matrix with only ones in positions to select the subvector at the (2k-p) block-entry in $\mathbf{y}$.

3.3.2.2 Non-linear Circuit Simulation

For a non-linear circuit, the augmented system (3.31) is modified as follows

$$\widetilde{\mathcal{C}}\xi_{n+1} + \widetilde{\mathcal{G}}\xi_{n+1} + \widetilde{\rho}_{n+1} = \widetilde{\mathcal{B}}_{n+1} \tag{3.45}$$

where,

$$\widetilde{\rho}_{n+1} = \left[\begin{array}{ccccc} \rho_{n+1}^{(0)\,T} & \rho_{n+1}^{(1)\,T} & \dots & \rho_{n+1}^{(k-1)^T} & \mathbf{0}^T \end{array} \right]^T \tag{3.46}$$

and $\rho_{n+1}^{(i)} = h^i \frac{d^i}{dt^i}\mathbf{f}(\mathbf{x}(t))$ at t_{n+1}. At each t_{n+1}, Newton-Raphson method is required to solve for ξ_{n+1} iteratively. The error vector is modified as

$$\Phi_{n+1} = \mathbf{M}\hat{\mathbf{z}}_{n+1} + \widetilde{\rho}_{n+1} - \mathbf{u}_{n+1} \tag{3.47}$$

where $\mathbf{M}$, $\hat{\mathbf{z}}_{n+1}$ and $\mathbf{u}_{n+1}$ are referred to equations (3.40), (3.36) and (3.41). The Jacobian of the error vector is

$$\mathbf{J} = \widetilde{\mathcal{G}} + \widetilde{\mathcal{C}} + \frac{\partial \widetilde{\rho}_{n+1}}{\partial \xi_{n+1}} \tag{3.48}$$

Solving high order derivatives of non-linear expression $\mathbf{f}(\mathbf{x}(t))$ relies on the fact that i^{th} order derivative of $\mathbf{f}(\mathbf{x}(t))$ can be computed with knowledge of up to $i-1$ order derivative of $\mathbf{f}(\mathbf{x}(t))$ and up to i^{th} order derivatives of $\mathbf{x}(t)$.

For example, let $f(x(t)) = e^{x(t)}$. Expand both $x(t)$ and $f(x(t))$ to their Talor series around $t = t_{n+1}$. We get

$$x(t) = u_0 + \sum_{j=1} u_j \tau^j$$

$$f(x(t)) = v_0 + \sum_{j=1} v_j \tau^j \tag{3.49}$$

where $u_j = 1/j!h^j d^j x(t)/dt^j$, $1/j!h^j d^j f(x(t))/dt^j$ and $\tau = (t - t_{n+1})/h$. To find u_0, we compute $u_0 = x|_{\tau=0}$. And $v_0 = f|_{\tau=0} = f(u_0)$. To find v_1, we use

$$\frac{df}{d\tau} = \frac{df}{dx}\frac{dx}{d\tau}$$

From equation (3.49), we have

$$\frac{dx}{d\tau} = u_1 + \sum_{j=2} j u_j \tau^{j-1}$$

$$\frac{df}{d\tau} = v_1 + \sum_{j=2} j v_j \tau^{j-1} \tag{3.50}$$

And since

$$\frac{df}{dx} = \frac{d(e^x)}{dx} = e^x = f(x) = v_0 + \sum_{j=1} v_j \tau^j$$

, thus,

$$v_1 + \sum_{j=2} j v_j \tau^{j-1} = \left(v_0 + \sum_{j=1} v_j \tau^j\right)\left(u_1 + \sum_{j=2} j u_j \tau^{j-1}\right) \tag{3.51}$$

At $\tau = 0$, we have $v_1 = v_0 \times u_1$. To find v_j, $j = 2, 3, \ldots$, we use same techniques taking derivatives in equation (3.51) and putting $\tau = 0$. It can be shown that

$$v_j = \frac{1}{j} \sum_{m=0}^{j-1} (j - m)\, v_m \times u_{j-m} \tag{3.52}$$

Similarly, the high order derivative expression for some usual mathematical operations, such as $\times$, $\div$, log, ..., can also be found [9]. Using the similar recursive relation as (3.52), the values of $\rho_{n+1}^{(j)}$ can be computed.

In the Jacobian matrix $\partial\widetilde{\rho}_{n+1}/\partial\xi_{n+1}$ is a new item, which requires the partial derivatives of $\mathbf{f}^{(j)}(\mathbf{x}(t))$ with respect to $\mathbf{x}^{(j)}(t)$ at $t = t_{n+1}$. Those patrial derivatives for different mathematical operations can be computed using recursive relation. For example, the recursive relation of $f(x(t)) = e^{x(t)}$ is shown in (3.52). The partial derivative $\partial v_i/\partial u_j$ is given by

$$\frac{\partial v_i}{\partial u_j} = \frac{1}{i}\left(\sum_{m=0}^{i-1}(i-m)\frac{\partial v_m}{\partial u_j}u_{i-m} + v_{i-j}\right) \tag{3.53}$$

, which is also a recursive relation. Similarly, the partial derivatives for other mathematical operation can also be found recursively. The total Jacobian $\mathbf{J}$ of the error vector Φ can be built after every element is found. Then the iteration method will update the values of $\widetilde{z}_{n+1}$ as follows

$$\widetilde{z}_{n+1,new} \leftarrow \widetilde{z}_{n+1,old} - \mathbf{J}^{-1}\Phi_{n+1}$$

3.4 Summary and Discussion

This chapter presented a brief idea on the basic techniques used to solve differential equations numerically. It was intended mainly to give the reader a flavor of the computational tasks involved in performing the task of timing analysis using a dynamic approach.

The chapter presented two techniques: the first one is based on LMS method and is known as the BDF method or Gear's method. This technique is by far the most popular in circuit simulation or in connercial packages.

The other class of methods presented in this chapter is based on using high-order derivatives. A particular class of these methods is based on the Obreshkov method,

and has been used for circuit simulation quite recently in [23].

Chapter 4

Prony's Method and Modified Prony's Method

This chapter describes Prony's method, which is used to construct exponential approximations for the impulse response of a Linear Time Invariant (LTI) system, using a sample of equi-distant points of this response. The goal of this chapter is to lay the necessary background in order to show how this method may be extended to predict the future values taken by a certain waveform, using its past values or derivatives.

4.1 Description of Prony's Method

The basic idea of Prony's method is to construct an approximation for a function or a waveform in the form of sums of exponentials. Let $f(t)$ be the function that needs to be approximated. Thus one needs to find a set of residues R_i and poles λ_i such

that

$$f(t) \approx \sum_{i=1}^{n} R_i e^{\lambda_i t}. \tag{4.1}$$

Obviously, there are three sets of unknowns that must be solved for. These are the residues R_i, poles λ_i and the number of those poles (residues), n.

The process of finding those unknown in the Prony's method relies on having access to a set of equi-distant samples of $f(t)$, at points $t = t_0$, t_1, ..., t_{n-1}, where $t_i - t_{i-1} = h$, and h is the step size between any two given samples. Denote $f_i := f(t_i)$ and construct the following system of N equations.

$$\begin{aligned}
f_0 &= R_1 e^{\lambda_1 t_0} + R_2 e^{\lambda_2 t_0} + R_3 e^{\lambda_3 t_0} + \ldots + R_n e^{\lambda_n t_0} \\
f_1 &= R_1 e^{\lambda_1 t_0} z_1 + R_2 e^{\lambda_2 t_0} z_2 + R_3 e^{\lambda_3 t_0} z_3 + \ldots + R_n e^{\lambda_n t_0} z_n \\
f_2 &= R_1 e^{\lambda_1 t_0} z_1^2 + R_2 e^{\lambda_2 t_0} z_2^2 + R_3 e^{\lambda_3 t_0} z_3^2 + \ldots + R_n e^{\lambda_n t_0} z_n^2 \\
&\vdots \\
f_{N-1} &= R_1 e^{\lambda_1 t_0} z_1^{N-1} + R_2 e^{\lambda_2 t_0} z_2^{N-1} + R_3 e^{\lambda_3 t_0} z_3^{N-1} + \ldots + R_n e^{\lambda_n t_0} z_n^{N-1},
\end{aligned} \tag{4.2}$$

where $z_i = e^{\lambda_i h}$.

Assuming that n is known, or at least can be estimated based on some *a priori* information, then the above system has $2n$ unknowns. To solve for these unknowns exactly, we will have to construct $2n$ equations by stipulating that the number of samples satisfy

$$N = 2n. \tag{4.3}$$

If the number of samples N does not satisfy (4.3), then the above system can still be solved, but only in an approximate manner using the least squares method.[11]

Now assuming that N satisfies (4.3), solving (4.2) is still a difficult matter since

it is nonlinear in λ_i. This difficulty is overcome through employing the procedures described next.

- Computing the poles λ_i

  It can be shown that the samples f_0, ..., f_{N-1} satisfy the following system of equations [11]

$$\begin{aligned} f_{n-1}a_1 + f_{n-2}a_2 + \ldots + f_0 a_n &= f_n \\ f_n a_1 + f_{n-1}a_2 + \ldots + f_1 a_n &= f_{n+1} \\ &\vdots \\ f_{N-2}a_1 + f_{N-3}a_2 + \ldots + f_{N-n-1}a_n &= f_{N-1}, \end{aligned} \tag{4.4}$$

  where the unknowns a_i are the coefficients of the following polynomial

$$z^n - a_1 z^{n-1} - a_2 z^{n-2} - \ldots - a_{n-1}z - a_n = 0 \tag{4.5}$$

  and the roots of the above polynomial are the solution for the set of unknowns z_i in (4.2) whose values can be used to find λ_i using the following

$$\lambda_i = \frac{1}{h}\ln(z_i) \tag{4.6}$$

  Hence to obtain λ_i one must first solve the system (4.4) for a_i and use that solution to construct the polynomial (4.5), where upon finding the roots of the polynomial gives z_i. λ_i then follows by using (4.6).

- Computing the residues R_i

  Using the values of z_i and λ_i obtained above, the solution for the coefficient follows easily as the solution of the linear system

$$\mathbf{ZR} = \mathbf{f}_0, \tag{4.7}$$

where

$$\mathbf{Z} = \begin{bmatrix} e^{\lambda_1 t_0} & e^{\lambda_2 t_0} & \dots & e^{\lambda_n t_0} \\ e^{\lambda_1 t_0} z_1 & e^{\lambda_2 t_0} z_2 & \dots & e^{\lambda_n t_0} z_n \\ & & \vdots & \\ e^{\lambda_1 t_0} z_1^{n-1} & e^{\lambda_2 t_0} z_2^{n-1} & \dots & e^{\lambda_n t_0} z_n^{n-1} \end{bmatrix}, \mathbf{R} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \end{bmatrix}, \mathbf{f}_0 = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{bmatrix}.$$

The solution for system (4.4) can be written in a compact form as follows $a_0 = 1$ and $\mathbf{a} = -\left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{f}$ which is a valid representation for values of $N \geq 2n$, where

$$\mathbf{a} = \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} f_0 & f_1 & f_2 & \dots & f_{n-1} \\ f_1 & f_2 & f_3 & \dots & f_n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_{N-n-1} & f_{N-n} & f_{N-n+1} & \dots & f_{N-2} \end{bmatrix},$$

and

$$\mathbf{f} = \begin{bmatrix} f_n \\ f_{n+1} \\ \vdots \\ f_{N-1} \end{bmatrix}.$$

An alternative approach, known as the second Prony's method [19], is to find the eigenvector corresponding to the minimum eigenvalue of the matrix $(\mathbf{F}')^T\mathbf{F}'$ where

$$\mathbf{F}' := \begin{bmatrix} f_0 & f_1 & \dots & f_n \\ f_1 & f_2 & \dots & f_{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ f_{N-n-1} & f_{N-n} & \dots & f_N \end{bmatrix}. \tag{4.8}$$

, which is the same matrix as $\mathbf{F}$, but augmented with the column $\mathbf{f}$. Denoting such eigenvector by $\mathbf{a}' = [a'_0 \;\; a'_1 \;\; a'_2 \dots a'_n]^T$, then the coefficients a_i that represent the solution to (4.4) are obtained using

$$a_1 = \frac{a'_1}{a'_0} \quad , a_2 = \frac{a'_2}{a'_0} \quad , \ldots , a_n = \frac{a'_n}{a'_0}.$$

To demonstrate the computational steps involved in the Prony's method, we consider the attempt to reconstruct the function:

$$f(t) = 1.42e^{-t} - 1.08e^{-2t} + 1.20e^{-3t} \tag{4.9}$$

from the values for $t_i = 0, 1, 2, 3, 4, 5$ given as follows

t_i	0	1	2	3	4	5
f_i	1.54000	0.43597	0.17537	0.068169	0.025653	0.0095192

Table 4.1: Prony's Method Example

Procefure is described as follows

1. Construct the system
$$\begin{bmatrix} f_0 & f_1 & f_2 \\ f_1 & f_2 & f_3 \\ f_2 & f_3 & f_4 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} f_3 \\ f_4 \\ f_5 \end{bmatrix}$$
2. Solve the system and get the solution as
$$\begin{bmatrix} a_3 \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} 0.0024857 \\ -0.074728 \\ 0.55266 \end{bmatrix}$$
3. Compute the roots of the characteristic function $z^3 - a_1 z^2 - a_2 z - a_3 = 0$. The results are $z_1 = 0.36791$, $z_2 = 0.13453$ and $z_3 = 0.05022$.

4. Compute poles $\lambda' s$ by taking $\lambda h = \ln z$. The resultant poles are $\lambda_1 = -0.9992$, $\lambda_2 = -2.006$ and $\lambda_3 = -2.9913$.

5. Compute the residues $R' s$ by solving the following system.

$$\begin{bmatrix} e^{\lambda_1 t_0} & e^{\lambda_2 t_0} & e^{\lambda_3 t_0} \\ e^{\lambda_1 t_1} & e^{\lambda_2 t_1} & e^{\lambda_3 t_1} \\ e^{\lambda_1 t_2} & e^{\lambda_2 t_2} & e^{\lambda_3 t_2} \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1.4193 \\ -1.0943 \\ 1.215 \end{bmatrix}$$

6. Reconstruct the function by combining the poles $\lambda' s$ and $R' s$ together to get

$$\begin{aligned} f_{new}(t) &= R_1 e^{\lambda_1 t} + R_2 e^{\lambda_2 t} + R_3 e^{\lambda_3 t} \\ &= 1.4193 e^{-0.99992t} - 1.0943 e^{-2.006t} + 1.215 e^{-2.9913t} \end{aligned}$$

This example also demonstrates that in order to recover 3 poles and 3 residues from the equally spaced $f(t)$ data, 6 samples of $f(t)$ are required.

4.2 Modified Prony's Method

This section provides a modified version of Prony's method. The goal of the modified version will be to use the derivatives to construct approximations for function in the form of a sum of exponential terms.

Assume that it is required to construct an approximation for $f(t)$ in the following form

$$f(t) = \sum_{i=1}^{n} R_i e^{\lambda_i t} \tag{4.10}$$

and assume that $f^{(i)}(t_j)$ are known, $i = 0, \ldots, n$, $j = 0, \ldots, n-1$, where $f^{(i)} = \frac{d^i}{dt^i} f(t)$. The following lemma is pivotal in restoring the approximation of (4.10) from the samples and its derivatives.

Lemma 1 The set of poles λ_i that can be used to construct approximations matching $f^{(i)}(t_j)$, $i = 0, \ldots, n$, $j = 0, \ldots, n-1$ are the roots of the following polynomial

$$\lambda^n - \sum_{i=0}^{n-1} a_{n-i} \lambda^i = 0$$

where the coefficients a_{n-i} are the solution of the following system of equations

$$\begin{bmatrix} f(t_0) & f'(t_0) & f''(t_0) & \ldots & f^{(n-1)}(t_0) \\ f(t_1) & f'(t_1) & f''(t_1) & \ldots & f^{(n-1)}(t_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f(t_{n-1}) & f'(t_{n-1}) & f''(t_{n-1}) & \ldots & f^{(n-1)}(t_{n-1}) \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \end{bmatrix} = \begin{bmatrix} f^{(n)}(t_0) \\ f^{(n)}(t_1) \\ \vdots \\ f^{(n)}(t_{n-1}) \end{bmatrix} \tag{4.11}$$

Proof: To prove the above lemma we write $f^{(i)}(t)$ as shown next

$$f^{(i)}(t) = \sum_{j=1}^{n} R_j \lambda_j^i e^{\lambda_j t} \tag{4.12}$$

Multiply both sides of (4.12) by a_{n-i} for $i = 0, \ldots, n-1$ and for $i = n$ multiply both sides by -1, then add all the equations. This should result in the following

$$-f^{(n)}(t) + \sum_{i=0}^{n-1} a_{n-i} f^{(i)}(t) = -\sum_{i=1}^{n} R_i \lambda_i^n e^{\lambda_i t} + \sum_{j=0}^{n-1} \sum_{i=1}^{n} a_{n-j} R_i \lambda_i^j e^{\lambda_i t}$$

Rewrite it as

$$-f^{(n)}(t) + \sum_{i=0}^{n-1} a_{n-i} f^{(i)}(t) = \sum_{i=1}^{n} R_i e^{\lambda_i t} [-\lambda_i^n + \sum_{j=0}^{n-1} a_{n-j} \lambda_i^j]$$

Since, by assumption, λ_i, $i = 1, \ldots, n$ are the roots of the polynomial

$$-\lambda^n + \sum_{j=0}^{n-1} a_{n-j}\lambda^j,$$

it follows that

$$-f^{(n)}(t) + \sum_{i=0}^{n-1} a_{n-i} f^{(i)}(t) = 0 \tag{4.13}$$

for all t. Thus, substituting for $t = t_k$, $k = 0, \ldots, n-1$ in (4.13) shows that the coefficients a_j, $j = 0, \ldots, n-1$ must be the solution to the system in (4.11).

The previous lemma shows that it is possible to recover the poles of an approximating set of exponentials via a set of values and higher-order derivatives at a set of discrete points. This lemma, however, shows that the number of high-order derivatives must be equal to the number of time-samples. The following lemma proves that the same can be achieved by using only one high-order derivatives at equally spaced time points.

Lemma 2 Assume that the m^{th}-order derivative $f^{(m)}(t_j)$ is available at equally spaced time points $t = t_j$, $j = 0, \ldots, N-1$ where $N = 2n$. Then the set λ_i used to construct the approximation (4.10) can be derived as $\lambda_i = \ln(z_i)/h$, where h is the constant step size between any two given samples and z_i are the roots of the following polynomial

$$-z^n + \sum_{i=0}^{n-1} a_{n-i} z^i,$$

in which a_{n-i} are the coefficients obtained by solving the following linear system

$$\begin{bmatrix} f^{(m)}(t_0) & f^{(m)}(t_1) & \ldots & f^{(m)}(t_{n-1}) \\ f^{(m)}(t_1) & f^{(m)}(t_2) & \ldots & f^{(m)}(t_n) \\ \vdots & \vdots & \vdots & \vdots \\ f^{(m)}(t_{N-n-1}) & f^{(m)}(t_{N-n}) & \ldots & f^{(m)}(t_{N-2}) \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \end{bmatrix} = \begin{bmatrix} f^{(m)}(t_n) \\ f^{(m)}(t_{n+1}) \\ \vdots \\ f^{(m)}(t_{N-1}) \end{bmatrix} \tag{4.14}$$

Proof: To prove the above lemma, we write $f^{(m)}(t_j)$ as shown next

$$f^{(m)}(t_j) = \sum_{i=1}^{n} R_i \lambda_i^m e^{\lambda_i t_0} z_i^j, \tag{4.15}$$

where $z_i = e^{\lambda_i h}$ and $j = 0, \ldots, N-1$. First we choose the first $n+1$ sets of equations only, i.e. $j = 0, \ldots, n$. Multiply both side of (4.15) by a_{n-j} for $j = 0, \ldots, n-1$ and for $j = n$ multiply both sides by -1, then add all the $n+1$ equations. This should result in the following

$$-f^{(m)}(t_n) + \sum_{i=0}^{n-1} a_{n-i} f^{(m)}(t_i) = -\sum_{i=1}^{n} R_i \lambda_i^m e^{\lambda_i t_0} z_i^n + \sum_{j=0}^{n-1} \sum_{i=1}^{n} a_{n-j} R_i \lambda_i e^{\lambda_i t_0} z_i^j$$

Rewrite it as

$$-f^{(m)}(t_n) + \sum_{i=0}^{n-1} a_{n-i} f^{(m)}(t_i) = \sum_{i=1}^{n} R_i \lambda_i^m e^{\lambda_i t_0} [-z_i^n + \sum_{j=0}^{n-1} a_{n-j} z_i^j]$$

Since, by assumption z_i, $i = 1, \ldots, n$ are the roots of the polynomial

$$-z^n + \sum_{j=0}^{n-1} a_{n-j} z^j,$$

it follows that

$$-f^{(m)}(t_n) + \sum_{i=0}^{n-1} a_{n-i} f^{(m)}(t_i) = 0 \tag{4.16}$$

which represents the first equation in (4.14).

By choosing the samples t_j, $j = 1, \ldots, n+1$, we obtain the second equation. In a similar manner, it is possible to construct the remaining equations in (4.14) which proves the lemma.

It should be noted that for the more general case where $N > 2n$, the coefficients a_i's can be obtained using the method of least squares, where $\mathbf{a}$ can be described as $\mathbf{a} = -\left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{f}$ with

$$\mathbf{a} = \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \end{bmatrix}, \qquad \mathbf{f} = \begin{bmatrix} f^{(m)}(t_n) \\ f^{(m)}(t_{n+1}) \\ \vdots \\ f^{(m)}(t_{N-1}) \end{bmatrix}$$

and

$$\mathbf{F} = \begin{bmatrix} f^{(m)}(t_0) & f^{(m)}(t_1) & f^{(m)}(t_2) & \dots & f^{(m)}(t_{n-1}) \\ f^{(m)}(t_1) & f^{(m)}(t_2) & f^{(m)}(t_3) & \dots & f^{(m)}(t_n) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f^{(m)}(t_{N-n-1}) & f^{(m)}(t_{N-n}) & f^{(m)}(t_{N-n+1}) & \dots & f^{(m)}(t_{N-2}) \end{bmatrix} \tag{4.17}$$

The proof of Lemma 2 does not require that the orders of the derivatives at different rows have to be the same, which directly leads to the following lemma.

Lemma 3 Assume that from 0 up to p^{th}-order derivative $f^{(p)}(t_j)$ are all available at equally spaced time points $t = t_j$, $j = 0, \dots, N$. Then the set λ_i used to construct the approximation (4.10) can be derived as $\lambda_i = \ln(z_i)/h$, where h is the constant step size between any two given samples and z_i are the roots of the following polynomial

$$-z^n + \sum_{i=0}^{n-1} a_{n-i} z^i,$$

in which a_{n-i} are the coefficients obtained by solving the following linear system

$$\begin{bmatrix} f^{(k_1)}(t_0) & f^{(k_1)}(t_1) & \dots & f^{(k_1)}(t_{n-1}) \\ f^{(k_1)}(t_1) & f^{(k_1)}(t_2) & \dots & f^{(k_1)}(t_n) \\ \vdots & \vdots & \vdots & \vdots \\ f^{(k_1)}(t_{N-n-1}) & f^{(k_1)}(t_{N-n}) & \dots & f^{(k_1)}(t_{N-2}) \\ \vdots & \vdots & \vdots & \vdots \\ f^{(k_i)}(t_0) & f^{(k_i)}(t_1) & \dots & f^{(k_i)}(t_{n-1}) \\ f^{(k_i)}(t_1) & f^{(k_i)}(t_2) & \dots & f^{(k_i)}(t_n) \\ \vdots & \vdots & \vdots & \vdots \\ f^{(k_i)}(t_{N-n-1}) & f^{(k_i)}(t_{N-n}) & \dots & f^{(k_i)}(t_{N-2}) \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \\ \\ \vdots \\ \\ \\ \\ a_1 \end{bmatrix} = \begin{bmatrix} f^{(k_1)}(t_n) \\ f^{(k_1)}(t_{n+1}) \\ \vdots \\ f^{(k_1)}(t_{N-1}) \\ \vdots \\ f^{(k_i)}(t_n) \\ f^{(k_i)}(t_{n+1}) \\ \vdots \\ f^{(k_i)}(t_{N-1}) \end{bmatrix}, \tag{4.18}$$

where $0 \leq k_i \leq p$. In order to make the system square, it is also clear the number of required derivatives that ensure the system is square is given by

$$q = \left\lceil \frac{n}{N-n} \right\rceil \tag{4.19}$$

where n is the number of poles/residues and N is the number of available time samples.

Lemma 1 demonstrates that constructing the approximation can be done by using a number of derivatives equal to the number of time-samples; ***Lemma 2*** demonstrates that the same approximation can be done by using a single high-order derivatives at a number of samples that is twice the number of samples; and ***Lemma 3*** demonstrates that constructing the Prony's approximation is possible by using a number of samples N that is less than $2n$, provided that appropriate number of high-order derivatives are available at those samples.

Discussion The idea of using less samples with high-order derivatives represents the main motivation for using modified Prony's method as a predictor for rather than a means to model the behavior of the system. Chapter 5 provides the analysis of the performance of the Prony's method as a predictor.

4.2.1 Procedure for Modified Prony's Method

Suppose all necessary derivatives at all necessary time points are given. We want to construct the approximation (4.10) from the samples and derivatives. The procedure can be described as the following steps

1. Choose a model order n;

2. Compute the poles $\lambda's$ using one of the lemmas;

3. Compute the residues $R's$ by solving the following system. Note, although, in theory, many system may be constructed to solve for $R's$, try to use only the values of 0^{th} order derivatives;

$$\begin{bmatrix} e^{\lambda_1 t_0} & e^{\lambda_2 t_0} & \dots & e^{\lambda_n t_0} \\ e^{\lambda_1 t_1} & e^{\lambda_2 t_1} & \dots & e^{\lambda_n t_1} \\ \vdots & \vdots & \vdots & \vdots \\ e^{\lambda_1 t_{n-1}} & e^{\lambda_2 t_{n-1}} & \dots & e^{\lambda_n t_{n-1}} \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \end{bmatrix} = \begin{bmatrix} f(t_0) \\ f(t_1) \\ \vdots \\ f(t_{n-1}) \end{bmatrix}$$

4. The approximation (4.10) can be re-constructed as

$$f_{new}(t) = R_1 e^{\lambda_1 t} + R_2 e^{\lambda_2 t} + \ldots + R_n e^{\lambda_n t}.$$

4.2.2 Examples

There are example A and B presented in this subsection to describe the procedure using Lemma 1 and Lemma 3 respectively. The procedure of using Lemma 2 is same as the procedure of using Lemma 3, except that using Lemma 2 takes only a single high-order derivatives at a number of samples.

Example A: This example will demonstrate the procedure needed to reconstrcut the approximation (4.10) using Lemma 1. Let's consider the same example function as in section 4.1: $f(t) = 1.42e^{-t} - 1.08e^{-2t} + 1.20e^{-3t}$. It has 1^{st} , 2^{nd} and 3^{rd} order derivatives as the following forms:

$$\begin{aligned} f'(t) &= -1.42e^{-t} + 2.16e^{-2t} - 3.60e^{-3t} \\ f''(t) &= 1.42e^{-t} - 4.32e^{-2t} + 10.80e^{-3t} \\ f'''(t) &= -1.42e^{-t} + 8.64e^{-2t} - 32.40e^{-3t}. \end{aligned}$$

We attempt to reconstruct the function

$$f(t) = 1.42e^{-t} - 1.08e^{-2t} + 1.20e^{-3t} \tag{4.20}$$

from the derivatives for $t_i = 0, 1, 2$ as given in the following table

t_i	0	1	2
$f(t_i)$	1.54000	0.43597	0.17537
$f'(t_i)$	-2.86000	-0.40930	-0.16154
$f''(t_i)$	7.90000	0.47544	0.13982
$f'''(t_i)$	-25.18000	-0.96619	-0.11424

Table 4.2: Modified Prony's Method Example A

Procedure is described as the following steps

1. Construct the system
$$\begin{bmatrix} f(0) & f'(0) & f''(0) \\ f(1) & f'(1) & f''(1) \\ f(2) & f'(2) & f''(2) \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} f'''(0) \\ f'''(1) \\ f'''(2) \end{bmatrix}$$
2. Solve the system and get the solution as
$$\begin{bmatrix} a_3 \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} -5.99790 \\ -10.99700 \\ -5.99930 \end{bmatrix}$$
3. Compute the roots of the characteristic function $\lambda^3 - a_1\lambda^2 - a_2\lambda - a_3 = 0$. The resultant poles are $\lambda_1 = -1.0002$, $\lambda_2 = -1.9987$ and $\lambda_3 = -3.0004$.

4. Compute the residues $R's$ by solving the following system.
$$\begin{bmatrix} e^{\lambda_1 t_0} & e^{\lambda_2 t_0} & e^{\lambda_3 t_0} \\ e^{\lambda_1 t_1} & e^{\lambda_2 t_1} & e^{\lambda_3 t_1} \\ e^{\lambda_1 t_2} & e^{\lambda_2 t_2} & e^{\lambda_3 t_2} \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} f(t_0) \\ f(t_1) \\ f(t_2) \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1.4208 \\ -1.0797 \\ 1.1988 \end{bmatrix}$$

5. Reconstruct the function by combining the poles $\lambda' s$ and $R' s$ together to get

$$\begin{aligned} f_{new}(t) &= R_1 e^{\lambda_1 t} + R_2 e^{\lambda_2 t} + R_3 e^{\lambda_3 t} \\ &= 1.4208 e^{-1.0002t} - 1.0797 e^{-1.9987t} + 1.1988 e^{-3.0004t} \end{aligned}$$

This example also demonstrates that in order to recover 3 poles and 3 residues from the data of $f(t)$, $f'(t)$, $f''(t)$ and $f'''(t)$, only 3 samples of those derivatives are required. Whereas, the original Prony's method needs 6 samples of $f(t)$.

Example B: This example will demonstrate the procedure needed reconstruct the approximation (4.10) using Lemma 3. Let's consider the same function as $f(t) = 1.42e^{-t} - 1.08e^{-2t} + 1.20e^{-3t}$. It has 1^{st} and 2^{nd} order derivatives as the following forms:

$$f'(t) = -1.42e^{-t} + 2.16e^{-2t} - 3.60e^{-3t}$$

$$f''(t) = 1.42e^{-t} - 4.32e^{-2t} + 10.80e^{-3t}.$$

We attempt to reconstruct the function

$$f(t) = 1.42e^{-t} - 1.08e^{-2t} + 1.20e^{-3t} \tag{4.21}$$

from the derivatives for $t_i = 0, 1, 2, 3$ as given in the following table

t_i	0	1	2	3
$f(t_i)$	1.54000	0.43597	0.17537	0.068169
$f'(t_i)$	-2.86000	-0.40930	-0.16154	-0.065788
$f''(t_i)$	7.90000	0.47544	0.13982	0.061322

Table 4.3: Modified Prony's Method Example B

Procedure is described as the following steps

1. Construct the system
$$\begin{bmatrix} f(0) & f(1) & f(2) \\ f'(0) & f'(1) & f'(2) \\ f''(0) & f''(1) & f''(2) \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} f(3) \\ f'(3) \\ f''(3) \end{bmatrix}$$
2. Solve the system and get the solution as
$$\begin{bmatrix} a_3 \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} 0.0024788 \\ -0.074779 \\ 0.55286 \end{bmatrix}$$
3. Compute the roots of the characteristic function $z^3 - a_1 z^2 - a_2 z - a_3 = 0$. The results are $z_1 = 0.36791$, $z_2 = 0.1351$ and $z_3 = 0.04985$.

4. Compute poles $\lambda' s$ by taking $\lambda h = \ln z$. The resultant poles are $\lambda_1 = -0.99992$, $\lambda_2 = -2.0017$ and $\lambda_3 = -2.9987$.

5. Compute the residues $R' s$ by solving the following system.
$$\begin{bmatrix} e^{\lambda_1 t_0} & e^{\lambda_2 t_0} & e^{\lambda_3 t_0} \\ e^{\lambda_1 t_1} & e^{\lambda_2 t_1} & e^{\lambda_3 t_1} \\ e^{\lambda_1 t_2} & e^{\lambda_2 t_2} & e^{\lambda_3 t_2} \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} f(t_0) \\ f(t_1) \\ f(t_2) \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1.42 \\ -1.08 \\ 1.2 \end{bmatrix}$$

6. Reconstruct the function by combining the poles $\lambda' s$ and $R' s$ together to get

$$\begin{aligned} f_{new}(t) &= R_1 e^{\lambda_1 t} + R_2 e^{\lambda_2 t} + R_3 e^{\lambda_3 t} \\ &= 1.42e^{-0.99992t} - 1.08e^{-2.0017t} + 1.2e^{-2.9987t} \end{aligned}$$

This example also demonstrates that in order to recover 3 poles and 3 residues from the $f(t)$, $f'(t)$ and $f''(t)$ data, only 4 samples of those derivatives are required. On the contrary, the original Prony's method requires 6 samples of $f(t)$.

4.3 Prony's Method and Linear System

It is important to note that the Prony's method identifies a model of signal, not a linear system or the transfer function of a linear system. Suppose we have a linear system with simple poles and it's impulse response is available, then it is possible to express the impulse response in term of

$$I(t) = R_1 e^{\lambda_1 t} + R_2 e^{\lambda_2 t} + \ldots + R_n e^{\lambda_n t} \tag{4.22}$$

Since an impulse function usually is not a practical exciting signal for a linear system, it is of interest to determine the form of the transient response due to an arbitrary exciting waveform. A general response function $r(t)$ is given as

$$r(t) = \sum_{m=1}^{n} B_m e^{\lambda_m t} + g(t) \tag{4.23}$$

where B_m contains the R_m multiplied by some influence of the driving function, and the added term $g(t)$ is dependent on the driving function. (4.23) shows that in general the transient response due to an arbitrary exciting waveform can be written as a sum of complex exponentials plus some added term $g(t)$. If the exciting waveform itself has pole singularities, as in the case of a step function or a sinusoidal function, then the $g(t)$ term also may be expressed as a sum of exponentials [14]. If the exciting waveform is of finite duration, that is it is turned off after some time t_0, then it may be shown that the term $g(t)$ is identically zero for $t > t_0$ [5]. The input signals for digital circuits are in this category. Thus, Prony's method may be applied to transient output responses of digital circuits.

4.4 Problems Ascotiated with Prony's Method

Although Prony's method is a powerful tool in signal analysis and system identification, it has some limitations. Noise, a linear trend, or a signal mean in the raw data can cause significant errors in the Prony's analysis results [18] [17] [6]. Hence, it is important to pre-process the data before applying the Prony method. Here are some basic methods. For example, one needs to analyze signal interfered with Gausian noise. To reduce the impact of noise, data should be filtered to isolate the signal components by frequency. And the second Prony's method should be applied. In order to remove the linear trend and the mean from the signal, one should compute the least-squares fit of a straight line to the data and subtracts the resulting function from the data [16]. Fortunately, all of those three are not big issues in digital circuit simulation.

Prony's method requires some information about the system be identified, which is the model order. In practice, the model order is unknown. As there is no straightforward method to compute the model order of an arbitrary system, an estimate of model order can be obtained from the user [12].

Chapter 5

Applications of the Prony's Method to the Problem of Delay Analysis in Digital Circuits

This chapter investigates using the idea of Prony's method in the delay analysis of digital circuits. The goal of this chapter is two-fold. We first seek to demonstrate implement modified Prony's method in dynamic timing analysis. We then consider gearing the basic Prony's method in static timing analysis.

5.1 Prony's Method for DTA

Our goal here is to use the differential equations describing the entire circuit to predict basic timing parameters such as the delay, rise or fall time of a digital pulse.

First recall that the Prony's method seeks to capture a general waveform in the

form of sums of exponentials. Hence, it is an ideal choice if the waveform to be captured is the response of a linear system to a certain stimulus.

Unfortunately, this is not the case as for as digital circuits are concerned, since those circuits are designed to behave strongly in a nonlinear manner.

However, the thesis assumes that the behavior of the digital or logic gate can be described as a linear system at least locally in the neighbourhood of a rising or falling input signal. This assumption may be justified by the fact that the main interest in this thesis is focused on predicting the delay or rise/fall time, which typically occurs in the transition from one state to the other.

The idea of capturing the response or at least the transition in the response of a digital gate is carried out through combining the modified Prony's method prepared in the previous chapter with a high-order method used to solve the circuit differential equations. More precisely, we use the high-order derivatives obtained at the output of each step taken by the high-order method for the first few time steps to construct an approximation for the waveform in the form of sums of exponentials, which is used to predict the values of the output waveform at future time points. The output derivatives are employed as indicated by ***Lemma 3*** of the previous chapter to obtain the poles and residues of the exponential approximation.

However, a note on adapting the output of the high-order method to the current task is its order. Recall that the obtained high-order derivatives are always obtained scaled by power of h, the time-step, while ***Lemma 3*** requires only the raw high-order derivatives. Although it is possible to descale the output of the high-order solver by h to recover the raw values, those raw values have very large magnitudes and using them directly could lead to numerical stability issue. Fortunately, ***Lemma 3*** can

handle the output of high-order solver without any modifications. The following paragraph will demonstrate the reason why ***Lemma 3*** can be directly applied to scaled high-order derivatives.

Define the output of the high-order solver as $f_s^{(k_i)}$ and the corresponding raw derivatives as $f_r^{(k_i)}$. It is clear that $f_s^{(k_i)} = h^{k_i} f_r^{(k_i)}$. Recall ***Lemma 3***. The a_i's can be determined by the following linear system

$$\begin{bmatrix} f_r^{(k_1)}(t_0) & f_r^{(k_1)}(t_1) & \dots & f_r^{(k_1)}(t_{n-1}) \\ f_r^{(k_1)}(t_1) & f_r^{(k_1)}(t_2) & \dots & f_r^{(k_1)}(t_n) \\ \vdots & \vdots & \vdots & \vdots \\ f_r^{(k_1)}(t_{N-n-1}) & f_r^{(k_1)}(t_{N-n}) & \dots & f_r^{(k_1)}(t_{N-2}) \\ \vdots & \vdots & \vdots & \vdots \\ f_r^{(k_i)}(t_0) & f_r^{(k_i)}(t_1) & \dots & f_r^{(k_i)}(t_{n-1}) \\ f_r^{(k_i)}(t_1) & f_r^{(k_i)}(t_2) & \dots & f_r^{(k_i)}(t_n) \\ \vdots & \vdots & \vdots & \vdots \\ f_r^{(k_i)}(t_{N-n-1}) & f_r^{(k_i)}(t_{N-n}) & \dots & f_r^{(k_i)}(t_{N-2}) \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \\ \\ \vdots \\ \\ \\ \\ a_1 \end{bmatrix} = \begin{bmatrix} f_r^{(k_1)}(t_n) \\ f_r^{(k_1)}(t_{n+1}) \\ \vdots \\ f_r^{(k_1)}(t_{N-1}) \\ \vdots \\ f_r^{(k_i)}(t_n) \\ f_r^{(k_i)}(t_{n+1}) \\ \vdots \\ f_r^{(k_i)}(t_{N-1}) \end{bmatrix}, \quad (5.1)$$

or

$$\begin{bmatrix} \frac{1}{h^{k_1}} f_s^{(k_1)}(t_0) & \frac{1}{h^{k_1}} f_s^{(k_1)}(t_1) & \dots & \frac{1}{h^{k_1}} f_s^{(k_1)}(t_{n-1}) \\ \frac{1}{h^{k_1}} f_s^{(k_1)}(t_1) & \frac{1}{h^{k_1}} f_s^{(k_1)}(t_2) & \dots & \frac{1}{h^{k_1}} f_s^{(k_1)}(t_n) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{h^{k_1}} f_s^{(k_1)}(t_{N-n-1}) & \frac{1}{h^{k_1}} f_s^{(k_1)}(t_{N-n}) & \dots & \frac{1}{h^{k_1}} f_s^{(k_1)}(t_{N-2}) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{h^{k_i}} f_s^{(k_i)}(t_0) & \frac{1}{h^{k_i}} f_s^{(k_i)}(t_1) & \dots & \frac{1}{h^{k_i}} f_s^{(k_i)}(t_{n-1}) \\ \frac{1}{h^{k_i}} f_s^{(k_i)}(t_1) & \frac{1}{h^{k_i}} f_s^{(k_i)}(t_2) & \dots & \frac{1}{h^{k_i}} f_s^{(k_i)}(t_n) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{h^{k_i}} f_s^{(k_i)}(t_{N-n-1}) & \frac{1}{h^{k_i}} f_s^{(k_i)}(t_{N-n}) & \dots & \frac{1}{h^{k_i}} f_s^{(k_i)}(t_{N-2}) \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \\ \\ \vdots \\ \\ \\ \\ a_1 \end{bmatrix} = \begin{bmatrix} \frac{1}{h^{k_1}} f_s^{(k_1)}(t_n) \\ \frac{1}{h^{k_1}} f_s^{(k_1)}(t_{n+1}) \\ \vdots \\ \frac{1}{h^{k_1}} f_s^{(k_1)}(t_{N-1}) \\ \vdots \\ \frac{1}{h^{k_i}} f_s^{(k_i)}(t_n) \\ \frac{1}{h^{k_i}} f_s^{(k_i)}(t_{n+1}) \\ \vdots \\ \frac{1}{h^{k_i}} f_s^{(k_i)}(t_{N-1}) \end{bmatrix}. \tag{5.2}$$

It can be simplified as

$$\begin{bmatrix} f_s^{(k_1)}(t_0) & f_s^{(k_1)}(t_1) & \dots & f_s^{(k_1)}(t_{n-1}) \\ f_s^{(k_1)}(t_1) & f_s^{(k_1)}(t_2) & \dots & f_s^{(k_1)}(t_n) \\ \vdots & \vdots & \vdots & \vdots \\ f_s^{(k_1)}(t_{N-n-1}) & f_s^{(k_1)}(t_{N-n}) & \dots & f_s^{(k_1)}(t_{N-2}) \\ \vdots & \vdots & \vdots & \vdots \\ f_s^{(k_i)}(t_0) & f_s^{(k_i)}(t_1) & \dots & f_s^{(k_i)}(t_{n-1}) \\ f_s^{(k_i)}(t_1) & f_s^{(k_i)}(t_2) & \dots & f_s^{(k_i)}(t_n) \\ \vdots & \vdots & \vdots & \vdots \\ f_s^{(k_i)}(t_{N-n-1}) & f_s^{(k_i)}(t_{N-n}) & \dots & f_s^{(k_i)}(t_{N-2}) \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \\ \\ \vdots \\ \\ \\ \\ a_1 \end{bmatrix} = \begin{bmatrix} f_s^{(k_1)}(t_n) \\ f_s^{(k_1)}(t_{n+1}) \\ \vdots \\ f_s^{(k_1)}(t_{N-1}) \\ \vdots \\ f_s^{(k_i)}(t_n) \\ f_s^{(k_i)}(t_{n+1}) \\ \vdots \\ f_s^{(k_i)}(t_{N-1}) \end{bmatrix}. \tag{5.3}$$

(5.3) gives the same solution of a_i's as what in (5.1), but (5.3) directly uses the output of the high-order solver. Once $a_i's$ have been computed based on (5.3), the poles of the response and the residues R_i can be computed as explained in Chapter 4.

5.2 Prony's Method and STA Delay Models

As mentioned in Chapter 2, the classical STA timing analysis is based on storing the delay models for digital circuits in the form of delay-lookup tables, which contain only the rise/fall delay and output slew for each logic gate. These three parameters are retrieved for given input and output conditions to represent the response at the output of the digital logic gate. The accuracy of this approach clearly depends on capturing the output waveforms through those three parameters.

To improve the accuracy, the Weibull CDF was proposed to represent the input waveform. Being a two-parameter waveform, Weibull CDF promises to offer higher capabilities in capturing a general waveforms.

This thesis proposes using Prony's method to develop an approximation for the waveform at the output of the digital gate. The Prony's approximation will take the form of a finite set of poles and residues that can be accessed a contiguous entry in a table for given input and output conditions. The fact that the waveform is being stored as a set of poles/residues should allow a greater flexibility to capture more complicated waveforms at the gate's output.

In the next subsection, we illustrate the main idea here by considering its application to the cascaded inverter of Figure 5.1.

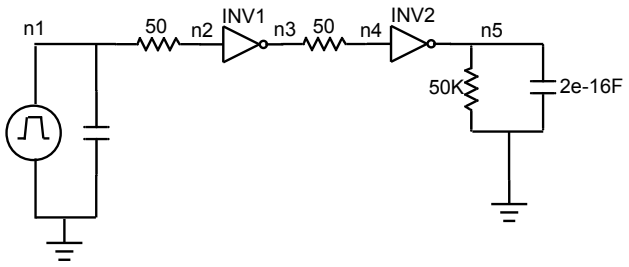

Figure 5.1: Two cascaded inverters

5.2.1 Illustration of Using Prony's Method for STA

We consider here two cases, for representing the waveform at the input of the gate.

- Case 1: Single Parameter Input

  In this case, the input waveform used is saturated ramp given as follows,

$$r(t)/V_{DD} = \begin{cases} 0, t \leq t_{50\%} - 0.5t_s \\ \frac{t - t_{50\%} + 0.5t_s}{t_s}, t_{50\%} - 0.5t_s \leq t \leq t_{50\%} + 0.5t_s \\ 1, t \geq t_{50\%} + 0.5t_s \end{cases}, \tag{5.4}$$

  where $t_{50\%}$ stands for time at which the signal reach $0.5V_{DD}$; t_s stands for time slew. The following Figure 5.2 gives the saturated ramp signal with V_{DD}=1.2V, $t_{50\%}$=0.1ns and t_s=0.2ns.

  The output waveform for each gate is stored as a group of poles/residues. Each group can be thought of a single entry in a two-dimensional table array that stores those entries corresponding to a certain input slew $t_{s_{in}}$ and output load capacitance C_L. Once those input and output conditions are determined, the corresponding group of poles and residues is retrieved to represent the output waveform.

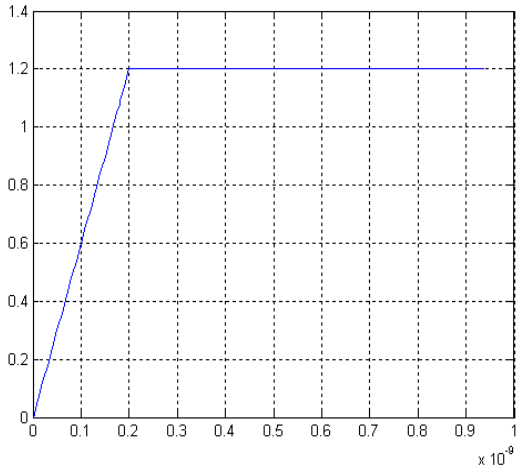

Figure 5.2: A saturated ramp signal

If the output waveform represents the rising edge, then the output waveform is constructed as follows,

$$f_r(t) = V_{DD} + \sum_{i=1}^{N} R_i e^{\lambda_i t}$$

where λ_i and R_i are the poles and residues representing the waveform for the particular input and output conditions.

On the other hand, if the output waveform represents the falling edge, then it can be described analytically as follows,

$$f_f(t) = \sum_{i=1}^{N} R_i e^{\lambda_i t}$$

The next step is to use the output waveform obtained above to extract the information required to access and reconstruct the output waveform at the next logical gate. Given that, this information is stored for a given input slew $t_{s_{in}}$ and output capacitance C_L, then the output slew at the previous gate will need to be computed. For that purpose, Newton-Raphson iterations are used to find

the output slew, $t_{s_{out}}$, which is defined as

$$t_{s_{out}} = |t_{10\%} - t_{90\%}|$$

where $t_{10\%}$ and $t_{90\%}$ are defined as

$$f_r(t_{10\%}) = 0.1V_{DD}$$

$$f_r(t_{90\%}) = 0.9V_{DD}$$

for a rising edge, or

$$f_f(t_{10\%}) = 0.9V_{DD}$$

$$f_f(t_{90\%}) = 0.1V_{DD}$$

for a falling edge.

Newton-Raphson iterations can be invoked to determine $t_{10\%}$ and $t_{90\%}$ as defined by the above nonlinear equations.

Once the output slew is determined and the output capacitance for the second gate is defined, the set of poles/residues corresponding to the gate output is retrieved and the output signal is reconstructed as a sum of exponentials.

- Case 2: Two-Parameter Input

  In this case we assume that the input to the gate is a waveform with two parameters such as the Weibull waveform defined by

  $$WB(t, \alpha, \beta)/Vdd = 1 - e^{-(\frac{t}{\beta})^{\alpha}} \tag{5.5}$$

  An example of a Weibull waveform habing $\alpha = 1.46$ and $\beta = 1.07 \times 10^{-10}$ is given in Figure 5.3.

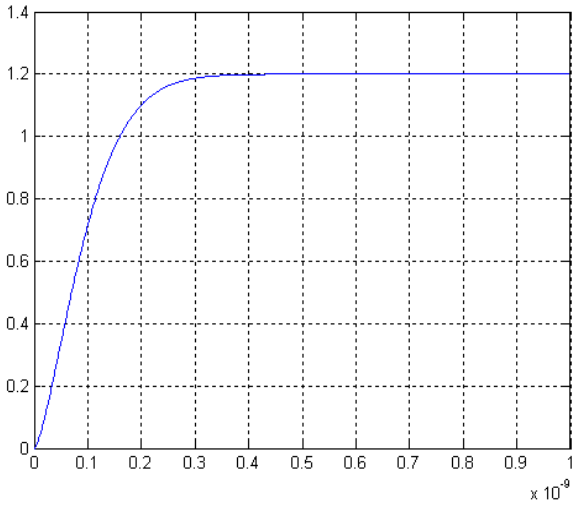

Figure 5.3: A Weibull waveform

Representing a general waveform obtained from circuit simulation is typically carried out using the following steps.[3]

1. Obtain time-value (t, v) waveform from circuit simulation.
2. Normalize by V_{DD}.
3. If (t, v) represents a falling transition, set $v = 1 - v$.
4. Perform a linear regression (least-squares method) on $y = b + ax$ to obtain b and a, where $y = \ln(t)$, $x = \ln(\ln(\frac{1}{1-v}))$.
5. Obtain α and β using $\alpha = 1/a$ and $b = \ln(\beta)$

Moreover, slew time t_s can be obtained analytically as:[3]

$$t_s = t_{0.9} - t_{0.1} = \beta((\ln(10))^{1/\alpha} - (\ln(1.11))^{1/\alpha}) \tag{5.6}$$

In addition to the output capacitance C_L and input slew $t_{s_{in}}$ used to index the entries of each gate, the parameter α is used to index the entries. This

makes the tables used in this case three-dimensional tables. Similar to the two-dimensional look-up tables used with single parameter case, the Prony-based waveform look-up table stores the whole waveform at the output in the form of sets of poles/residues. Thus for a given shape factor α, input slew $t_{s_{in}}$ and output capacitance C_L, the table will produce a set of poles and residues that best approximates the output waveform.

Chapter 6

Numerical Results

This chapter includes some examples that demonstrate: 1, modified Prony's method is qualified to perform signal analysis as the original Prony's method is, but less steps are required; 2, modified Prony's method can use a few sampling steps to model the transient state of a digital output; 3, Prony's approximation can be used as delay models in STA. Let's start from the DTA application.

6.1 Modified Prony's Method for DTA

This section summarizes the numerical experiments performed to test the Mdified Prony's method developed in Chapter 4 to the problem of Dynamic Timing Analysis(DTA). We first start by demonstrating the performance of the modified Prony's method on simple linear circuits, where the response can be obtained analytically in a closed-form.

6.1.1 Application to Linear Circuits

As a proof of concept, we consider the linear circuit shown in Figure 6.1, where the input signal is current source given as follows

$$I(t) = 0.01 + 0.1\cos 50 \times 10^3 t(mA)$$

while the constant in the circuit elements are given by

$$K_1 = 1\times 10^6,\ K_2 = 2\times 10^6,\ K_3 = 3\times 10^6,\ K_4 = 4\times 10^6,\ K_5 = 5\times 10^6,\ K_6 = 6\times 10^6$$

$$P_1 = 150,\ P_2 = 300,\ P_3 = 400,\ P_4 = 500,\ P_5 = 600,\ P_6 = 7000$$

The output of this circuit is defined as the voltage across the current source.

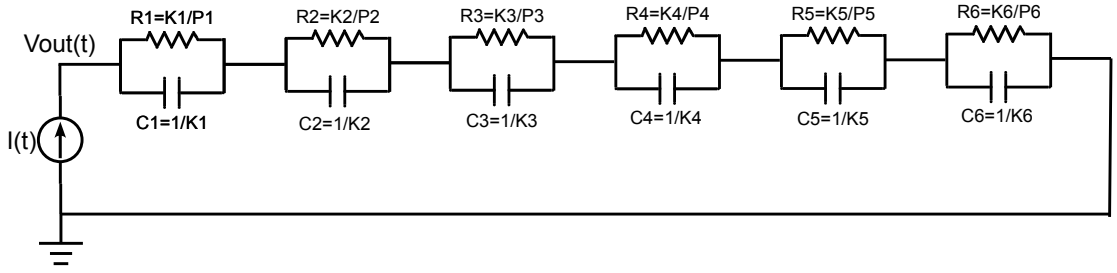

Figure 6.1: A Linear Circuit Example

Thus, the trasient output voltage of the circuit has the following shape in Figure 6.2.

6.1.1.1 Modified Method No. 1

This method needs 0^{th} order derivative up to n^{th} order derivatives at n sampling points. The sampling step size is chosen as 0.005. The following poles have been generated. $P_1 = -1.5848\times 10^{-10} + j50000$, $P_2 = -1.5848\times 10^{-10} - j50000$, $P_3 = -7000$,

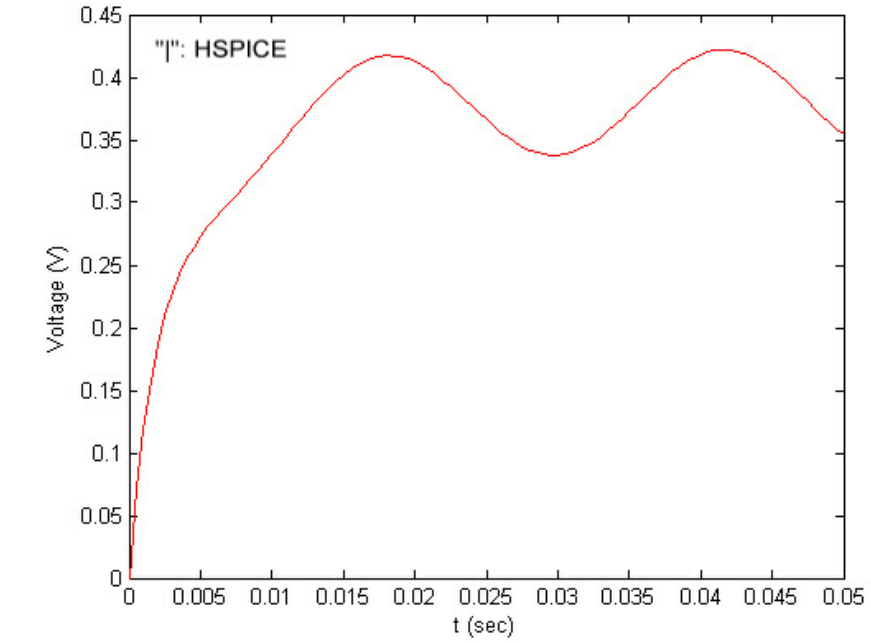

Figure 6.2: The output of the linear circuit

$P_4 = -583.99$, $P_5 = -394.78$ and $P_6 = -164.33$. And their corresponding residues are $R_1 = 0.001 - j0.0209$, $R_2 = 0.001 + j0.0209$, $R_3 = -0.0102$, $R_4 = -0.1230$, $R_5 = -0.1622$ and $R_6 = -0.0868$. After the pole-residue model has been constructed, it can be used to predict the future value of the transient output as shown in Figure 6.3, where "*" stand for the output of the pole-residue model, the solid curve stands for the actual output and the vertical line indicates up to what step the derivatives have been used.

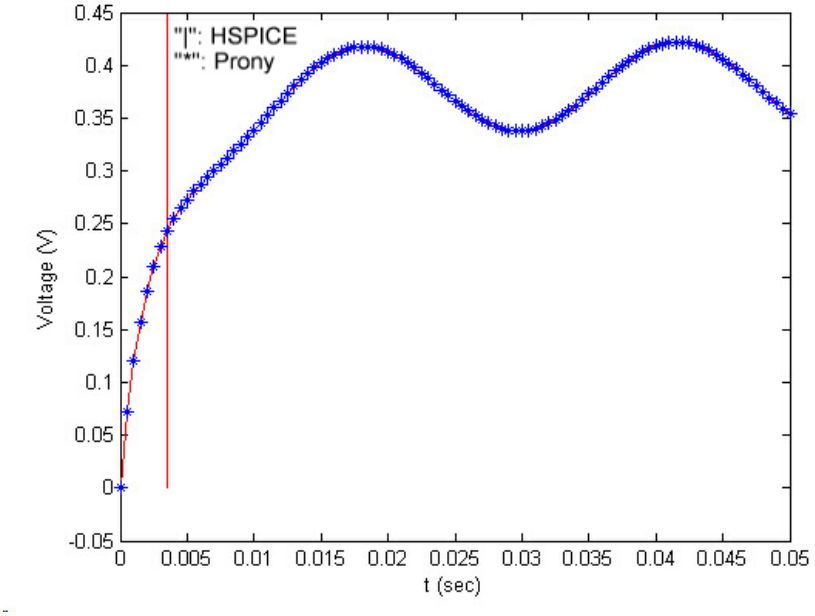

Figure 6.3: The output using modified method No.1

6.1.1.2 Modified Method No. 2

In this example, 0^{th} order derivatives up to 4^{th} order derivatives are used. Each of them contributes 11 samples. The sampling step is 0.005. In this example, we expect stable poles. Therefore, 1 positive poles are discarded, so that 7 resultant poles are $P_1 = -3.1544 \times 10^{-8} + j265.48$, $P_2 = -3.1544 \times 10^{-8} - j265.48$, $P_3 = -159.06$, $P_4 = -353.84$, $P_5 = -504.41$, $P_6 = -603.21$ and $P_7 = -7000$. And their corresponding residues are $R_1 = 0.0010 - j0.0209$, $R_2 = 0.0010 + j0.0209$, $R_3 = -0.0772$, $R_4 = -0.1208$, $R_5 = -0.098$, $R_6 = -0.0758$ and $R_7 = -0.0102$. After the pole-residue model has been constructed, it can be used to predict the future value of the transient output as in Figure 6.3, where "*" stand for the output of the pole-residue model, the solid curve stands for the actual output and the vertical line indicates up to what step the derivatives have been used.

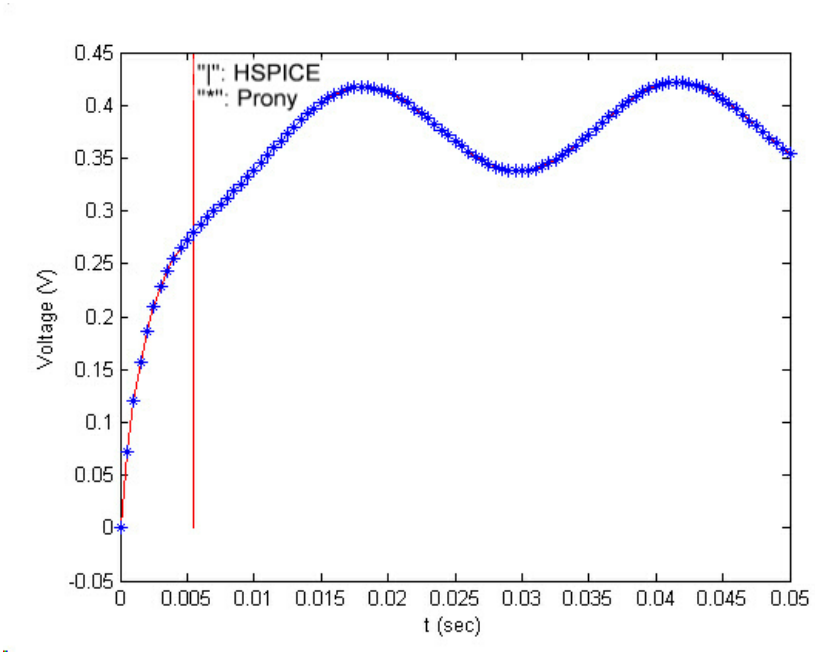

Figure 6.4: The output using modified method No.2

6.1.2 Modified Prony's Method and Digital Circuits

Due to the lack of very higher order derivatives, only modified Method No.2 is tested for digital circuits. Several circuits have been tested. CPU time has also been evaluated for both the modified Prony's approach and the regular constant step simulation. The results indicate that the CPU time of the modified Prony's approach for the following examples is around 1/5 of that of the regular constant step simulation.

6.1.2.1 NAND gate

The NAND gate in Figure 6.5 has been tested.

The first 21 samples, with derivatives ranging from 0^{th} to 2^{nd} order were used in that example. The sampling step is 8×10^{-13}sec. 15 pairs of poles and residues are generated. Figure 6.6 and 6.7 show the predicted transient output in"*" compared

to the actual transient output in solid curve. The vertical line indicates up to what step the derivatives have been generated.

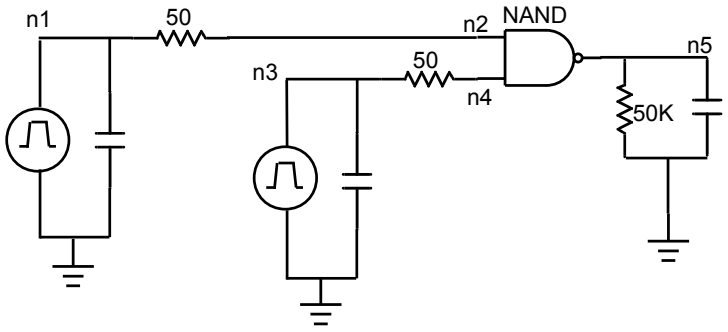

Figure 6.5: A Two-input NAND Gate

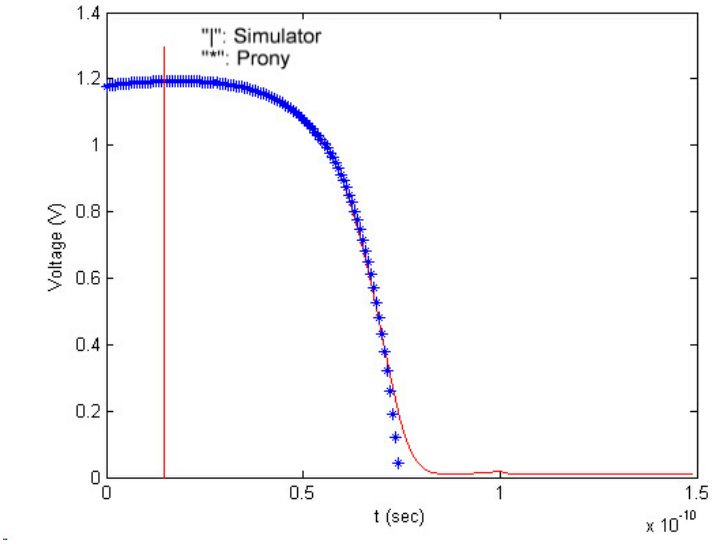

Figure 6.6: The Falling Edge of the Two-input NAND Gate in Figure 6.5

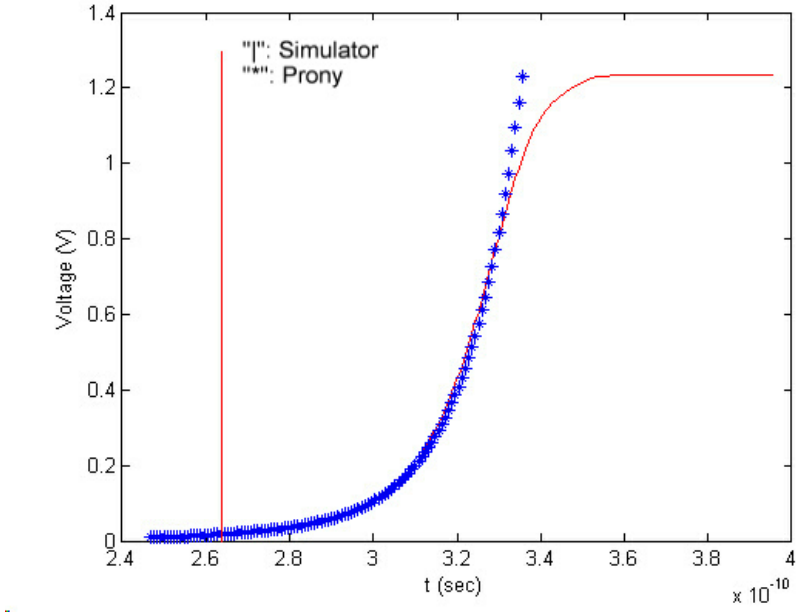

Figure 6.7: The Rising Edge of the Two-input NAND Gate in Figure 6.5

6.1.2.2 NOR gate

The following NOR gate in Figure 6.8 has next been considered for Method No.2.

The first 21 samples, with derivatives ranging from 0^{th} to 2^{nd} order were used in that example. The sampling step is 8×10^{-13}sec. 15 pairs of poles and residues are generated. Figure 6.9 and 6.10 show the predicted transient output in"*" compared to the actual transient output in solid curve. The vertical line indicates up to what step the derivatives have been used.

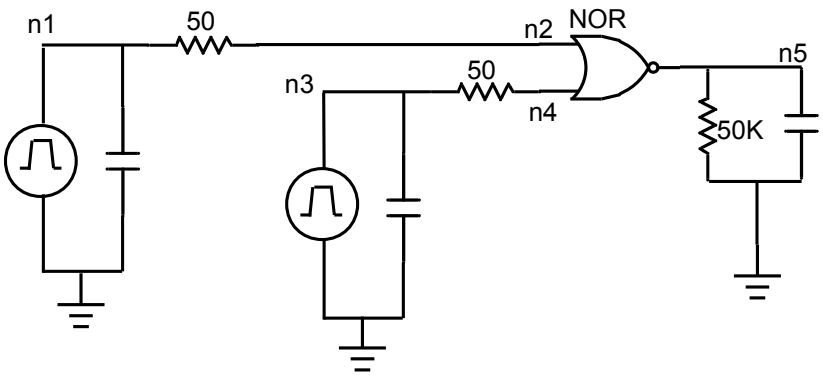

Figure 6.8: A Two-input NOR Gate

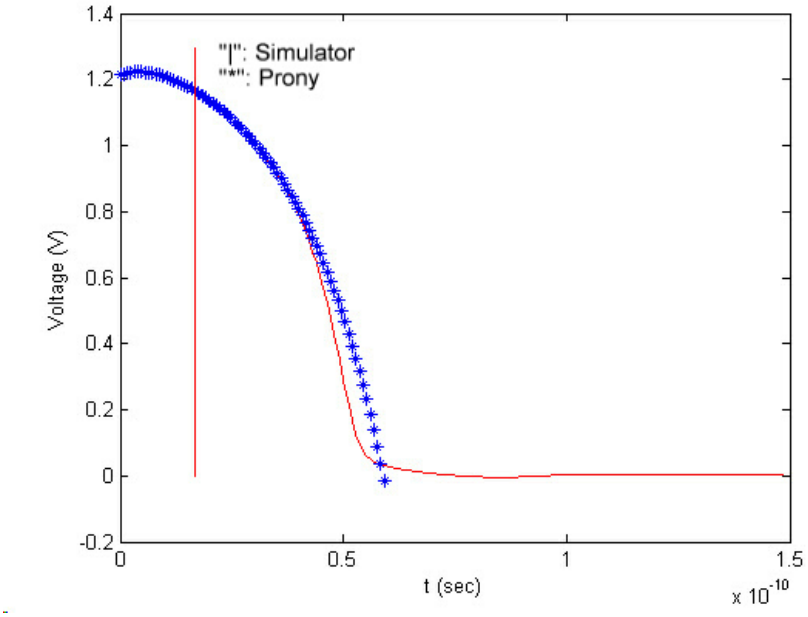

Figure 6.9: The Falling Edge of the Two-input NOR Gate in Figure 6.8

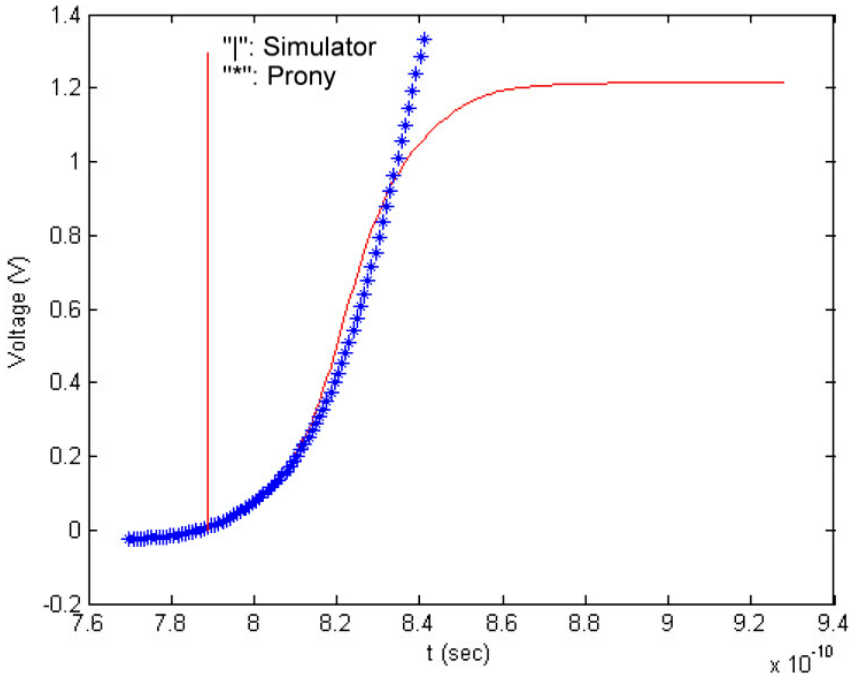

Figure 6.10: The Rising Edge of the Two-input NOR Gate in Figure 6.8

6.1.2.3 Inverter-NAND gate

The following Inverter-NAND gate in Figure 6.11 is considered.

The first 21 samples, with derivatives ranging from 0^{th} to 2^{nd} order were used in that example. The sampling step is 9×10^{-13}sec. 15 pairs of poles and residues are generated. Figure 6.12 and 6.13 show the predicted transient output in"*" compared to the actual transient output in solid curve. The vertical line indicates up to what step the derivatives have been used.

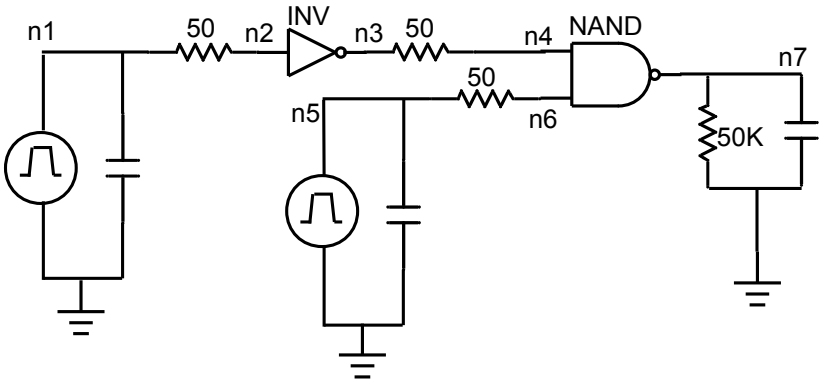

Figure 6.11: An Inverter-NAND Gate

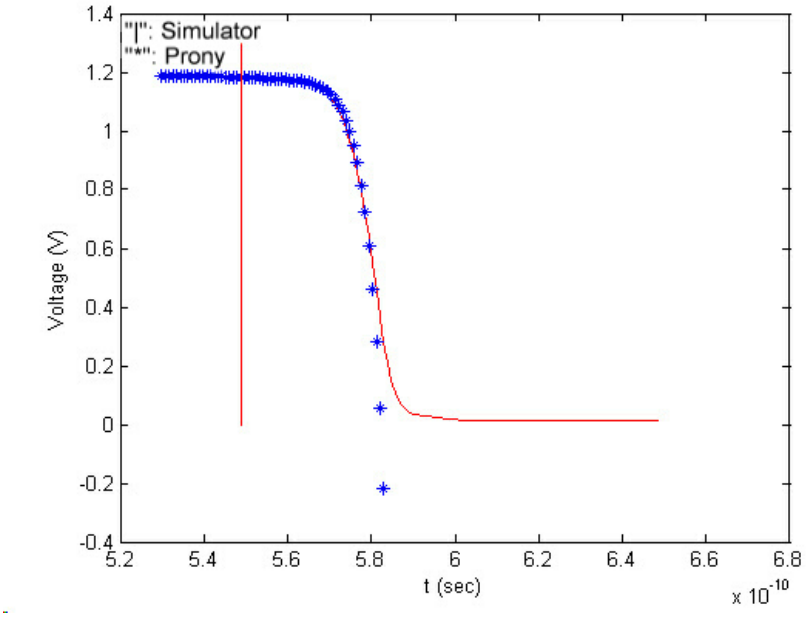

Figure 6.12: The Falling Edge of the Inverter-NAND Gate in Figure 6.11

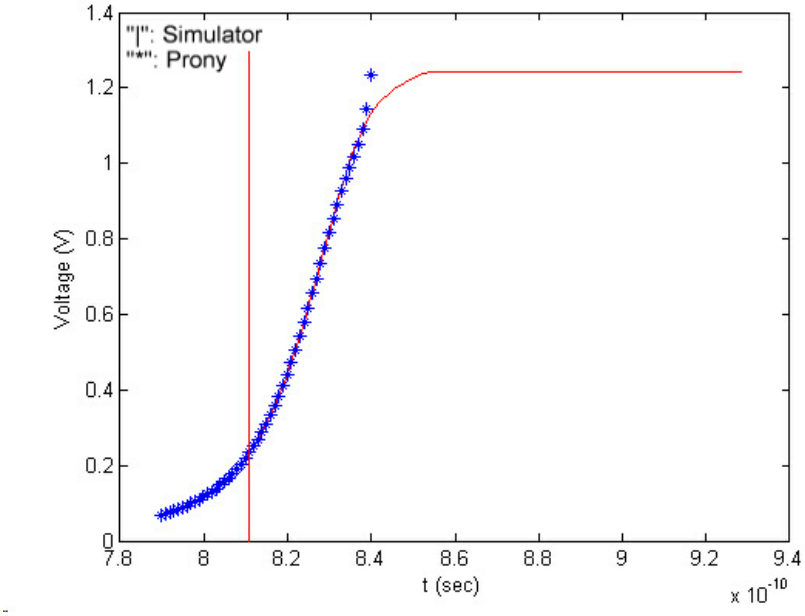

Figure 6.13: The Rising Edge of the Inverter-NAND Gate in Figure 6.11

6.1.2.4 Inverter-NOR gate

The following Inverter-NOR gate in Figure 6.14 is considered.

The first 21 samples, with derivatives ranging from 0^{th} to 2^{nd} order were used in that example. The sampling step is 10×10^{-13}sec. 15 pairs of poles and residues are generated. Figure 6.15 and 6.16 show the predicted transient output in"*" compared to the actual transient output in solid curve. The vertical line indicates up to what step the derivatives have been used.

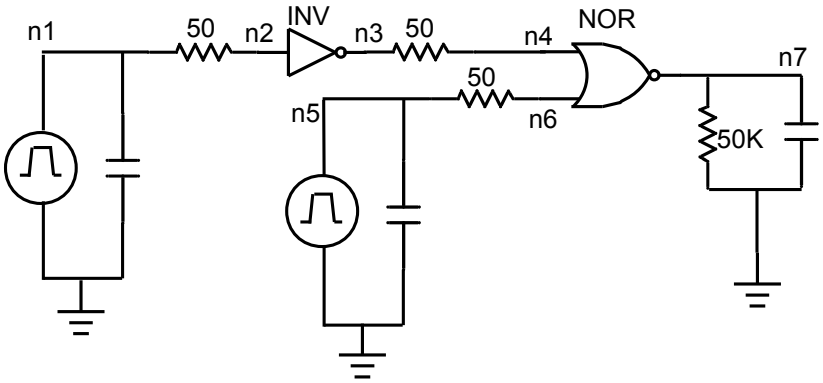

Figure 6.14: An Inverter-NOR Gate

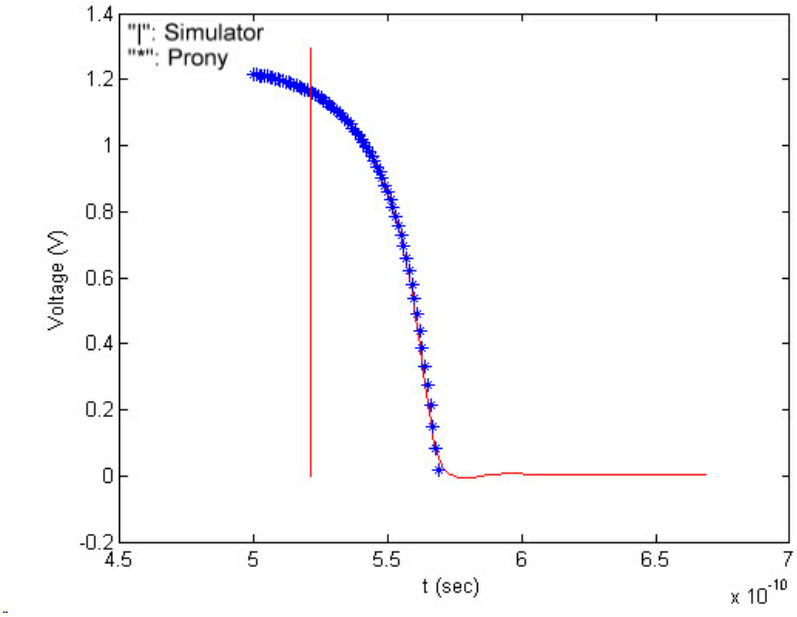

Figure 6.15: The Falling Edge of the Inverter-NOR Gate in Figure 6.14

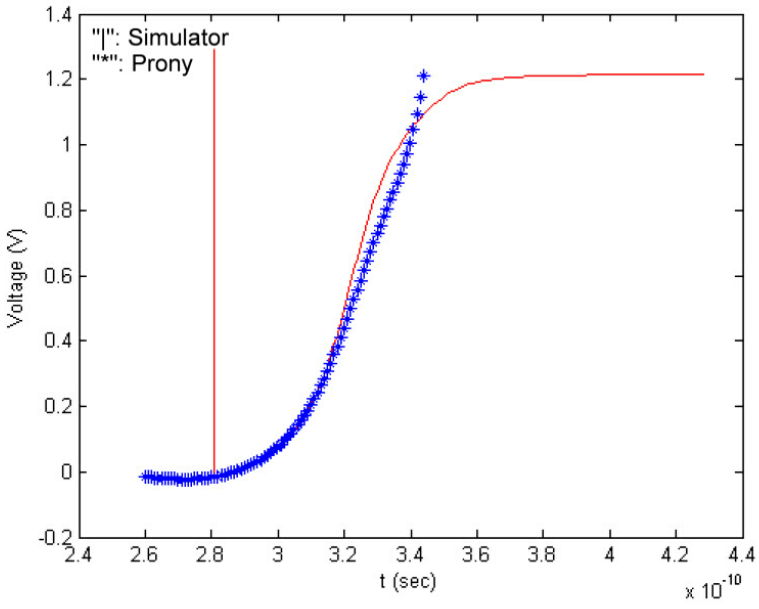

Figure 6.16: The Rising Edge of the Inverter-NOR Gate in Figure 6.14

6.2 STA Delay Look-up Table

This section investigates the application of the look-up based table approach to STA to the following test circuit, and for inputs that are represented by a one or two-parameter waveform.

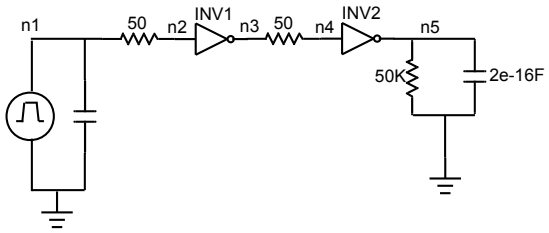

Figure 6.17: Two cascaded inverters

6.2.1 Single Parameter Input Waveform

Consider the following saturated ramp input waveform:

$$r(t)/V_{DD} = \begin{cases} 0, t \leq t_{50\%} - 0.5t_s \\ \frac{t - t_{50\%} + 0.5t_s}{t_s}, t_{50\%} - 0.5t_s \leq t \leq t_{50\%} + 0.5t_s \\ 1, t \geq t_{50\%} + 0.5t_s \end{cases}, \quad (6.1)$$

where $t_{50\%}$=0.1ns, t_s=0.2ns and V_{DD}=1.2V. The following Figure 6.18 gives its shape.

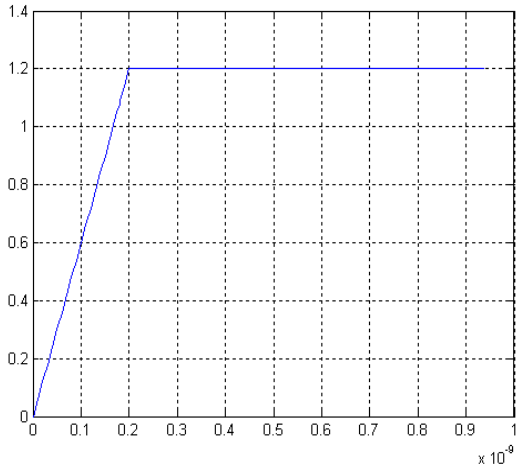

Figure 6.18: A saturated ramp signal

Using the method mentioned in Chapter 2, the effective load capacitance for the first inverter can be found as 0.2pF. Next, using t_s=0.2ns and load capacitance C_L=0.2pF, the output waveform $W1$ for the first inverter can be found in the fall waveform look-up table, as shown in Table 6.1.

Fall Table	C_L=0.2pF
t_s=0.2ns	W1

Table 6.1: A Fall Output Waveform Table

Figure 6.19 shows the waveform W1.

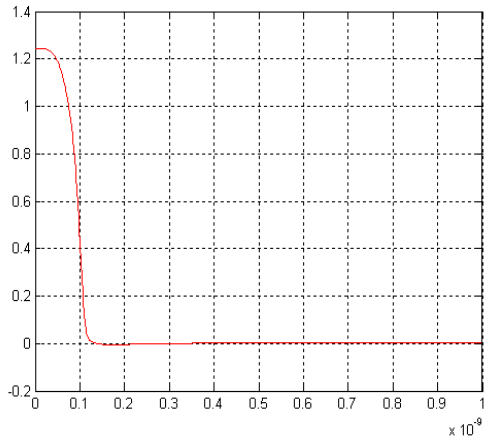

Figure 6.19: The output waveform of the first inverter: W1

Or in the form of Prony's approximation, it can be represented by the following poles and residues.

Poles($\times 10^{11}$)	$-0.143 \pm j2.808$	$-0.245 \pm j2.335$	$-0.373 \pm j1.830$	$-0.527 \pm j1.658$
Residues	$0.0029 \pm j0.0045$	$0.0113 \pm j0.0541$	$-0.252 \pm j0.773$	$0.24 \pm j - 2.150$

Table 6.2: the Prony's approximation of W1 (to be continued)

Poles($\times 10^{11}$)	$-0.3617 \pm j1.2059$	$-0.3623 \pm j0.7173$	$-0.3792 \pm j0.2424$	-0.87
Residues	$1.5632 \pm j0.2320$	$2.167 \pm j2.9913$	$-8.0818 \pm j9.5011$	9.94

Table 6.3: the Prony's approximation of W1 (continued)

The output delay 0.7265e-11(sec) and the slew time 5.599e-11(sec) can be obtained using Newtion iteration, and they will be passed on to the second inverter. The slew time of the first inverter will be used to describe the input signal of the second inverter. Similar to the first inverter, the input slew and the load capacitance will determine the output waveform of the second inverter.

Rise Table	C_L=2e-16F
t_s=5.599e-11sec	W2

Table 6.4: A Rise Output Waveform Table

Figure 6.20 shows the waveform W2 along with its input waveform.

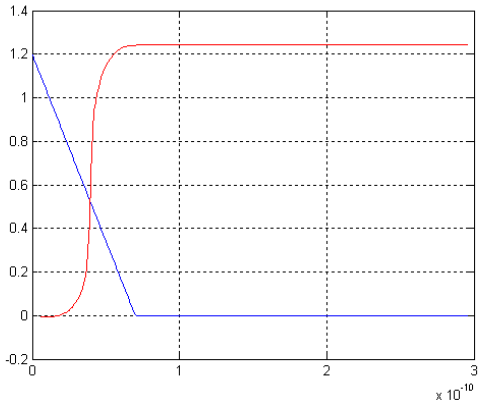

Figure 6.20: The input/output waveform of the second inverter: W2

Again, the output delay 1.2710e-11(sec) and the slew time 1.432e-11(sec) can be computed using Newton iteration from W2. Therefore, the total output delay is 0.7265e-11+1.2710e-11=1.9975e-11(sec) and the total slew is 1.432e-11(sec). The following figure shows W2 in "+" and simulation output in solid line.

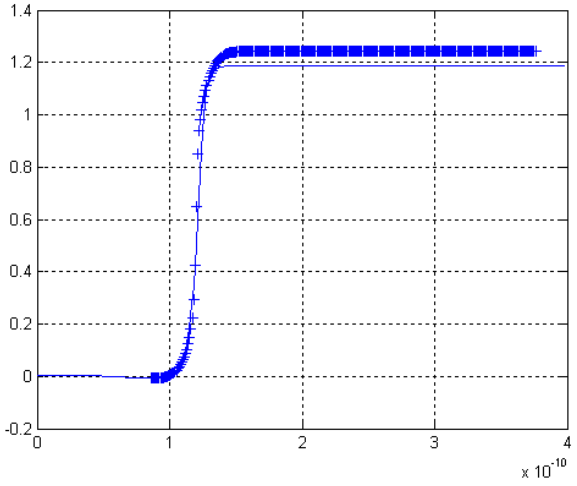

Figure 6.21: W2 and Output of Simulation together

For convenience, both results can be found in the following table.

	Simulation	Waveform Look-up Table Method
output delay($\times 10^{-11}$sec)	2.0956	1.9975
output slew($\times 10^{-11}$sec)	1.477	1.432

Table 6.5: Results from Both Methods

6.2.2 Two-Parameter Input Waveform

Consider the following Weibull function as the circuit input.

$$WB(t, \alpha, \beta)/Vdd = 1 - e^{-(\frac{t}{\beta})^{\alpha}} \quad , \tag{6.2}$$

where α=1.46, β=1.07e-10 and Vdd=1.2V. And t_s=9.3996e-11(sec) can be computed using equation 5.6. The following Figure 6.22 gives its shape.

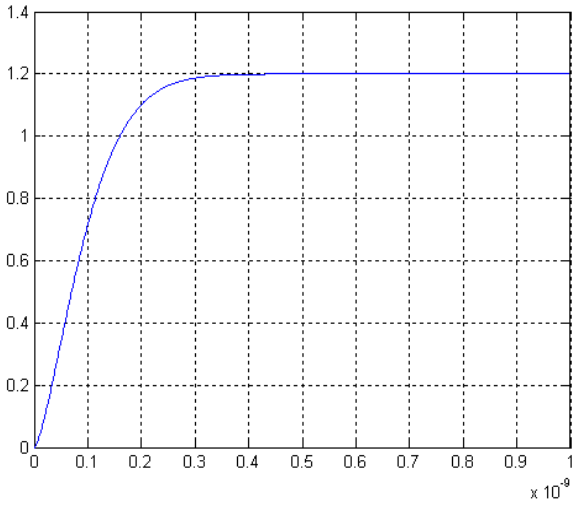

Figure 6.22: A Weibull's waveform

Using the method mentioned in Chapter 2, the effective load capacitance for the first inverter can be found as 0.2pF. Next, using α=1.46 , t_s=9.3996e-11(sec) and load capacitance C_L=0.2pF, the output waveform $W1$ for the first inverter can be found in the fall waveform look-up table, as shown in Table 6.2.2.

Fall Table	C_L=0.2pF
α=1.46, t_s=9.3996e-11sec	W1

Table 6.6: A Fall Output Waveform Table

Figure 6.23 shows the waveform $W1$.

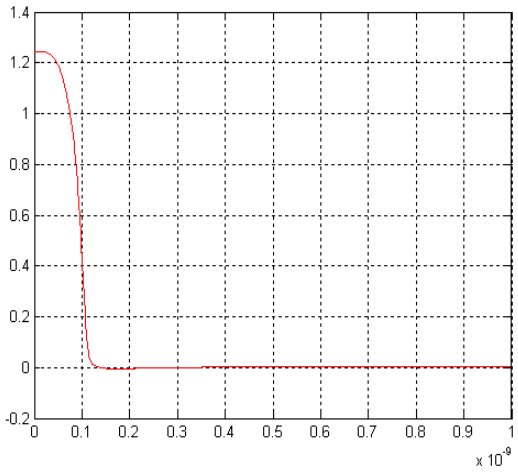

Figure 6.23: The output waveform of the first inverter: W1

Or in the form of Prony's approximation, it can be represented by the following poles and residues.

Poles($\times 10^{11}$)	$-0.251 \pm j0.0585$	-1.0293	$-1.1772 \pm j0.6158$	$-1.2856 \pm j1.1105$
Residues	$-1.2916 \pm j1.4303$	5700.6	$10888 \pm j - 1661.9$	$7297.7 \pm j - 14229$

Table 6.7: the Prony's approximation of W1 (to be continued)

Poles($\times 10^{11}$)	$-1.3857 \pm j2.5753$	$-1.3899 \pm j2.0786$	$-1.3549 \pm j1.5928$
Residues	$-1946.7 \pm j3553.6$	$-10202 \pm j1186.5$	$-8880.6 \pm j - 12652$

Table 6.8: the Prony's approximation of W1 (continued)

The output shape factor α=4.7793 can be found using Weibull fitting algorithm. The output t_s= 4.6582e-11 can also be found using equation 5.6. And they will be passed on to the second inverter. The output shape factor and the output slew time of the first inverter will be used to describe the input signal of the second inverter. Similar to the first inverter, the shape factor, the input slew and the load capacitance will determine the output waveform W2 of the second inverter.

Rise Table	C_L=2e-16F
α=4.7793, t_s=4.6582-11sec	W2

Table 6.9: A Rising Output Waveform Table

Figure 6.24 shows the waveform $W2$.

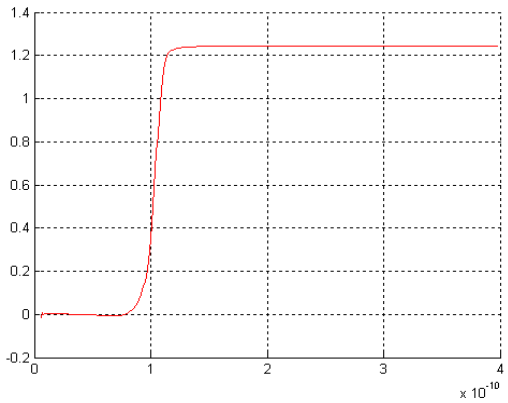

Figure 6.24: The output waveform of the second inverter: W2

Again, the output delay 2.0253e-11(sec) and the slew time 1.4638e-11(sec) can be computed using Weibull fitting algorithm from W2. The following figure shows W2 in "+" and simulation output in solid line.

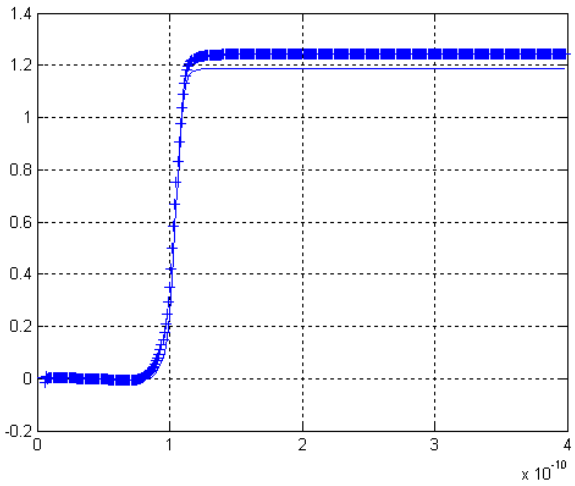

Figure 6.25: W2 and Output of simulation together

For convenience, both results can be found in the following table.

	Simulation	Waveform Look-up Table Method
output delay($\times 10^{-11}$sec)	2.1092	2.0253
output slew($\times 10^{-11}$sec)	1.3971	1.4638

Table 6.10: Results from Both Methods

Chapter 7

Conclusions

This thesis examined the potential of Prony's method to the application of timing analysis in digital circuits. Traditionally, Prony's method has been used in system identification to approximate the propagation modes in electromagnetic structures. This thesis provided a scheme to extend its applications to the problem of timing analysis in digital circuits. Two approaches have been proposed to achieve this end.

The first approach relies on modifying the existing Prony-based approaches so that it can be used to predict the future shape of the waveform. The proposed approach is based on using the high-order derivatives at fewer sampling points towards the beginning of the waveform to predict its long term behavior. The proposed approach was termed as a Dynamic Timing Analysis(DTA) due to its ability to analyze the gates under dynamic loads. It was shown that this approach is able, with varying degrees of success, to predict the behavior of a number of digital gates. And the CPU time of using the proposed approach is much less than that of using the regular constant step simulation.

Another approach based on the idea of Static Timing Analysis using Prony's method has been presented in the thesis. That approach works by characterizing individual logical gates using their response, under different input and load conditions, via a set of poles and residues. These characteristics are stored in a look-up table and retrieved for a specific input and loading conditions. Several examples have been presented to demonstrate the accuracy of the proposed STA approach.

7.1 Future Work

Efficiency in memory size is a practical requirement of look-up table, which requires least number of pole-residue pair.

When the modified Prony's method is applied in rise/fall delay analysis, it is actually an issue of linear system identification. In order to have a reasonable identification, some extra boundary conditions may be applied. For example, the slope of the digital transient output curve should tend to 0 when it approaches the high/low voltage level. Currently, this information has not been used in the prediction procedure.

Secondly, the modification No.1 is worth further study, because it does not require constant step size and it requires the least steps to reconstruct the pole-residue model.

Appendix A

Derive a High Order Derivatives Formula

We use a polynomial to approximate a curve and it's derivatives. In general, we can fit a polynomial of $m = 2k - l$ degree. Define the targeting polynomial as:

$$x_m(t) = \sum_{j=0}^{m} d_j \left(\frac{t_{n+k} - t}{h}\right)^j \tag{A.1}$$

and it has l^{th} order derivatives as:

$$x_m^{(l)}(t) = (-1)^l \frac{1}{h^l} \sum_{j=l}^{m} \frac{j!}{(j-l)!} d_j \left(\frac{t_{n+k} - t}{h}\right)^{j-l} \tag{A.2}$$

where $l = 0, 1, 2, ..., m$.

A.1 Predictor

The predictor's coefficients can be determined through the following system:

$d_0^P + d_1^P + d_2^P + \ldots + d_m^P = x_{n+k-1}$

$$d_0^P + 2d_1^P + 2^2 d_2^P + \ldots + 2^m d_m^P = x_{n+k-2}$$

$$d_0^P + 3d_1^P + 3^2 d_2^P + \ldots + 3^m d_m^P = x_{n+k-3}$$

$$\vdots$$

$$d_1^P + 2d_2^P + 3d_3^P + \ldots + md_m^P = -hx'_{n+k-1}$$

$$d_1^P + 2 \times 2d_2^P + 3 \times 2^2 d_3^P + \ldots + m2^{m-1} d_m^P = -hx_{n+k-2}$$

$$d_1^P + 2 \times 3d_2^P + 3 \times 3^3 d_3^P + \ldots + m3^{m-1} d_m^P = -hx_{n+k-3}$$

$$\vdots$$

$$l!d_l^P + (l+1)!d_{l+1}^P + \ldots + \tfrac{m!}{(m-l)!} d_m^P = (-1)^l h^l x_{n+k-1}^{(l)}$$

$$l!d_l^P + (l+1)! \times 2d_{l+1}^P + \ldots + \tfrac{m!}{(m-l)!} 2^{m-l} d_m^P = (-1)^l h^l x_{n+k-2}^{(l)}$$

$$l!d_l^P + (l+1)! \times 3d_{l+1}^P + \ldots + \tfrac{m!}{(m-l)!} 3^{m-l} d_m^P = (-1)^l h^l x_{n+k-3}^{(l)}$$

In the matrix form it is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & . & . & 1 \\ 1 & 2 & 2^2 & 2^3 & . & . & 2^m \\ 1 & 3 & 3^2 & 3^3 & . & . & 3^m \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 2 & 3 & . & . & m \\ 0 & 1 & 2 \times 2 & 3 \times 2^2 & . & . & m2^{m-1} \\ 0 & 1 & 2 \times 3 & 3 \times 3^2 & . & . & m3^{m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \ldots & \ldots & \ldots & \ldots & . & . & \ldots \\ 0 & 0 & \ldots & l! & (l+1)! & \ldots & \frac{m!}{(m-l)!} \\ 0 & 0 & \ldots & l! & (l+1)! \times 2 & \ldots & \frac{m!}{(m-l)!} 2^{m-l} \\ 0 & 0 & \ldots & l! & (l+1)! \times 3 & \ldots & \frac{m!}{(m-l)!} 3^{m-l} \end{bmatrix} \begin{bmatrix} d_0^P \\ d_1^P \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ d_m^P \end{bmatrix} = \begin{bmatrix} x_{n+k-1} \\ x_{n+k-2} \\ x_{n+k-3} \\ \vdots \\ -hx'_{n+k-1} \\ -hx'_{n+k-2} \\ -hx'_{n+k-3} \\ \vdots \\ \ldots \\ (-1)^l h^l x_{n+k-1}^{(l)} \\ (-1)^l h^l x_{n+k-2}^{(l)} \\ (-1)^l h^l x_{n+k-3}^{(l)} \end{bmatrix} \tag{A.3}$$

Let's denote this system A.3 as

$$\mathbf{V}^P \mathbf{d}^P = \mathbf{z}^P \tag{A.4}$$

The full set of the vector $\mathbf{d}^P$ is not needed for our purpose, what we need is the value $x_{n+k} = d_0^P$. Rather than solving the system A.3 we solve for the following system.

$$\left(\mathbf{V}^P\right)^T \begin{bmatrix} a_1^P \\ a_2^P \\ a_3^P \\ \vdots \\ b_1^P \\ b_2^P \\ b_3^P \\ \vdots \\ \cdots \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ \cdots \end{bmatrix} \tag{A.5}$$

Let's denote this system A.5 as

$$\left(\mathbf{V}^P\right)^T \phi^P = \mathbf{e}_0 \tag{A.6}$$

Since we have $x_{n+k} = d_0^P = \mathbf{e}_0^T \mathbf{d}^P = (\phi^P)^T \mathbf{V}^P \mathbf{d}^P = (\phi^P)^T \mathbf{z}^P$, using the entrices of ϕ^P and $\mathbf{z}^P$, we have the following predictor formula

$$x_{n+k} = \sum_{j=1}^{k_1} a_j^P x_{n+k-j} - h \sum_{j=1}^{k_2} b_j^P x'_{n+k-j} + h^2 \sum_{j=1}^{k_3} c_j^P x''_{n+k-j} + \ldots \tag{A.7}$$

A.2 Corrector

The corrector formula is required to represent the derivative $x_{n+k}^{(l)}$ using previous value x_i, previous derivatives $x_i^{(q)}$, where $q = 1, 2, \ldots, l, i < n + k$, and from the current

values of $x_{n+k}^{(l-1)}, x_{n+k}^{(l-2)}, \ldots, x_{n+k}$. We need the following system.

$$d_0^C = x_{n+k}$$

$$d_0^C + d_1^C + d_2^C + \ldots + d_m^C = x_{n+k-1}$$

$$d_0^C + 2d_1^C + 2^2 d_2^C + \ldots + 2^m d_m^C = x_{n+k-2}$$

$$\vdots$$

$$d_1^C = -hx'_{n+k}$$

$$d_1^C + 2d_2^C + 3d_3^C + \ldots + md_m^C = -hx'_{n+k-1}$$

$$d_1^C + 2 \times 2d_2^C + 3 \times 2^2 d_2^C + \ldots + m2^{m-1} d_m^C = -hx'_{n+k-2}$$

$$\vdots$$

$$l!d_l^C + (l+1)!d_{l+1}^C + \ldots + \tfrac{m!}{(m-l)!} d_m^C = (-1)^l h^l x_{n+k}^{(l)}$$

$$l!d_l^C + (l+1) \times 2!d_{l+1}^C + \ldots + \tfrac{m!}{(m-l)!} 2^{m-l} d_m^C = (-1)^l h^l x_{n+k-1}^{(l)}$$

$$\vdots$$

In matrix form, it becomes

$$\begin{bmatrix} 1 & 0 & 0 & 0 & . & . & 0 \\ 1 & 1 & 1 & 1 & . & . & 1 \\ 1 & 2 & 2^2 & 2^3 & . & . & 2^m \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & 0 & . & . & 0 \\ 0 & 1 & 2 & 3 & . & . & m \\ 0 & 1 & 2\times 2 & 3\times 2^2 & . & . & m2^{m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & \dots & \dots & \dots & . & . & \dots \\ 0 & 0 & \dots & 0 & l! & \dots & \frac{m!}{(m-l)!} \\ 0 & 0 & \dots & 0 & l!\times 2 & \dots & \frac{m!}{(m-l)!}2^{m-l} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} d_0^C \\ d_1^C \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ d_m^C \end{bmatrix} = \begin{bmatrix} x_{n+k} \\ x_{n+k-1} \\ x_{n+k-2} \\ \vdots \\ -hx'_{n+k} \\ -hx'_{n+k-1} \\ -hx'_{n+k-2} \\ \vdots \\ \dots \\ (-1)^l h^l x^{(l)}_{n+k} \\ (-1)^l h^l x^{(l)}_{n+k-1} \\ \vdots \end{bmatrix} \tag{A.8}$$

Let's denote this system A.8 as

$$\mathbf{V}^C \mathbf{d}^C = \mathbf{z}^C \tag{A.9}$$

The full set of $\mathbf{d}^C$ is not needed for out purpose, what we need is the value $(-1)^l h^l x^{(l)} = d_l^C$. Rather than solving the system A.8 we solve for the following

system.

$$\left(\mathbf{V}^C\right)^T \begin{bmatrix} a_1^C \\ a_2^C \\ a_3^C \\ \vdots \\ b_1^C \\ b_2^C \\ b_3^C \\ \vdots \\ \cdots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ l! \\ 0 \\ 0 \\ 0 \\ \vdots \\ \cdots \end{bmatrix} \tag{A.10}$$

Let's denote this system A.10 as

$$\left(\mathbf{V}^C\right)^T \phi^C = \mathbf{e}_l \tag{A.11}$$

Since we have $(-1)^l l! h^l x^{(l)} = d_l^C = \mathbf{e}_l^T \mathbf{d}^C = (\phi^C)^T \mathbf{V}^C \mathbf{d}^C = (\phi^C)^T \mathbf{z}^C$, using the entrices of ϕ^C and $\mathbf{z}^C$, we have the following corrector formula

$$x_{n+k}^{(l)} = (-1)^l \frac{1}{h^l} \left(\sum_{j=0}^{k_1} a_j^C x_{n+k-j} - h \sum_{j=0}^{k_2} b_j^C x'_{n+k-j} + \ldots + (-1)^l h^l \sum_{j=0}^{k_m} c_j^C x_{n+k-j}^{(l)} \right) \tag{A.12}$$

Bibliography

[1] K. Agarwal, D. Sylvester, and D. Blaauw. An effective capacitance based driver output model for on-chip rlc interconnects. *DAC*, June:2–6, 2003.

[2] M. J. Amatangelo. Vlsi-ii lecture notes. *The University of Texas at Austin*, 2008.

[3] C. S. Amin, F. Dartu, and Y. I. Ismail. Weibull-based analytical waveform model. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24:1156–1168, 2005.

[4] H. Bhatnagar. *Advanced ASIC Chip Synthesis: Using synopsys Design Compiler, Physical Compiler and Prime Time*. Kluwer Academic Publishers, second edition, 2002.

[5] M. L. V. Blaricum and R. Mitrra. A technique for extracting the poles and residues of a system directly from its transient response. *IEEE Trans. on Antennas and Propagation*, Ap-23(6), 1975.

[6] Y. Bresler and A. Macovski. Exact maximum likelihood parameter estimation of superimposed exponential signals in noise. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 34:1081–1089, 1986.

[7] G. Dahlquist. A special stability problem for linear multistep methods. *BIT*, 3:27–43, 1963.

[8] F. Dartu, N. Menezes, J. Qian, and L. T. Pillage. A gate-delay model for high-speed cmos circuits. *31st ACM/IEEE Design Automation Conf.*, 1994.

[9] E. Gad, R. Khazaka, M. Nakhla, and R. Griffith. A circuit reduction technique for finding the steady-state solution of nonlinear circuits. *IEEE Trans. on Microwave Theory Tech.*, 48(10):2389–2396, 2000.

[10] E. Gad, M. Nakhla, R. Achar, and Y. Zhou. A-stable and l-stable high-order integration methods for solving stiff differential equations. *IEEE Trans. on CAD*, 2009.

[11] F. B. Hildebrand. *Introduction to Numerical Analysis*. McGraw-Hill Book Company, INC., first edition, 1956.

[12] N. J. Holt and J. R. Antill. Determining the number of terms in a prony algorithm exponential fit. *Mathematical Biosciences*, 36:319–332, 1977.

[13] Z. Huang, A. Kurokawa, J. Pan, and Y. Inoue. Modeling the effective capacitance of interconnect loads for predicting comos gate slew. *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, 2005.

[14] T. Kailath. *Linear Systems*. Prentice Hall, Englewood Cliffs, 1980.

[15] F. Khalvati and M. Nummer. Digital systems engineering lecture slides. *University of Waterloo*, Winter, 2007.

[16] M. Khan, M. S. Mackisack, M. R. Osborne, and G. K. Smyth. the consistency of prony method and related algorithms. *Journal of Computer. Graph. Statistics*, 1:329–349, 1992.

[17] P. Loskot and N. C. Beaulieu. Further results on prony approximation for evaluation of the average probability of error. *ICC*, 2008.

[18] M. R. Osborne and G. K. Smyth. A modified prony algorithm for exponential function fitting. 1995.

[19] H. Price. An improved prony algorithm for exponential analysis. *IEEE Int. Symp. Electromagn. Compat.*, 1:310–313, 1979.

[20] J. Qian, S. Pullela, and L. Pillage. Modeling the effective capacitance for the rc interconnect of cmos gates. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 13:1526–1535, 1994.

[21] K. Radecka and Z. Zilic. *Verification by Error Modeling-Using Testing Techniques in Hardware Verification*. Kluwer Academic Publishers, 2003.

[22] P. Schaumont. Digital design ii lecture notes. *Virginia Tech*, Spring, 2008.

[23] L. Scheffer, L. Lavagno, and G. Martin. *EDA for IC Implementation, Circuit Design, and Process Technology*. Taylor and Francis Group, LLC, first edition, 2006.

[24] A. Shebaita, D. Das, D. Petranovic, and Y. Ismail. A novel moment based framework for accurate and efficient static timing analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2010.

[25] A. Shebaita, D. Petranovic, and Y. Ismail. Including inductance in static timing analysis. *Int. Conf. on Computer Aided Design*, 11:686–691, 2007.

[26] J. Vlach and K. Singhal. *Computer Methods for Circuit Analysis and Design*. Kluwer Academic Publishers, second edition, 2003.