Macromodeling Nonlinear Circuits Using Proper Orthogonal Decomposition and Artificial Neural Networks

Marwan Kanaan

Department of Electrical and Computer Engineering
McGill University, Montreal, Canada

June 2021

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

“ The Buddha, the Godhead, resides quite as comfortably in the circuits of a digital computer or the gears of a cycle transmission as he does at the top of the mountain, or in the petals of a flower. ”

— Robert M. Pirsig, *Zen and the Art of Motorcycle Maintenance*

Contents

List of Figures

List of Acronyms

ANN	Artificial neural network
AC	Alternating current
ASIC	Application-specific integrated circuit
CAD	Computer-aided design
CMOS	Complementary metal-oxide-semiconductor
CPU	Central processing unit
DC	Direct current
DFT	Direct Fourier transform
EDA	Electronic design automation
FFT	Fast Fourier transform
HB	Harmonic balance
HDL	Hardware description language
IC	Integrated circuit
IDFT	Inverse direct Fourier transform
IP	Intellectual property
KCL	Kirchhoff's current law
KVL	Kirchhoff's voltage law
LTI	Linear time invariant

LTV	Linear time variant
MLP	Multilayer perceptron
MNA	Modified nodal analysis
MOR	Model order reduction
POD	Proper orthogonal decomposition
RBF	Radial basis function
RC	Resistor-capacitor circuit
RF	Radio frequency
RLC	Resistor-inductor-capacitor circuit
SPICE	Simulation program with integrated circuit emphasis
SoC	System on chip
SVD	Singular value decomposition
VLSI	Very large scale integrated circuit

Abstract

Reduced-order macromodels are a useful tool that designers can use to speed up circuit simulations. When carefully constructed, these macromodels can guarantee a high degree of accuracy and replace the original subcircuits under certain conditions, which the designer can also choose. Generating reduced-order macromodels of linear subcircuits has been thoroughly studied in the literature and several algorithms have been proposed which can successfully achieve that. On the other hand, generating nonlinear reduced-order macromodels proved to be a much more difficult task.

This thesis presents a general and systematic macromodeling algorithm that can be used to reduce the size of a wide range of nonlinear electronic circuits to speed up simulation. The method is composed of two separate algorithms that are used sequentially. In the first step, the size of equations of the electronic circuit is reduced using Proper Orthogonal Decomposition. The system is projected onto a predefined reduced subspace resulting in massive reduction of circuit size. This first step can only be applied to the linear part of the circuit, and because of the nonlinear nature of electronic circuits, the speedup gained from this reduction remains limited. To alleviate that, the second step targets the nonlinear part. Feedforward neural networks, known for their effectiveness as a curve fitting tool, are used to replace the functions describing the nonlinear part of the circuit.

By applying these two steps one after the other, the algorithm generates nonlinear

reduced-order macromodels capable of replacing complete electronic circuits. The macromodels can be directly added to a larger system and result, on average, in four to five times speedup in simulation time. These macromodels are valid over a specific range of conditions, such as input power, frequency, or loading conditions, which is chosen by the designer at construction.

Résumé

Les macromodèles d'ordre réduit sont un outil utile que les concepteurs peuvent utiliser pour accélérer les simulations de circuits. Une fois élaborés avec soin, ces macromodèles peuvent garantir un haut degré de précision et remplacer les sous-circuits originaux sous certaines conditions, que le concepteur peut également choisir. La génération de macromodèles d'ordre réduit de sous-circuits linéaires a été pleinement étudiée dans la littérature, et plusieurs algorithmes ont été proposés qui peuvent y parvenir avec succès. Cependant, la génération de macromodèles d'ordre réduit non-linéaires s'est avérée être une tâche beaucoup plus difficile.

Cette thèse présente un algorithme de macromodélisation général et systématique qui peut être utilisé pour réduire la taille d'une large gamme de circuits électroniques non-linéaires et en accélérer la simulation. La méthode est composée de deux algorithmes distincts qui sont exécutés séquentiellement. Dans la première étape, la taille des équations du circuit électronique est réduite à l'aide d'une décomposition orthogonale aux valeurs propres. Le système est projeté sur un sous-espace réduit prédéfini, ce qui entraîne une réduction massive de la taille du circuit. Cette première étape ne peut s'appliquer qu'à la partie linéaire du circuit, et du fait du caractère non-linéaire des circuits électroniques, l'accélération obtenue grâce à cette réduction reste limitée. Pour remédier à cela, la deuxième étape cible la partie non-linéaire. Les perceptrons multicouches, connus pour leur efficacité en tant qu'outil de régression, sont utilisés pour approximer la partie non linéaire du circuit.

En appliquant ces deux étapes l'une après l'autre, l'algorithme génère des macromodèles non-linéaires d'ordre réduit capables de remplacer des circuits électroniques complets. Les macromodèles peuvent également être directement ajoutés à un système plus grand, et peuvent permettre de réduire le temps de simulation par un facteur quatre ou cinq. Ces macromodèles sont valides sur une plage spécifique de conditions, telles que la puissance d'entrée, la fréquence ou les conditions de charge, qui sont choisies par le concepteur lors de la construction.

Acknowledgments

This work would not have been possible without the help and assistance of many people. First and foremost, I would like to thank my supervisor Roni Khazaka, who played the central role in guiding and advising me. His commitment, not only to my education but also to my future career, has had a very positive impact on my life. I would also like to thank the members of my supervisory committee and my oral defence committee for their guidance, questions, and input.

I am very thankful to my labmates and colleagues, with whom I have had many fruitful discussions over the years. The knowledge and expertise they shared with me during our many discussions have shaped my views and made me a better researcher. Lastly, I want to thank my friends and family in Montreal and abroad for their constant encouragement and support, without which this work would not have been possible.

Chapter 1

Introduction

1.1 Motivation

In the modern digital age, it seems hard to imagine that analog still has a place in our lives. After all, everything we use and interact with has become increasingly digital. The truth, however, is that analog remains and will continue to be an indispensable part of electronic design as long as the real world remains analog [1, 2]. While digital keeps on overtaking more and more functionalities in modern day systems, analog circuits are still essential when it comes to interfacing with the outside world. In today's System-on-Chips (SoCs), where whole systems with both analog and digital components are integrated on one die to achieve a lower cost [3], analog circuits remain key elements of the input, output, and the mixed-signal parts of the system [1]. For this reason, it is estimated that 90% of today's SoCs contain analog components [4].

In addition to the increased integration in modern chip design, there has also been an explosion in complexity. Today's circuits perform numerous functions, operate on lower power, occupy smaller chip area, and contain a greater number of components most of which

have also become more complex [2, 5–7]. Not only that but as it turns out, new shrinking technologies are bringing to light never before seen problems, and analog engineers are now dealing with very challenging design constraints [5, 7, 8]. Combined with the ever-present need to reduce time to market, today's chip manufacturers face a multitude of problems while trying to meet market demands.

In the midst of all this increased complexity, shrinking time to market, and new technologies, computer-aided design (CAD) tools emerge as the key to managing all those problems [1]. Unfortunately, unlike their digital counterpart, analog CAD tools do not live up to the challenge and remain very limited [8, 9]. There exists no complete set of tools which can guide an analog circuit designer through a full design cycle. Instead, the most used analog CAD tool today is a powerful, yet limited, industry-standard validation software known as simulation program with integrated circuit emphasis (SPICE). It was developed in the mid seventies of the last century at the University of California, Berkeley [10–15].

There are several reasons why there is a lack of mature CAD tools for analog design [1]. First and foremost, analog design is less systematic, certainly when compared with digital design. While the configuration of a basic logic gate, for example, is well established and can be pulled up from an existing library whenever needed, the configuration of an amplifier, as another example, is by no means standard. Second, analog design lacks higher abstraction. In digital design, once the basic logic gate is built, this particular circuit block can be easily packaged and has a very well defined set of input and outputs with clear relationships between them. The logic gate can now be combined with other logic gates to create a more complex system. That system itself could also be packaged and will have its own well defined set of input-output relationship. Unfortunately, there is no such thing in analog design. Analog circuit complexity and time-continuity prohibits abstraction a great deal and necessitates using the full circuit block to clearly define the input-output relationship. Lastly, analog

circuits are simply more complex. It should come as no surprise that developing CAD tools for circuits with nonlinear elements is inherently more difficult than developing tools for circuits that can represented with simple Boolean algebra.

This lack of mature CAD tools combined with the complexity of analog circuit design brings about an alarming fact: while analog circuits on average occupy a small chip area (about 20% [4]), their design is often the bottleneck of the design of the whole system and they are responsible for a disproportionate number of design faults and manufacturing reruns [1,9]. Specifically, transistor-level simulation of the whole system, commonly referred to as *system verification*, has become almost infeasible due to the increased CPU cost [6, 7, 16]. System verification is a very common step in the design cycle and is typically done at the very end. This is when the designers combine all the components and verify they all work together in the intended manner. Even with the fastest simulators used today, such as fast-SPICE [17], this task remains daunting.

To solve these problems, designers have come up with numerous solutions, one of which is the topic of this thesis: macromodeling. The process of macromodeling refers to creating reduced-order models of complex systems. A complex large circuit block is replaced by simpler smaller block which captures, to a certain degree of accuracy, the input-output relationship of the original one [18–20]. This order-reduction process can be done in different ways and will be explored in detail in Chapter 3.

Macromodeling has several advantages, the most obvious of which is simulation speedup. In reference to the problem of system verification discussed above, instead of simulating the whole system at transistor level and spending days or even weeks to achieve the final result, designers can now replace some of the large circuit blocks with their reduced-order models and perform the simulation. By doing so, they can still check the validity of the system in a shorter time. Of course, in this case the price paid is accuracy and one would hope that with

a carefully generated macromodel, the most crucial circuit characteristics are preserved and little loss of information occurs. Another advantage of using macromodels is fulfilling the much sought after need for abstraction in analog design. While it certainly is not on par with digital abstraction, macromodels can now serve as some form of higher level abstraction in analog circuit design. Lastly, macromodels are very useful in protecting intellectual property (IP) [1]. If a chip manufacturer decides not to share the inner workings of their design, a macromodeling technique could be used to conceal all the details. This is in contrast with providing a SPICE netlist which could be easily reverse-engineered.

Two well-known macromodeling techniques studied extensively in the literature are proper orthogonal decomposition (POD) and artificial neural networks (ANNs). Proper orthogonal decomposition belongs to a class of macromodeling techniques known as model order reduction (MOR). It has been mainly applied to linear circuits and in a limited way to nonlinear circuits. ANNs, on the other hand, belong to a different class of macromodeling techniques referred to as black box macromodeling. They have been used in numerous circuit simulation applications including linear and nonlinear circuit macromodeling.

In this thesis, we present a new macromodeling technique which aims to systematically generate reduced-order models of nonlinear circuits using POD and ANNs. Our novel approach combines the projection ability of POD with the dynamic modeling ability of neural networks. The result is a macromodeling method that can be used on different circuit topologies in both the time and frequency domains to generate accurate reduced-order models efficiently.

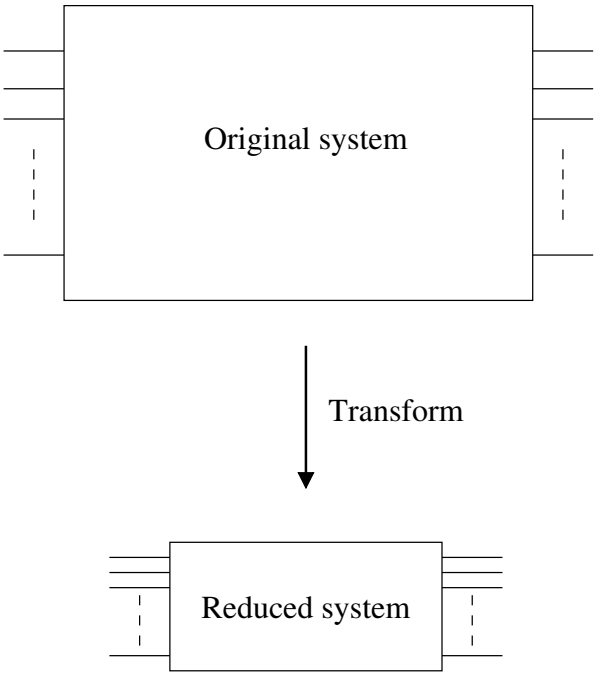

Fig. 1.1 An illustration of how macromodeling works

1.2 Problem Statement

Any circuit could be represented by a set of equations similar to the one shown in Equation 1.1 below:

$$Gx + Cx + f(x) = b \tag{1.1}$$

where the matrices and vectors G, C, f, and b contain all the known parameters and fully describe all the circuit elements. The vector x is the vector of unknown parameters which we hope to find by solving the equation. Later in Chapter 2, the reader will be presented with a systematic way to generate these equations, but for now this information is sufficient to illustrate the problem statement.

The number of equations in Equation 1.1 or its size, denoted by n, depends directly on the number of elements in this circuit. If n is large, for example in the case of a full SoC,

then solving Equation 1.1 becomes computationally very expensive.

Macromodeling aims to transform Equation 1.1 into Equation 1.2 whose size is q, where $q \ll n$. Now solving the new equation should be much easier given that it is much smaller.

$$\hat{G}\hat{x} + \hat{C}\hat{x} + \hat{f}(\hat{x}) = \hat{b} \tag{1.2}$$

This new smaller system could potentially be used instead of the original large system to speed up simulation whenever needed. The challenge, however, is finding the *reduced* matrices and vectors $\hat{G}$, $\hat{C}$, $\hat{f}$, and $\hat{b}$.

1.3 Contributions

The main contribution of this work is developing a new technique that combines two existing methods to generate reduced-order macromodels of nonlinear circuits in both the time and frequency domains. Specifically this thesis contains three distinct contributions:

1. Macromodeling of radio frequency circuits in the frequency domain [21]: RF circuits are typically simulated using the Harmonic Balance (HB) technique in order to study their nonlinear behaviour. Even for small circuits that contain only a handful of nodes, using the HB technique results in matrices that are hundreds of times larger than the original circuit equation, thus increasing simulation times. To address this, we extended a macromodeling technique, originally used for time-domain simulation only, to the frequency domain. First, we transformed the circuit equations into macromodel form that is suitable for reduction, and next we applied model-order reduction to reduce circuit size. This contribution is described in detail in Subsection 4.1.1 and Subsection 4.2.1 respectively.

2. Faster macromodels using neural networks: Proper orthogonal decomposition, while able to reduce the size of the circuit equations, lacks the ability to reduce the nonlinear part of the equations. To address this, we use feedforward neural networks to replace the nonlinear vector. This second contribution is an improvement over the first contribution and allows for much faster simulation. This contribution is discussed in detail in Subsection 4.2.2.

3. Macromodeling of nonlinear circuits in the time domain [22]: Circuit equations in the time domain were previously reduced using proper orthogonal decomposition. We extended this work by relying on POD to extract out the dynamic parts of the system, the MNA matrices, and used artificial neural networks to model the static parts, the nonlinear vector. To decouple the static part of the system, the nonlinear functions, from the dynamic parts, the MNA matrices, we combined proper orthogonal decomposition with neural networks. POD was used to reduce the matrices, while ANNs were used to model the nonlinear vector. Details of this contribution can be found in Chapter 5.

1.4 Organization

This thesis is organized as follows. Chapter 2 presents an introduction to the topic of circuit simulation and neural networks. The basic formulation of circuit equations and the types of analysis performed by industry-standard simulators are presented. A brief summary of artificial neural networks, their types, architectures, activation functions, and training algorithms follows. Chapter 3 contains a short history of macromodeling techniques in the literature. All the main types of macromodeling are presented, then special attention is given to the two classes of macromodeling relevant to this thesis: model order reduction and

artificial neural networks. Chapter 4 presents the first and second contribution of this work, macromodeling radio frequency circuits, along with numerical results. Chapter 5 presents the third contribution, macromodeling nonlinear circuits in the time domain. Lastly, Chapter 6 concludes the thesis and discusses opportunities for future work. A full bibliography is listed at the end.

Chapter 2

Background on Circuit Simulation and Artificial Neural Networks

This chapter presents the reader with a background on the fundamental aspects of circuit simulation and artificial neural networks. It is by no means complete or comprehensive, but merely an introduction to the most relevant aspects of both topics to this thesis.

2.1 Circuit Simulation

This section presents a brief introduction to the fundamentals of circuit simulation. Commercial and industry-standard circuit simulators such as SPICE use a similar set of equations to describe electric circuits [10–15]. We begin with an introduction of how these equations are derived and can be systematically formed. The later sections describe the most common types of analysis performed by modern simulators.

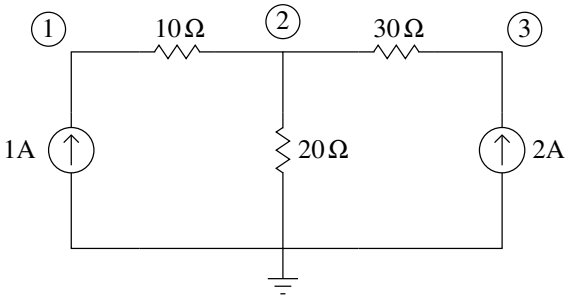

Fig. 2.1 Circuit example

2.1.1 Circuit Netlist

Consider the circuit example shown in Figure 2.1. The SPICE netlist of this circuit would look like:

```
R1 1 2 10
R2 2 3 30
R3 2 0 20
I1 0 1 DC 1
I2 0 3 DC 2
```

To analyze this netlist, SPICE arranges the information presented above as a set of equations. Depending on the type of analysis, SPICE would then solve these equations and present the user with the desired solution. In the few pages that follow, a brief description of this process is presented.

Consider the circuit of Figure 2.1 drawn again in Figure 2.2 but with the node numbers replaced by the voltage variables and the circuit components with names for clarity.

Using Kirchhoff's Voltage Law (KVL) and Kirchhoff's Current Law (KCL) one can derive a set of equations that describes the relationship between the voltages and currents for the circuit shown in Figure 2.2. Those equations can be written as:

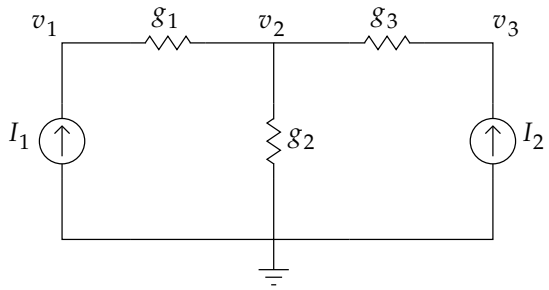

Fig. 2.2 Circuit example

$$\begin{aligned} g_1(v_1 - v_2) &= I_1 \\ g_1(v_2 - v_1) + g_3(v_2 - v_3) + g_2 v_2 &= 0 \\ g_3(v_3 - v_2) &= I_2 \end{aligned} \tag{2.1}$$

The above equations can be rearranged and written in matrix form as follows:

$$\begin{bmatrix} g_1 & -g_1 & 0 \\ -g_1 & g_1 + g_2 + g_3 & -g_3 \\ 0 & -g_3 & g_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} I_1 \\ 0 \\ I_2 \end{bmatrix} \tag{2.2}$$

The set of equations in Equation (2.2) completely describes the circuit and solving them will give the value of the unknown voltages of the circuit.

Modified Nodal Analysis

By carefully inspecting the matrices in Equation (2.2), one notices that the contribution of each resistor is as follows:

$$\begin{bmatrix} +g & -g \\ -g & +g \end{bmatrix} \tag{2.3}$$

where g is the admittance of the resistor. A similar *stamp* could be derived for the current sources in the circuit in Figure 2.2. Other circuit elements such as capacitors, inductors, voltage sources, and diodes have their own stamps which could be added to the overall set of equations of the circuit. Those equations are commonly known as the Modified Nodal Analysis (MNA) formulation [23, 24].

The time domain MNA formulation for any circuit which contains several linear and nonlinear elements can be written as follows:

$$\boldsymbol{G}\boldsymbol{x}(t) + \boldsymbol{C}\dot{\boldsymbol{x}}(t) + \boldsymbol{f}(\boldsymbol{x}(t)) = \boldsymbol{b}(t) \tag{2.4}$$

where

- $\boldsymbol{G} \in \mathbb{R}^{n \times n}$ contains the contributions of the memoryless elements such as resistors,
- $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ contains the contributions of the memory elements such as capacitors and inductors,
- $\boldsymbol{f}(\boldsymbol{x}(t)) \in \mathbb{R}^{n \times 1}$ contains the contributions of the nonlinear elements such as diodes and transistors,
- $\boldsymbol{b}(t) \in \mathbb{R}^{n \times 1}$ contains the contributions of the dependent and independent current and voltage sources,
- $\boldsymbol{x}(t) \in \mathbb{R}^{n \times 1}$ is the unknown node voltages and currents,
- and n is the size of the MNA equations.

2.1.2 Circuit Analysis

Several types of simulations could be done on the circuit netlist to obtain different types of solutions. The most common types are discussed here.

DC Analysis

The most basic type of analysis is DC analysis which provides the DC node voltages and branch currents of the circuit. Finding the DC solution is almost always required for all other types of simulations. For example, it serves as a starting point for transient simulation and as an operating point for AC analysis.

The MNA equations for a circuit containing several linear and nonlinear elements can be written as:

$$\boldsymbol{G}\boldsymbol{x}(t) + \boldsymbol{C}\dot{\boldsymbol{x}}(t) + \boldsymbol{f}(\boldsymbol{x}(t)) = \boldsymbol{b}(t) \tag{2.5}$$

For DC analysis the input is constant and therefore the derivative of the voltages and currents with respect to time, $\dot{\boldsymbol{x}}(t)$, will be zero, simplifying the above differential nonlinear equations to only algebraic nonlinear equations:

$$\boldsymbol{G}\boldsymbol{x} + \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{b} \tag{2.6}$$

These nonlinear equations can then be solved using iterative methods like the Newton-Raphson method. The problem now is transformed into solving for the root of the following equation:

$$\boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{b} \tag{2.7}$$

The method begins by choosing an initial guess $\boldsymbol{x_0}$ for the root and then finding a better approximation of the solution at every iteration as follows:

$$\boldsymbol{x}_{new} = \boldsymbol{x}_{old} + \Delta\boldsymbol{x} \tag{2.8}$$

where

$$\Delta\boldsymbol{x} = -\frac{\boldsymbol{\Phi}(\boldsymbol{x}_{old})}{\boldsymbol{\Phi}'(\boldsymbol{x}_{old})} \tag{2.9}$$

and

$$\boldsymbol{\Phi}'(\boldsymbol{x}_{old}) = \left.\frac{\partial\boldsymbol{\Phi}(\boldsymbol{x})}{\partial\boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}_{old}} = \boldsymbol{G} + \left.\frac{\partial\boldsymbol{f}(\boldsymbol{x})}{\partial\boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}_{old}} \tag{2.10}$$

The method converges and the solution is reached when Δx is less than a chosen error tolerance.

AC Analysis

A second common type of simulation performed by SPICE is AC analysis. This type of analysis can only be done on linear circuits or linearized nonlinear circuits. Consider the MNA equations for a linear circuit:

$$\boldsymbol{G}\boldsymbol{x}(t) + \boldsymbol{C}\dot{\boldsymbol{x}}(t) = \boldsymbol{b}(t) \tag{2.11}$$

If the input to the circuit is a sinusoidal source of frequency ω, then in phasor form we can write:

$$\boldsymbol{b}(t) \rightarrow \boldsymbol{B}e^{j\omega t} \tag{2.12}$$

and

$$\boldsymbol{x}(t) \rightarrow \boldsymbol{X}e^{j\omega t} \tag{2.13}$$

where $\boldsymbol{B}$ and $\boldsymbol{X}$ are complex vectors containing the amplitude and phase of the input and the solution vectors respectively.

The MNA equations expressed in phasor form become:

$$\boldsymbol{G}\boldsymbol{X}e^{j\omega t} + \boldsymbol{C}\frac{\partial \boldsymbol{X}e^{j\omega t}}{\partial t} = \boldsymbol{B}e^{j\omega t} \tag{2.14}$$

After differentiation and canceling out $e^{j\omega t}$, the MNA equations can be rewritten as:

$$\boldsymbol{G}\boldsymbol{X} + j\omega\boldsymbol{C}\boldsymbol{X} = \boldsymbol{B} \tag{2.15}$$

The unknown vector $\boldsymbol{X}$ can then be easily found as follows:

$$\boldsymbol{X} = (\boldsymbol{G} + j\omega\boldsymbol{C})^{-1}\boldsymbol{B} \tag{2.16}$$

Transient Analysis

The third most common type of analysis is the transient simulation. The simulation starts with a given initial condition, usually the DC solution of the circuit, and then it steps forward in time using numerical integration.

For illustration, consider the Backward Euler Rule, which is a first order method defined as:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + h\dot{\boldsymbol{x}}_{n+1} \tag{2.17}$$

and so

$$\dot{\boldsymbol{x}}_{n+1} = \frac{1}{h}\left(\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\right) \tag{2.18}$$

For simplicity, at time t_{n+1}, the MNA equations can be expressed as:

$$\boldsymbol{G}\boldsymbol{x}_{n+1} + \boldsymbol{C}\dot{\boldsymbol{x}}_{n+1} + \boldsymbol{f}(\boldsymbol{x}_{n+1}) = \boldsymbol{b}_{n+1} \tag{2.19}$$

Substituting Equation (2.18) into Equation (2.19) would result in the following equation:

$$\boldsymbol{G}\boldsymbol{x}_{n+1} + \frac{\boldsymbol{C}}{h}\left(\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\right) + \boldsymbol{f}(\boldsymbol{x}_{n+1}) = \boldsymbol{b}_{n+1} \tag{2.20}$$

After reordering, the Backward Euler difference equation can be written as:

$$\left(\boldsymbol{G} + \frac{\boldsymbol{C}}{h}\right)\boldsymbol{x}_{n+1} + \boldsymbol{f}(\boldsymbol{x}_{n+1}) = \boldsymbol{b}_{n+1} + \frac{\boldsymbol{C}}{h}\boldsymbol{x}_n \tag{2.21}$$

The above set of equations are nonlinear algebraic equations which can be solved using iterative methods similar to the one described in the DC analysis section above.

Harmonic Balance

For a linear circuit with an input of frequency f_1, the output will simply have the same frequency f_1. This is not the case in nonlinear circuits, where the output would have an amplitude at the input frequency f_1, referred to as the fundamental, and an amplitude at every multiple of that fundamental, f_1, $2f_1$, $3f_1 \ldots$, which are referred to as the harmonics. Furthermore, if the input has two fundamental frequencies f_1 and f_2, then the output would have amplitudes at both frequencies, their harmonics, and the addition and subtraction of them and their harmonics. This is primarily why traditional frequency domain solutions do not work here and instead the Harmonic Balance technique is used [23, 24].

As an illustration, assume the input to a nonlinear circuit is periodic with a fundamental frequency ω, then the output is also periodic with a fundamental frequency ω and its harmonics are 2ω, 3ω, etc. A good way to describe such a signal is by using the Fourier transform, in which case every unknown voltage and current in the circuit could be written as:

$$x(t) = a_o + \sum_{k=1}^{H} (a_k \cos(\omega_k t) + b_k sin(\omega_k t)) \tag{2.22}$$

and

$$\dot{x}(t) = \sum_{k=1}^{H} (b_k \omega_k \cos(\omega_k t) + a_k \omega_k \sin(\omega_k t)) \tag{2.23}$$

The nonlinear vector $\boldsymbol{f}(\boldsymbol{x}(t))$ can also be expressed as:

$$f(t) = f_o + \sum_{k=1}^{H} (f_{ck} \cos(\omega_k t) + f_{sk} sin(\omega_k t)) \tag{2.24}$$

Substituting the Fourier coefficients of the solution vector and its derivative with respect to time, the nonlinear function vector, and the source vector into the MNA equations of the circuit results in what is referred to as the Harmonic Balance equations expressed as:

$$\bar{\boldsymbol{G}}\bar{\boldsymbol{X}} + \bar{\boldsymbol{C}}\bar{\boldsymbol{X}} + \boldsymbol{F}(\bar{\boldsymbol{X}}) = \bar{\boldsymbol{B}} \tag{2.25}$$

where

- $\bar{\boldsymbol{X}} \in \mathbb{R}^{N_{hb} \times 1}$ is a vector that contains the unknown Fourier coefficients of $\boldsymbol{x}(t)$,
- $\bar{\boldsymbol{B}} \in \mathbb{R}^{N_{hb} \times 1}$ is a vector that contains the Fourier coefficients of $\boldsymbol{b}(t)$,
- $\boldsymbol{F}(\bar{\boldsymbol{X}}) \in \mathbb{R}^{N_{hb} \times 1}$ is a vector that contains the Fourier coefficients of $\boldsymbol{f}(\boldsymbol{x}(t))$,

- $\bar{\boldsymbol{G}} \in \mathbb{R}^{N_{hb} \times N_{hb}}$ is a block matrix whose blocks $\boldsymbol{G}_{ij} \in \mathbb{R}^{N_h \times N_h}$ are

$$\boldsymbol{G}_{ij} = \text{diag}(g_{ij}, \ldots, g_{ij})$$

  where g_{ij} is the corresponding element in the original $\boldsymbol{G}$ matrix of the MNA equations,

- $\bar{\boldsymbol{C}} \in \mathbb{R}^{N_{hb} \times N_{hb}}$ is a block matrix whose blocks $\boldsymbol{C}_{ij} \in \mathbb{R}^{N_h \times N_h}$ are

$$\boldsymbol{C}_{ij} = c_{ij} \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \omega & \cdots & 0 & 0 \\ 0 & -\omega & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & H\omega \\ 0 & 0 & 0 & \cdots & -H\omega & 0 \end{bmatrix}$$

  where c_{ij} is the corresponding element in the original $\boldsymbol{C}$ matrix of the MNA equations,

- $N_{hb} = n \times N_h$,

- $N_h = 2H + 1$,

- H is the number of harmonics,

- and n is the size of the time domain MNA equations.

The above set of equations are nonlinear algebraic equations which can be solved using iterative methods such as Newton-Raphson. The challenge here is that the size of those equations is now much larger than the size of the original time domain equations, and therefore more computationally expensive. To solve using Newton-Raphson method, the problem is now finding the root of the following equation:

$$\bar{\boldsymbol{\Phi}}(\bar{\boldsymbol{X}}) = \bar{\boldsymbol{G}}\bar{\boldsymbol{X}} + \bar{\boldsymbol{C}}\bar{\boldsymbol{X}} + \boldsymbol{F}(\bar{\boldsymbol{X}}) - \bar{\boldsymbol{B}} \tag{2.26}$$

The method begins by choosing an initial guess $\bar{\boldsymbol{X}}_{\mathbf{0}}$, generally the DC solution, for the root and then finding a better approximation of the solution at every iteration as follows:

$$\bar{\boldsymbol{X}}_{new} = \bar{\boldsymbol{X}}_{old} + \Delta\bar{\boldsymbol{X}} \tag{2.27}$$

where

$$\Delta\bar{\boldsymbol{X}} = -\frac{\bar{\boldsymbol{\Phi}}(\bar{\boldsymbol{X}}_{old})}{\bar{\boldsymbol{\Phi}}'(\bar{\boldsymbol{X}}_{old})} \tag{2.28}$$

and

$$\bar{\boldsymbol{\Phi}}'(\bar{\boldsymbol{X}}_{old}) = \left.\frac{\partial\bar{\boldsymbol{\Phi}}(\bar{\boldsymbol{X}})}{\partial\bar{\boldsymbol{X}}}\right|_{\bar{\boldsymbol{X}}=\bar{\boldsymbol{X}}_{old}} = \bar{\boldsymbol{G}} + \bar{\boldsymbol{C}} + \left.\frac{\partial\boldsymbol{F}(\bar{\boldsymbol{X}})}{\partial\bar{\boldsymbol{X}}}\right|_{\bar{\boldsymbol{X}}=\bar{\boldsymbol{X}}_{old}} \tag{2.29}$$

To evaluate the nonlinear part of the Jacobian above, it is first required to evaluate it in the time-domain. To do so, we calculate the solution vector in the time domain using the Inverse Direct Fourier Transform (IDFT):

$$\boldsymbol{x}(t) = \Gamma\bar{\boldsymbol{X}} \tag{2.30}$$

Next, the derivative of the nonlinear function is evaluated in the time domain. Using the Direct Fourier Transform (DFT) we can then evaluate the nonlinear part of the Jacobian as follows:

$$\begin{aligned}\frac{\partial \boldsymbol{F}(\bar{\boldsymbol{X}})}{\partial \bar{\boldsymbol{X}}} &= \frac{\partial \Gamma^{-1}\boldsymbol{f}(\boldsymbol{x}(t))}{\partial \bar{\boldsymbol{X}}} \\ &= \Gamma^{-1}\frac{\partial \boldsymbol{f}(\boldsymbol{x}(t))}{\partial \bar{\boldsymbol{X}}} \\ &= \Gamma^{-1}\frac{\partial \boldsymbol{f}(\boldsymbol{x}(t))}{\partial \boldsymbol{x}(t)}\frac{\partial \boldsymbol{x}(t)}{\partial \bar{\boldsymbol{X}}} \\ &= \Gamma^{-1}\frac{\partial \boldsymbol{f}(\boldsymbol{x}(t))}{\partial \boldsymbol{x}(t)}\Gamma\end{aligned} \tag{2.31}$$

While the time domain derivative of the nonlinear function is not necessarily a dense matrix, multiplying and premultiplying by Γ and its inverse makes for a very dense matrix. Since the number of HB equations is already pretty large, combined with the increased density, this makes inverting the Jacobian very computationally expensive.

Nonetheless, the method converges and the solution is reached when Δx is less than a predefined error tolerance.

2.2 Artificial Neural Networks

Artificial neural networks (ANNs) are machine learning algorithms which have demonstrated a remarkable ability in approximating mathematical functions. Inspired by biological neural networks, artificial neural networks are made up of simple processing units known as artificial neurons. These neurons are arranged in layers and are connected together in various ways. Neurons that receive input from outside the network constitute the input layer, while neurons that provide an output make up the output layer. All other neurons in between are said to be hidden and could be arranged in one or more hidden layers [25].

Every artificial neuron in the network handles inputs from several other neurons and generates an output, which is in turn fed into other neurons or used as an output of the

ANN. Every connection through which the neuron receives an input has a weight associated with it. The way the neuron handles those inputs is determined by the type of its activation function.

The power of neural networks lies in their ability to *learn* from the data presented to them. Original data from the function/system is used in a process referred to as *training*, whereby the neural network adjusts the connection weights to best mimic the original data. So while the connections, activation functions, and number of neurons have to be predetermined, it is the weights associated with the connections that are adjusted during the training process.

Different types of ANNs are distinguished based on the activation functions used by their neurons and the way the neurons are connected [26]. In what follows we present the most common ANN architectures used in the context of circuit simulation.

2.2.1 Feedforward Neural Networks

One of the most common type of neural networks used today is the feedforward neural network. As the name suggests, information in a feedforward ANN flows in one direction: forward. Feedforward ANNs are classified based on the activation function used by their neurons. In circuit simulation, there are three common types: multilayer perceptron (MLP), radial basis function (RBF) networks, and wavelet networks. Other feedforward structures, which are mainly inspired from some knowledge of the underlying circuit under test, exist and will be presented in a later section.

Multilayer Perceptron

MLPs are by far the most common type of neural networks used today, mainly due to their well-established training algorithm, the backward propagation of errors, more commonly known as backpropagation. In fact, in many cases MLPs are sometimes referred to as

backpropagation neural networks precisely because of this. Figure 2.3 shows a three layer MLP with m-inputs and n-outputs.

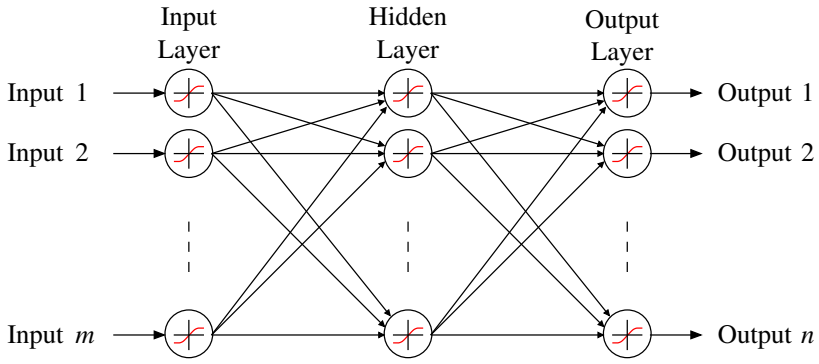

Fig. 2.3 A three layer MLP

The general structure of an artificial neuron of an MLP is shown in Figure 2.4. Each neuron receives inputs from every neuron of the previous layer. If this neuron is in the input layer, then its inputs are the inputs of the neural network itself. Every input is then multiplied by a weight associated with it and the result is added to a bias specific to each neuron. This summation is then later processed by the activation function of the neuron before being provided as the output.

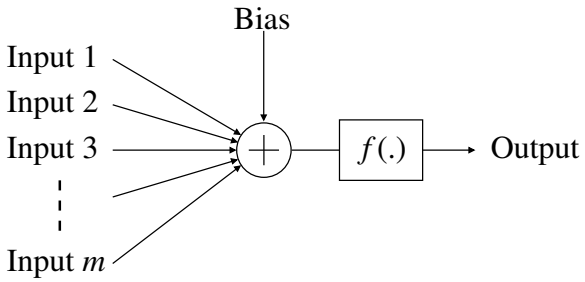

Fig. 2.4 The general structure of an artificial neuron

There are three types of activation functions which are commonly used in MLPs. The first is the log-sigmoid function shown in Figure 2.5 and expressed in Equation 2.32. The

second is the hyperbolic tangent sigmoid function shown in Figure 2.6 and expressed in Equation 2.33. Lastly, the purely linear function is shown in Figure 2.7 and expressed in Equation 2.34.

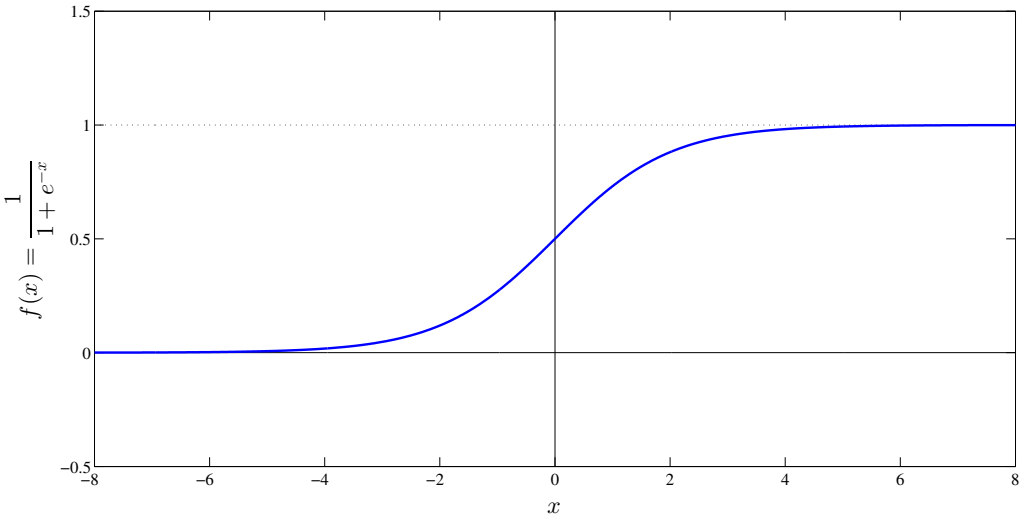

Fig. 2.5 The log-sigmoid activation function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.32}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.33}$$

$$f(x) = x \tag{2.34}$$

A major advantage MLPs have is that they are proven to be universal approximators [27]. That is to say that it has been mathematically proven that an MLP with sufficient neurons in the hidden layer is capable of approximating any continuous function with any desired degree

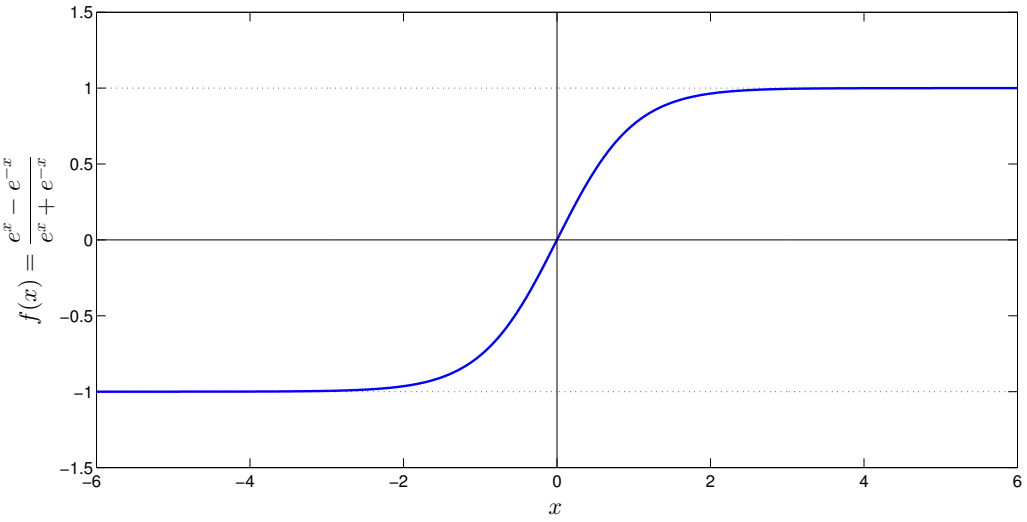

Fig. 2.6 The tan-sigmoid activation function

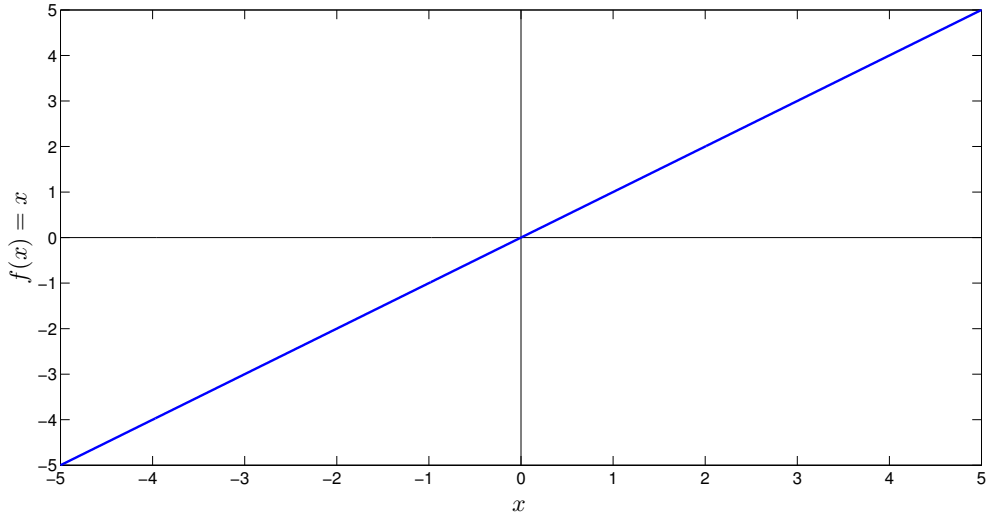

Fig. 2.7 The linear activation function

of accuracy. As of now, we do not possess a way to determine the number of neurons needed to approximate the desired function. So if a trained MLP was not able to approximate a function, it could be deduced that we either did not use enough neurons in the hidden layer, we do not have enough training data, or the function is random and not continuous [26].

Another major advantage MLPs have is their very well-established training algorithm, backpropagation [28]. The basic idea of backpropagation is as follows. A forward pass of information takes place first and the outputs are calculated. Next, the outputs are compared with the training data. For the output neurons, the weights are adjusted such that the error difference between the MLPs' outputs and the training data is minimized. For the hidden and input neurons, the weights are also adjusted in the same manner, however the proceeding neurons have to be taken into account.

To illustrate how this works mathematically, consider the two-layer MLP shown in Figure 2.8. The main components of both layers are shown for clarity and x and y are the inputs and outputs respectively. w_1 and w_2 are matrices containing the weights associated with the neurons of layer 1 and layer 2 respectively. f_1 and f_2 are the activation functions of the neurons of layer 1 and layer 2 respectively. Finally, a, b, and c are the outputs of the intermediate stages as shown in the figure.

The first step is to define the error E.

$$E = -\frac{1}{2}(d - y)^2 \tag{2.35}$$

where d is the output provided by the training data.

Next, the weights are updated according to the following equation to minimize the error.

$$w_{ij} = w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}} \tag{2.36}$$

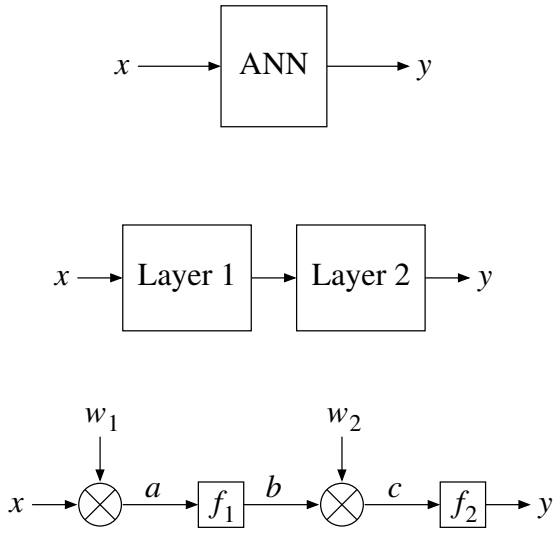

Fig. 2.8 General structure of a feedforward neural network

where α is the learning rate.

To find the derivative of the error with respect to the weights, we could use the following equation:

$$\frac{\partial E}{\partial w_{ij}} = -(d-y)\frac{\partial (d-y)}{\partial w_{ij}} = (d-y)\frac{\partial y}{\partial w_{ij}} \tag{2.37}$$

To find the effect of each weight on the output, we compute the derivative of the output with respect to every weight as shown in the equations below:

$$\frac{\partial y}{\partial w_2} = \frac{\partial y}{\partial c}\frac{\partial c}{\partial w_2} = \frac{\partial f_2(c)}{\partial c}\frac{\partial (w_2 \times b)}{\partial w_2} = b\frac{\partial f_2(c)}{\partial c} \tag{2.38}$$

$$\frac{\partial y}{\partial w_1} = \frac{\partial y}{\partial c} \times \frac{\partial c}{\partial b} \times \frac{\partial b}{\partial a} \times \frac{\partial a}{\partial w_1} = \frac{\partial f_2(c)}{\partial c} \times w_2 \times \frac{\partial f_1(a)}{\partial a} \times x \tag{2.39}$$

It should be noted that one major advantage here is that there is no need to compute the derivatives of the activation functions in the above equation. Through some simple mathe-

matical manipulation, it can be shown that the value of the derivative could be computed from the function itself as expressed below:

$$f(x) = \frac{1}{1+e^{-x}} \implies \frac{\partial f(x)}{\partial x} = f(x)\,(1 - f(x)) \tag{2.40}$$

Radial Basis Function Network

Another common type of feedforward neural networks is the radial basis function (RBF) neural network. It is similar to MLPs, but with only one hidden layer with an RBF activation function, which is shown in Figure 2.10 and expressed in Equation 2.41. The basic general structure of an RBF network is shown in Figure 2.9.

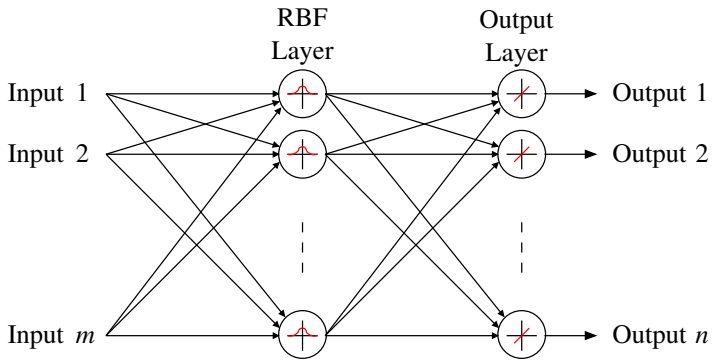

Fig. 2.9 Radial basis function network

$$f(x) = e^{x^2} \tag{2.41}$$

Like MLPs, RBF networks have also been proven to be universal approximators [29], giving them a huge advantage over other ANN structures. In terms of training, RBF networks are trained in two steps [26]. The first step is determining the centers of the activation functions of the hidden neurons. The second training step is a backpropagation to adjust

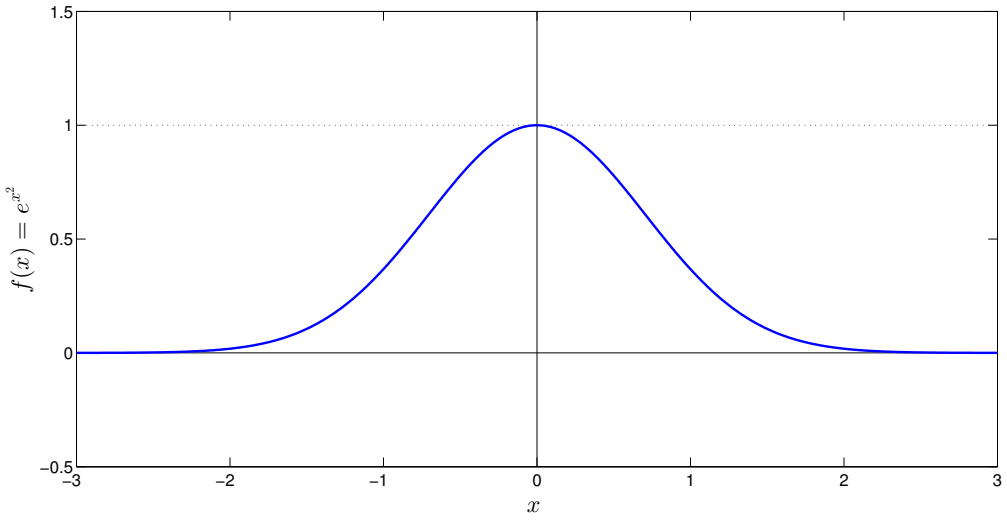

Fig. 2.10 The radial basis activation function

the weights of the connections.

Wavelet Neural Networks

A third type of feedforward neural networks used in circuit simulation are wavelet neural networks [30]. They are similar to RBF networks, with one hidden layer with its activation functions being the wavelet function. For more details on the wavelet functions and the wavelet transform, the reader is referred to [26]. More details on how wavelet neural networks are used in circuit simulation are presented in the next chapter.

2.2.2 Recurrent Neural Networks

The second major type of artificial neural networks used today both in general and in circuit simulation is recurrent neural networks (RNNs). The general structure of an RNN is shown in Figure 2.11.

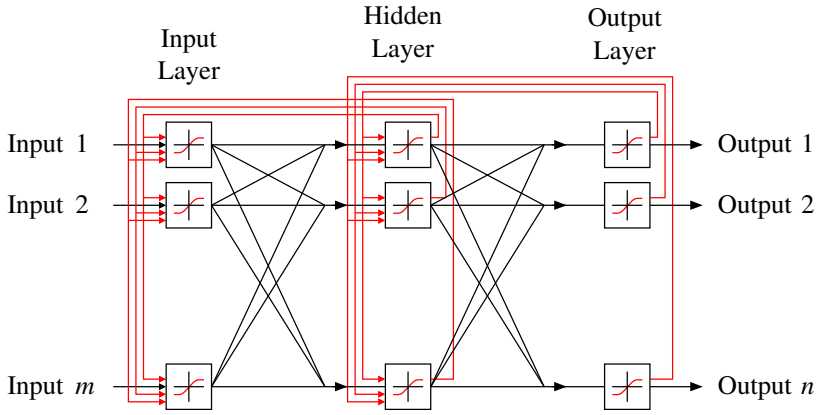

Fig. 2.11 General structure of a recurrent neural network

The biggest difference between feedforward networks and RNNs is the presence of feedback. Outputs of RNNs can be used as inputs to input and hidden neurons. This is done in a delayed fashion where the current outputs are stored to be later used in the next forward pass. Otherwise, activation functions and the overall structure of RNNs remain similar to feedforward networks. Training of RNNs is also adjusted to take into account the presence of feedback.

2.2.3 Other Structures

Several other structures of artificial neural networks have been used and proposed in the literature for use in circuit simulation. These structures are not standard, systematic, or general, but rather specific to certain applications. This is why their study is left for the next chapter, where they will be covered.

Chapter 3

Review of Macromodeling Techniques

In this chapter, a brief review of the macromodeling techniques used in circuit simulation is presented. After presenting the general types of macromodeling, special attention is given to the types related to this thesis: model order reduction and neural networks.

3.1 Types of Macromodeling

In the context of circuit simulation, there are four main macromodeling techniques used by designers today [18, 20].

3.1.1 Manual Abstraction

Manual macromodeling, or an automated version of it, is one of the earliest macromodeling techniques used by designers. An engineer attempts to create a macromodel of the original circuit by capturing specific circuit behavior and representing it in a reduced-order form. Manual macromodeling requires advanced expertise and detailed knowledge of the internal structure of the circuit block under consideration and hence is generally carried out by the

original designer. Examples can be found in [31, 32]. This technique is unsystematic, time consuming, and error prone [18, 20].

3.1.2 Symbolic Analysis

First appeared in the 1960s, this type of macromodeling represents all circuit elements and variables by symbols. The simulator can then proceed to use the symbolic circuit to produce a symbolic expression of its input-output behavior. This expression could be further simplified to produce a reduced-order macromodel of the circuit block [33–36].

3.1.3 Black-Box Methods

Macromodeling techniques based on black-box methods assume no knowledge of or have no access to the internal structure of the circuit block, but rather a collection of simulated or measured input and output data is available. Using this data, a reduced-order macromodel can then be generated. Examples include artificial neural networks [37–39], vector fitting [40, 41], and the Loewner matrix approach [42–44].

3.1.4 Model Order Reduction

Algorithmic macromodeling [16,18], more commonly known as model order reduction (MOR), starts with a set of mathematical equations that describes the behavior of the circuit. It then attempt to replace this set of equations with a smaller one without too much loss in accuracy. This could be done in several ways and will be explored in the following two sections.

MOR has several advantages over the other methods. First, it tends to be more general. Since MOR is applied to the numerical equations describing the original system, it allows for one macromodeling algorithm to be used on several types of systems as long as they are presented in the same fashion [20]. Second, since the macromodels are directly generated from

the original system itself, MOR makes use of more information to create accurate macromodels with significantly less effort [45]. Third, better error estimation is possible because of the presence of the original system [45]. Lastly, second-order effects or perturbations can also be included in the macromodel because of the availability of the original system [20].

3.2 Linear Model Order Reduction

The first linear model order reduction techniques focused on macromodeling large interconnect networks in an attempt to reduce their simulation time. These techniques can generally be split into three categories: direct moment matching, projection methods, and truncated balanced realization.

3.2.1 Direct Moment Matching

Asymptotic waveform evaluation (AWE), introduced in 1990, is a transfer function fitting method that matches the first few moments of the transfer function to a rational Padé approximation [46–49].

To illustrate how this method works, consider the MNA equations in the frequency domain of a linear circuit shown below:

$$\boldsymbol{G}\boldsymbol{X}(s) + s\boldsymbol{C}\boldsymbol{X}(s) = \boldsymbol{b}(s) \tag{3.1}$$

where,

$$X(s) = \begin{bmatrix} \vdots \\ V_{out}(s) \\ \vdots \end{bmatrix} \tag{3.2}$$

and V_{out} is the output of the circuit whose transfer function we are interested in finding.

To generate the moments of the circuit, the transfer function $h(s)$ is expanded in a Taylor series:

$$V_{out}(s) = h(s) = m_0 + m_1 s + m_2 s^2 + \dots \tag{3.3}$$

Substituting the moments in the MNA equations gives:

$$\boldsymbol{G}(M_0 + M_1 s + \cdots + M_n s^n) + s\boldsymbol{C}(M_0 + M_1 s + \cdots + M_n s^n) = \boldsymbol{b} \tag{3.4}$$

where,

$$M_0 = \begin{bmatrix} \vdots \\ m_0 \\ \vdots \end{bmatrix}, \quad M_1 = \begin{bmatrix} \vdots \\ m_1 \\ \vdots \end{bmatrix}, \quad M_2 = \begin{bmatrix} \vdots \\ m_2 \\ \vdots \end{bmatrix}, \quad \dots \tag{3.5}$$

To calculate the moments, s is set to zero and the coefficients of the powers of s are equated on both sides of the equation:

$$\begin{aligned} M_0 &= \boldsymbol{G}^{-1}\boldsymbol{b} \\ M_1 &= -\boldsymbol{G}^{-1}\boldsymbol{C}M_0 \\ &\vdots \\ M_i &= -\boldsymbol{G}^{-1}\boldsymbol{C}M_{i-1} \end{aligned} \tag{3.6}$$

Now that the moments are calculated, we can find the approximation for the transfer function. This can be done using a Padé rational function of an order L/M.

$$h(s) = m_0 + m_1 s + m_2 s^2 + \cdots = \frac{a_0 + a_1 s + a_2 s^2 + \cdots + a_L s^L}{1 + b_1 s + b_2 s^2 + \cdots + b_M s^M} \tag{3.7}$$

Cross multiplying the above equation results in:

$$\begin{bmatrix} m_{L-M+1} & m_{L-M+2} & \cdots & m_L \\ m_{L-M+2} & m_{L-M+3} & \cdots & m_{L+1} \\ \vdots & \vdots & \ddots & \vdots \\ m_L & m_{L+1} & \cdots & m_{L+M-1} \end{bmatrix} \begin{bmatrix} b_M \\ b_{M-1} \\ \vdots \\ b_1 \end{bmatrix} = - \begin{bmatrix} m_{L+1} \\ m_{L+2} \\ \vdots \\ m_{L+M} \end{bmatrix} \tag{3.8}$$

By equating the coefficients of the powers of s, we can calculate:

$$\begin{aligned} a_0 &= m_0 \\ a_1 &= m_0 b_1 + m_1 \\ a_2 &= m_0 b_2 + m_1 b_1 + m_2 \\ &\vdots \\ a_L &= m_L + \sum_{n=1}^{min(L,M)} b_n m_{L-n} \end{aligned} \tag{3.9}$$

Complex frequency hopping (CFH) introduced in 1995 extends the concept of AWE to multi-point expansions in order to capture more dominant poles [50].

3.2.2 Projection Methods

Krylov based reduction offers two main advantages over direct moments matching. First, it is possible to capture many more dominant poles in one expansion. Second, the reduced-order macromodels created are passive by construction.

To illustrate how this works, consider the frequency domain MNA equations of a linear circuit:

$$\boldsymbol{G}\boldsymbol{X}(s) + s\boldsymbol{C}\boldsymbol{X}(s) = \boldsymbol{B} \tag{3.10}$$

where

$$\boldsymbol{X}(s) = M_0 + M_1 s + M_2 s^2 + M_3 s^3 + \dots \tag{3.11}$$

and

$$\begin{aligned}
\boldsymbol{G}M_0 = \boldsymbol{B} &\implies M_0 = \boldsymbol{G}^{-1}\boldsymbol{B} \\
\boldsymbol{G}M_1 = -\boldsymbol{C}M_0 &\implies M_1 = -\boldsymbol{G}^{-1}\boldsymbol{C}M_0 \\
\boldsymbol{G}M_2 = -\boldsymbol{C}M_1 &\implies M_2 = -\boldsymbol{G}^{-1}\boldsymbol{C}M_1 \\
&\vdots \\
\boldsymbol{G}M_{n+1} = -\boldsymbol{C}M_n &\implies M_{n+1} = -\boldsymbol{G}^{-1}\boldsymbol{C}M_n
\end{aligned} \tag{3.12}$$

Consider the matrix $\boldsymbol{K}$ defined as

$$\boldsymbol{K} = [M_0, M_1, M_2, \dots M_q] \tag{3.13}$$

Because of the nature of circuits, the above Krylov subspace will contain many redundant directions and therefore will be ill-conditioned. To overcome this, we can construct an orthonormal basis $\boldsymbol{Q}$ of this subspace:

$$\text{colsp}[\boldsymbol{Q}] = \text{colsp}[\boldsymbol{K}] \tag{3.14}$$

Model order reduction is then applied by congruence transformation using the orthonormal basis $\boldsymbol{Q}$.

$$\boldsymbol{G}\boldsymbol{X}(s) + s\boldsymbol{C}\boldsymbol{X}(s) = \boldsymbol{B} \tag{3.15}$$

First, a change of variables is made:

$$\boldsymbol{X}(s) = \boldsymbol{Q}\hat{\boldsymbol{X}}(s) \tag{3.16}$$

and $\boldsymbol{X}(s)$ is replaced in the equation:

$$\boldsymbol{G}\boldsymbol{Q}\hat{\boldsymbol{X}}(s) + s\boldsymbol{C}\boldsymbol{Q}\hat{\boldsymbol{X}}(s) = \boldsymbol{B} \tag{3.17}$$

Next is premultiplying by $\boldsymbol{Q}^T$

$$\boldsymbol{Q}^T\boldsymbol{G}\boldsymbol{Q}\hat{\boldsymbol{X}}(s) + s\boldsymbol{Q}^T\boldsymbol{C}\boldsymbol{Q}\hat{\boldsymbol{X}}(s) = \boldsymbol{Q}^T\boldsymbol{B} \tag{3.18}$$

The equation can now be rewritten as

$$\hat{\boldsymbol{G}}\hat{\boldsymbol{X}}(s) + s\hat{\boldsymbol{C}}\hat{\boldsymbol{X}}(s) = \hat{\boldsymbol{B}} \tag{3.19}$$

where

$$\hat{\boldsymbol{G}} = \boldsymbol{Q}^T\boldsymbol{G}\boldsymbol{Q} \tag{3.20a}$$

$$\hat{\boldsymbol{C}} = \boldsymbol{Q}^T\boldsymbol{C}\boldsymbol{Q} \tag{3.20b}$$

$$\hat{\boldsymbol{B}} = \boldsymbol{Q}^T\boldsymbol{B} \tag{3.20c}$$

The most known of these methods are Padé via Lanczos [51–53], Arnoldi [54], Congruence Transformations [55–57], and PRIMA [58–60].

3.2.3 Truncated Balanced Realization

Truncated Balanced Realization (TBR) was borrowed from control theory [61, 62], and first appeared in circuit simulation in [63], then later expanded in [64–66]. TBR is more accurate than other methods and has an error bound, however it is computationally very expensive to extract the macromodel. It is usually applied as the last step in combination with another macromodeling technique that had been used to perform the first reduction step [66–68].

3.3 Nonlinear Model Order Reduction

Linear MOR techniques have been studied extensively with a high degree of success, and today we can say that we have a strong foundation for multiple techniques that can serve various macromodeling purposes. Unfortunately, the same is not true for nonlinear model order reduction. To this day, we still lack a comprehensive unified foundation for reducing the order of complexity of nonlinear circuits [19]. Furthermore, we do not even posses a single method which is capable of generating accurate nonlinear reduced-order macromodels suitable for the general nonlinear system [18].

There are several reasons behind this. First, unlike linear systems for which we have a thorough mathematical understanding, nonlinear systems are more complex and varied and have not been fully understood yet [18]. Second, reducing the nonlinear functions is not straightforward and in many cases the original non-reduced functions still have to be evaluated [69, 70]. Lastly, we have no guarantee on how well the macromodel will perform and all we can achieve is a simple comparison with the original system given a specific set

of inputs [16].

Nonlinear macromodeling techniques can be generally split into three main categories: time-varying, polynomial, and trajectory approximations.

3.3.1 Time-Varying Approximations

The first attempt at macromodeling nonlinear circuits was in 1998 when Roychowdhury [71] realized that it is possible to model a specific type of nonlinear circuits as a linear time-varying system. The input-output relationship of an RF mixer, for example, is indeed linear but time-shifted despite the strongly nonlinear behavior of the circuit itself. The technique, called time-varying Padé (TVP), first transforms the LTV system to an LTI systems by adding extra inputs to capture the time varying nature of the circuit. Next, a reduced-order model is obtained by applying any MOR technique to the new LTI equations. The last step is to reformulate the reduced LTI system back to its original LTV form.

3.3.2 Polynomial Approximations

Time-varying approximations presented above are only suitable for a specific type of nonlinear circuits where the input-output relationship is indeed linear but time shifted. However, for the general purpose nonlinear circuit where the input-output relationship is inherently nonlinear, time-varying approximations can no longer be used. Nonlinear model order reduction based on polynomial approximations were first introduced in 1999 by Roychowdhury [72] as an extension to his TVP method to model nonlinearities more accurately using the Volterra series. The main idea is to replace the original nonlinear functions of the circuit equations by a set of linear equations that represent and then later compute the distortions caused by the nonlinearities.

The main advantage of this technique is that instead of solving the original nonlinear

system, here we are solving a set of linear equations. By doing so, we can now make use of the well-established linear model order reduction techniques to reduce the size those equations. Each linear equation can be reduced to a smaller equation, for example using some projection technique. If the reduced subspace of x_1 is q_1, x_2 is q_2 and so on, then the reduced subspace of the whole system is the union of all the individual reduced subspaces [18]. Another interesting observation from using this technique is that the perturbation responses x_1, x_2, ... correspond to quantities of interest for circuit designers such as distortion and intermodulation [18]. The main drawback in using Volterra series approximation is that those type of algorithms are limited to weakly nonlinear systems. The reason behind that is that the underlying assumption here is that the first linear term is much larger than the perturbation terms.

From the above description, one realizes that the overall reduced subspace containing the union of all the individual reduced subspaces increases when more perturbation calculations are required. To overcome this, in [70] Phillips suggested performing a singular value decomposition on the reduced subspaces which results in a single reduced subspace. This new subspace could be used to perform all the projections resulting in smaller equation size. Furthermore, Phillips in [45,69] proposed rewriting the nonlinear system as a bilinear system of higher dimensionality. The advantage here is that the bilinear system is easier to solve.

NORM

In 2003, an important advancement in nonlinear macromodeling based on polynomial approximations was presented by Li and Pileggi in [73, 74]. The new algorithm called compact nonlinear model order reduction method (NORM) builds upon the ideas presented above.

QLMOR

Another milestone in polynomial approximation came in 2009 and was presented by Gu [75, 76]. Dubbed model order reduction via quadratic-linear systems (QLMOR), the new algorithm is also based on Volterra series approximation. The first step of the algorithm is to transform the differential algebraic equations describing the circuits under study into the special representation that is quadratic-linear differential algebraic equations. Unlike previous methods, no Taylor expansion is used here so QLMOR does not suffer from the same Taylor-related problems that the other methods suffer from. In the second step, the algorithm uses a projection-based approach to reduce the quadratic-linear differential equations resulting in a reduced macromodel. Results show that using this special representation results in a more general approach than the previous Volterra-based methods and is no longer limited to weakly nonlinear methods.

3.3.3 Trajectory Approximations

Polynomial approximations presented in the previous section were ideal for weakly nonlinear systems where the whole system is essentially biased at one point and only small perturbations around that point happen. For the general case, where a system has stronger nonlinearities and several operating points, polynomial approximations fail. Instead, researchers diverted their attention towards trajectory-based methods. The key idea here is to create a reduced-order macromodel which is valid on and limited to one specific trajectory.

Trajectory-based methods can be split into three broad categories. The first method is the projection method, where the nonlinear system is projected onto a reduced subspace using the regular linear projection approach. The key difference here is dealing with the nonlinear function, as will be shown in the following few paragraphs. The second approach

is the piecewise linear technique, where the nonlinear function is modeled as a linear piecewise function. The resulting combined linear functions are then projected on a reduced subspace via linear projection. The last trajectory-based technique is the most recent ManiMOR algorithm where a nonlinear projection is used.

Projection Methods

Projection methods in nonlinear macromodeling are a continuation of the linear ones with special attention given to the nonlinear functions. Consider again the circuit subsection containing linear and nonlinear elements expressed by Equation 3.21 below.

$$\begin{aligned} \boldsymbol{G}\boldsymbol{x}(t) + \boldsymbol{C}\dot{\boldsymbol{x}}(t) + \boldsymbol{f}(\boldsymbol{x}(t)) &= \boldsymbol{R}\boldsymbol{u}(t) + \boldsymbol{b}(t) \\ \boldsymbol{i}(t) &= \boldsymbol{R}^T\boldsymbol{x}(t) \end{aligned} \tag{3.21}$$

Using congruence transformation, the reduced-order nonlinear macromodel of this subsection could be written as shown in Equation 3.22 below.

$$\begin{aligned} \hat{\boldsymbol{G}}\hat{\boldsymbol{x}}(t) + \hat{\boldsymbol{C}}\dot{\hat{\boldsymbol{x}}}(t) + \hat{\boldsymbol{f}}(\hat{\boldsymbol{x}}(t)) &= \hat{\boldsymbol{R}}\boldsymbol{u}(t) + \hat{\boldsymbol{b}}(t) \\ \boldsymbol{i}(t) &= \hat{\boldsymbol{R}}^T\hat{\boldsymbol{x}}(t) \end{aligned} \tag{3.22}$$

where

$$\hat{\boldsymbol{G}} = \boldsymbol{Q}^T \boldsymbol{G} \boldsymbol{Q} \tag{3.23a}$$

$$\hat{\boldsymbol{C}} = \boldsymbol{Q}^T \boldsymbol{C} \boldsymbol{Q} \tag{3.23b}$$

$$\hat{\boldsymbol{R}} = \boldsymbol{Q}^T \boldsymbol{R} \tag{3.23c}$$

$$\hat{\boldsymbol{b}}(t) = \boldsymbol{Q}^T \boldsymbol{b}(t) \tag{3.23d}$$

$$\hat{\boldsymbol{f}}(\hat{\boldsymbol{x}})(t) = \boldsymbol{Q}^T \boldsymbol{f}(\boldsymbol{x}(t)) \tag{3.23e}$$

The main problem here is dealing with the nonlinear function $f(x(t))$. Every time we need to calculate the reduced version of this function, we have to go back to the original system and calculate the original function. After that, we can later project onto the reduced subspace to find the reduced order function $\hat{f}(\hat{x}(t))$. Clearly this is a major drawback of the projection methods. Calculating the nonlinear function of the original macromodel is often expensive and therefore limits any speedup achieved by reducing the other matrices. Furthermore, while a smaller size is achieved for the equations, this method is still inherently a linear order-reduction method. No real reduction of the nonlinear part is ever carried out.

Regarding the reduced subspace $\boldsymbol{Q}$, there exists several ways to construct it. One of the well established ones is proper orthogonal decomposition (POD) [77, 78]. In [77], the transient response of the circuit subsection is computed on a specific trajectory, essentially different inputs and loading conditions. The responses are then collected into a subspace $\boldsymbol{K}$:

$$K = [x_1(t) \quad x_2(t) \quad \dots x_n(t)] \tag{3.24}$$

Realizing that the above subspace contains many redundant directions, a singular value decomposition (SVD) is carried out to compute the most dominant directions of this sub-

space:

$$U\Sigma V = \mathrm{SVD}(K) \tag{3.25}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal matrices and $\boldsymbol{\Sigma}$ contains the singular values in descending order.

The reduced subspace Q is then constructed from the first few columns of U which correspond to the most dominant directions of the original subspace K.

Piecewise Linear Approach

The first attempt at piecewise linear approximations was the trajectory piecewise-linear (TPWL) technique introduced in 2001 by Rewienski and White [79,80]. The idea is to model the nonlinear system as a combination of reduced linear models. First, a training input is applied and a linearization is performed at specific points on this particular trajectory. Each linearized model is then reduced using Krylov subspace methods. The last step is to combine all the linear models to create an overall reduced model for the whole system.

For more details on the choice of weights and the reduced subspace, refer to [80]. Later, Tiwary and Rutenbar in [19,81] expanded the TPWL approach, making it scalable to larger circuits, parametric, and even faster. Furthermore, in [82, 83], Dong and Roychowdhury combined TPWL with polynomial approximations in a new method they termed piecewise polynomial (PWP). Instead of approximating the nonlinear functions with linear ones, they suggested polynomial approximations. This combines the global approximation capabilities of TPWL with the weakly nonlinear behavior that polynomial approximations are capable of capturing.

ManiMOR

The last trajectory-based approximation we discuss here was presented by Gu and Roychowdhury [84, 85] in 2008. Unlike the previous methods, which employ a linear projection scheme to generate the macromodels by projection on a reduced linear subspace, maniMOR performs a nonlinear projection on a nonlinear manifold. Moreover, unlike previous methods where the reduced subspace and the projection are done in one step, maniMOR explicitly splits them into two clear steps. In what follows, we outline the general steps of this algorithm.

The first step in this algorithm is creating the trajectory where the macromodel is considered valid. To do that, the original circuit is simulated under different conditions on this trajectory. Multiple input and different simulation types are performed, then sample points are taken based on a specific criterion. Next, the reduced nonlinear subspace, the manifold, is constructed from the sample points collected above. Lastly, the reduced-order macromodel is created by projecting the sample points onto the newly generated manifold.

The main advantage in this algorithm can be observed by noting that the nonlinear manifolds contain more information than their linear counterparts. This allows for the construction of smaller macromodels compared to other methods. In fact, the authors showed that in some cases using maniMOR, they were able to generate macromodels half the size of TPWL, but with the same accuracy.

3.4 Neural Network-Based Macromodeling

From the early days of macromodeling, artificial neural networks were used to approximate circuit behavior. ANN-based macromodeling belongs to the black box category of macromodeling. The designer does not have or even need to have access to the inner workings of

the circuit under test. Instead all they have is a set of input-output data with which they define the macromodel [27, 86–88].

Macromodeling using neural networks is very useful for the following reasons. First, no information about the original circuit is required aside from its input-output relationship represented with a set of data points. This data is then used to train a neural network to teach the behavior of the original circuit. If enough data points are available and the ANN is of a suitable structure, then the newly trained ANN can replace the original circuit and act as a reduced-order macromodel. Second, several neural network structures are proven to be universal approximators if the function they are approximating is continuous, which is the case for circuits.

In what follows, we present a short survey of the artificial neural network structures used so far in macromodeling circuit blocks. There are two common themes in those macromodeling techniques. First, many of the proposed algorithms rely on special structures of neural networks built for specific applications. The designer must know the inner workings of the subcircuit to be able to make use of those algorithms. The second theme is using a neural network to represent the whole input-output relationship of a subcircuit. This requires the usage of a special type of neural networks to capture dynamic behaviour and is not general enough for all circuits.

Feedforward neural networks were the first method of choice for circuit designers given their popularity and availability. Therefore, the first macromodeling attempts that used neural networks focused on feedforward networks with all its types. One of the earliest attempts to use artificial neural networks in circuit design came in 1996 when [89] used a feedforward neural network to model microstrip vias and interconnects.

Recurrent Neural Networks are currently among the most popular architectures used to model nonlinear circuits. In [90], the authors trained a recurrent neural network to predict

the input-output behaviour of nonlinear circuits. In [91], the authors used a similar approach to model an IO buffer. Also, in [92], the authors used recurrent neural networks to model highly nonlinear digital input/output drivers. Lastly, in [93], recurrent neural networks were used to dynamically model linear and nonlinear microwave circuits.

Radial basis neural networks have also been used for macromodeling nonlinear circuits. Specifically in [94], a radial-basis function neural network was used to model the behaviour of an RF power amplifier successfully.

In [38], an artificial neural network was used, aided by extra knowledge provided by the circuit under test. Dubbed Knowledge-based neural networks (KBNNs), the authors added extra neurons with special functions used to emulate the specific behaviour of the circuit being modeled. Given the special structure of the new network and the fact that it must be designed with a specific circuit in mind, it remains limited to those circuits and requires a significant amount of work and modifications to generalize or even modify for another type of circuit.

Similary, in [95], the author used their knowledge of the circuit being modeled to decide on the specific structure of neural network being used. While effective, a very thorough understanding and careful study of the circuit must be done in order to choose the network architecture and ensure accuracy. Here as well, the algorithm is not general enough and is tailored to specific circuits in mind.

Chapter 4

Macromodeling Radio Frequency Circuits

In this chapter, we present the proposed reduction method applied to radio frequency circuits. With the growth of the wireless communications sector, RF circuits are now widely used in numerous connected devices. The challenge with those types of circuits is that they require a special kind of analysis, known as the Harmonic Balance technique, discussed in Section 2.1.2 of this thesis. This type of analysis is necessary to study and observe specific nonlinear behaviour.

A typical RF circuit usually contains only a handful of elements, and therefore its MNA equations are small in size. However, given the nature of the Harmonic Balance technique, the resulting HB equations are typically quite large. As noted in Section 2.1.2, these equations are nonlinear algebraic equations. While there exist many techniques that are able to reduce the size of linear equations considerably, since these equations contain a nonlinear component, common model order reduction techniques will not work here straight out of the box. Instead, some modifications to those techniques are necessary to extend them for those types of

circuits and their simulations.

In this chapter, we present our proposed reduction technique that is suitable for use with radio frequency circuits. However, before we are able to do this, we need to reformulate our MNA equations into a macromodel format. This format will later allow us to insert the model into a larger circuit. With that out of the way, the first step of reduction, proper orthogonal decomposition, is used to reduce the linear part of the circuit. In the second step we use feedforward neural networks to replace the nonlinear part of the circuit. The result is a reduced-order macromodel, that has the same format as the original one (only smaller in size). The newly-formed macromodel will be able to replace the original and emulate its behaviour within a well-defined range chosen by the user.

4.1 Harmonic Balance Equations of a Nonlinear Subsection

In order to tackle the problem of creating reduced-order macromodels of nonlinear circuits, we will first reformat the MNA equations presented in Section 2.1.1 into macromodel form. This macromodel format, reduced or not, can be readily added to a larger circuit for simulation.

4.1.1 Network Formulation

Consider the multi-port circuit subsection which contains several linear and nonlinear elements shown in Figure 4.1.

The MNA formulation for this multi-port nonlinear circuit subsection in the time domain can be written as:

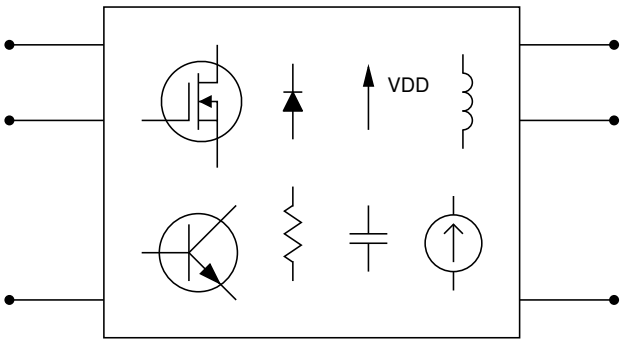

Fig. 4.1 A multi-port circuit subsection

$$\begin{aligned} \boldsymbol{G}\boldsymbol{x}(t) + \boldsymbol{C}\dot{\boldsymbol{x}}(t) + \boldsymbol{f}(\boldsymbol{x}(t)) &= \boldsymbol{R}\boldsymbol{u}(t) + \boldsymbol{b}(t) \\ \boldsymbol{i}(t) &= \boldsymbol{R}^T\boldsymbol{x}(t) \end{aligned} \tag{4.1}$$

where

- $\boldsymbol{u}(t) \in \mathbb{R}^{p\times 1}$ contains the ports' voltages,
- $\boldsymbol{i}(t) \in \mathbb{R}^{p\times 1}$ contains the ports' currents,
- $\boldsymbol{R} \in \mathbb{R}^{n\times p}$ is a selector matrix that maps the ports' voltages and currents into the state space of the circuit,
- p is the number of ports,
- and n is the size of the MNA equations.

This is the nonlinear circuit subsection, which we are interested in reducing. However, since the type of analysis that will be carried out is the Harmonic Balance technique, we need to rewrite Equation 4.1 describing this macromodel in a way that is appropriate to use with Harmonic Balance. This formulation could be written as:

$$\begin{aligned} \bar{\boldsymbol{G}}\bar{\boldsymbol{X}} + \bar{\boldsymbol{C}}\bar{\boldsymbol{X}} + \boldsymbol{F}(\bar{\boldsymbol{X}}) &= \bar{\boldsymbol{R}}\bar{\boldsymbol{U}} + \bar{\boldsymbol{B}} \\ \bar{\boldsymbol{I}} &= \bar{\boldsymbol{R}}^T \bar{\boldsymbol{X}} \end{aligned} \tag{4.2}$$

where

- $\bar{\boldsymbol{X}} \in \mathbb{R}^{N_{hb} \times 1}$ is a vector that contains the unknown Fourier coefficients of $\boldsymbol{x}(t)$,
- $\bar{\boldsymbol{B}} \in \mathbb{R}^{N_{hb} \times 1}$ is a vector that contains the Fourier coefficients of $\boldsymbol{b}(t)$,
- $\boldsymbol{F}(\bar{\boldsymbol{X}}) \in \mathbb{R}^{N_{hb} \times 1}$ is a vector that contains the Fourier coefficients of $\boldsymbol{f}(\boldsymbol{x}(t))$,
- $\bar{\boldsymbol{G}} \in \mathbb{R}^{N_{hb} \times N_{hb}}$ is a block matrix whose blocks $\boldsymbol{G}_{ij} \in \mathbb{R}^{N_h \times N_h}$ are

$$\boldsymbol{G}_{ij} = \mathrm{diag}(g_{ij}, \ldots, g_{ij})$$

  where g_{ij} is the corresponding element in the original $\boldsymbol{G}$ matrix of the MNA equations,

- $\bar{\boldsymbol{C}} \in \mathbb{R}^{N_{hb} \times N_{hb}}$ is a block matrix whose blocks $\boldsymbol{C}_{ij} \in \mathbb{R}^{N_h \times N_h}$ are

$$\boldsymbol{C}_{ij} = c_{ij} \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \omega & \cdots & 0 & 0 \\ 0 & -\omega & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & H\omega \\ 0 & 0 & 0 & \cdots & -H\omega & 0 \end{bmatrix}$$

  where c_{ij} is the corresponding element in the original $\boldsymbol{C}$ matrix of the MNA equations,

- $\bar{\boldsymbol{U}} \in \mathbb{R}^{N_{pb} \times 1}$ is a vector that contains the unknown Fourier coefficients of $\boldsymbol{u}(t)$,
- $\bar{\boldsymbol{I}} \in \mathbb{R}^{N_{pb} \times 1}$ is a vector that contains the unknown Fourier coefficients of $\boldsymbol{i}(t)$,
- $\bar{\boldsymbol{R}} \in \mathbb{R}^{N_{hb} \times N_{pb}}$ is a selector matrix that maps the Fourier coefficients of the ports' voltages and currents into the state space of the circuit,
- $N_{hb} = n \times N_h$,
- $N_{pb} = p \times N_h$,
- $N_h = 2H + 1$,
- H is the number of harmonics,
- and n is the size of the time domain MNA equations.

The nonlinear circuit subsection which we are interested in is now in a suitable format for reducing. The resulting reduced format will be identical to the original one, except smaller in size. However, in order to make use of this reduced macromodel, we should be able to add it to a larger system. This larger system might contain other macromodels and linear and nonlinear components.

4.1.2 System Formulation

Consider a circuit ϕ containing several linear and nonlinear elements in addition to m nonlinear circuit subsections. The contributions of linear and nonlinear components and the m nonlinear circuit subsections can be added to the overall equations of circuit ϕ as follows:

$$\begin{bmatrix} \boldsymbol{G}_\phi & \boldsymbol{D}_1\boldsymbol{R}_1^T & \cdots & \boldsymbol{D}_m\boldsymbol{R}_m^T \\ \boldsymbol{R}_1\boldsymbol{D}_1^T & \boldsymbol{G}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ \boldsymbol{R}_m\boldsymbol{D}_m^T & 0 & 0 & \boldsymbol{G}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_\phi(t) \\ \boldsymbol{x}_1(t) \\ \vdots \\ \boldsymbol{x}_m(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{C}_\phi & 0 & \cdots & 0 \\ 0 & \boldsymbol{C}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \boldsymbol{C}_m \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_\phi(t) \\ \dot{\boldsymbol{x}}_1(t) \\ \vdots \\ \dot{\boldsymbol{x}}_m(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{f}_\phi(\boldsymbol{x}_\phi(t)) \\ \boldsymbol{f}_1(\boldsymbol{x}_1(t)) \\ \vdots \\ \boldsymbol{f}_m(\boldsymbol{x}_m(t)) \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_\phi(t) \\ \boldsymbol{b}_1(t) \\ \vdots \\ \boldsymbol{b}_m(t) \end{bmatrix} \tag{4.3}$$

where

- $\boldsymbol{G}_\phi$, $\boldsymbol{C}_\phi$, $\boldsymbol{f}_\phi(\boldsymbol{x}_\phi(t))$ and $\boldsymbol{b}_\phi(t)$ are the vectors and matrices of the linear and nonlinear elements of circuit ϕ,
- $\boldsymbol{G}_1, \cdots, \boldsymbol{G}_m$, $\boldsymbol{C}_1, \cdots, \boldsymbol{C}_m$, $\boldsymbol{f}_1(\boldsymbol{x}_1(t)), \cdots, \boldsymbol{f}_m(\boldsymbol{x}_m(t))$, $\boldsymbol{b}_1(t), \cdots, \boldsymbol{b}_m(t)$ and $\boldsymbol{R}_1, \cdots, \boldsymbol{R}_m$ are the vectors and matrices of the m subsections,
- and $\boldsymbol{D}_1, \cdots, \boldsymbol{D}_m$ are selector matrices that map the port voltages and currents of the subsections to the state space of circuit ϕ.

Here again, since we are interested in simulating this system using the Harmonic Balance technique, we need to modify our system equations to allow for that. In this case, the Harmonic Balance equations of this larger circuit could be rewritten as:

$$\begin{bmatrix} \bar{\boldsymbol{G}}_\phi & \bar{\boldsymbol{D}}_1\bar{\boldsymbol{R}}_1^T & \cdots & \bar{\boldsymbol{D}}_m\bar{\boldsymbol{R}}_m^T \\ \bar{\boldsymbol{R}}_1\bar{\boldsymbol{D}}_1^T & \bar{\boldsymbol{G}}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ \bar{\boldsymbol{R}}_m\bar{\boldsymbol{D}}_m^T & 0 & 0 & \bar{\boldsymbol{G}}_m \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{X}}_\phi \\ \bar{\boldsymbol{X}}_1 \\ \vdots \\ \bar{\boldsymbol{X}}_m \end{bmatrix} + \begin{bmatrix} \bar{\boldsymbol{C}}_\phi & 0 & \cdots & 0 \\ 0 & \bar{\boldsymbol{C}}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \bar{\boldsymbol{C}}_m \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{X}}_\phi \\ \bar{\boldsymbol{X}}_1 \\ \vdots \\ \bar{\boldsymbol{X}}_m \end{bmatrix} + \begin{bmatrix} \boldsymbol{F}_\phi(\bar{\boldsymbol{X}}_\phi) \\ \boldsymbol{F}_1(\bar{\boldsymbol{X}}_1) \\ \vdots \\ \boldsymbol{F}_m(\bar{\boldsymbol{X}}_m) \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{B}}_\phi \\ \bar{\boldsymbol{B}}_1 \\ \vdots \\ \bar{\boldsymbol{B}}_m \end{bmatrix} \tag{4.4}$$

where

- $\bar{\boldsymbol{G}}_\phi$, $\bar{\boldsymbol{C}}_\phi$, $\boldsymbol{F}_\phi(\bar{\boldsymbol{X}}_\phi)$ and $\bar{\boldsymbol{B}}_\phi$ are the vectors and matrices of the linear and nonlinear elements of circuit ϕ,
- $\bar{\boldsymbol{G}}_1, \cdots, \bar{\boldsymbol{G}}_m$, $\bar{\boldsymbol{C}}_1, \cdots, \bar{\boldsymbol{C}}_m$, $\boldsymbol{F}_1(\bar{\boldsymbol{X}}_1), \cdots, \boldsymbol{F}_m(\bar{\boldsymbol{X}}_m)$, $\bar{\boldsymbol{B}}_1, \cdots, \bar{\boldsymbol{B}}_m$ and $\bar{\boldsymbol{R}}_1, \cdots, \bar{\boldsymbol{R}}_m$ are the vectors and matrices of the m subsections,
- and $\bar{\boldsymbol{D}}_1, \cdots, \bar{\boldsymbol{D}}_m$ are selector matrices that map the port voltages and currents of the subsections to the state space of circuit ϕ.

We are now able to represent the circuit we are interested in reducing in macromodel format and to add it using this format to a larger system. The macromodel format we are using was also expanded so that it can handle Harmonic Balance simulations since they are typically used for RF circuits. At this point, we are ready to proceed to the next step, which is the reduction technique itself.

4.2 Reduction Technique

The reduction technique proposed in this thesis can be split into two separate steps. Each step is, in fact, a full reduction technique in its own right and can be used to greatly reduce simulation times. Our goal here is to use both techniques together and get the advantages of each one.

The first step of our method is to use proper orthogonal decomposition. This step is aimed at the linear part of the circuit and therefore greatly reduces the size of the MNA matrices. However, despite this reduction, simulation speedup remains limited. Radio frequency circuits are nonlinear, and therefore their MNA equations contain a nonlinear component. This nonlinear part is usually solved using the Newton-Raphson method which requires calculating the nonlinear vector and its derivative in every iteration. Unfortunately, while POD is capable of reducing the size of the linear parts of the equation, it cannot tackle the nonlinear part. In fact, the only thing we can do to evaluate the nonlinear part and its derivative is to revert back to the original non-reduced system and calculate it. This quickly becomes the bottleneck in simulation and greatly limits any speedup achieved by using this method.

One way to overcome this is to replace the nonlinear part of the circuit with something else that can emulate its behaviour, but has a smaller size. This is the second step of the proposed reduction technique, where artificial neural networks are used to mimic the behaviour of the reduced-order form of the nonlinear vector without the need to resort to evaluating the original one.

In the following two sections, we present both steps of reduction: proper orthogonal decomposition and artificial neural networks. Effort is made to explain the specific choices taken throughout the process. Specific emphasis is given to the choices of the size of the congruence transformation matrix, the number of hidden neurons, the structure of the neural

networks, and the training of the neural networks.

4.2.1 Proper Orthogonal Decomposition

Proper orthogonal decomposition was initially proposed for linear systems and then later extended to only time-domain nonlinear systems. The method allows us to generate a reduced-order macromodel of a specific nonlinear circuit by projecting it onto a reduced orthogonal subspace $\boldsymbol{Q}$, also known as the congruence transformation matrix. It is precisely this projection that reduces the size of the MNA equations. For the first time, this method will be applied to the frequency domain. Specifically, we will extend this method such that it can be applied to the harmonic balance simulation. This is only possible because we rewrote the macromodel formulation in a way that is suitable for the Harmonic Balance technique.

Generating the congruence transformation matrix on which the subsection will be projected is the heart of this method. The next section will be devoted for that, however, for now we are going to assume that we already have this matrix and that it is indeed a reduced orthogonal subspace. This is to give the reader an appreciation of how effective the method is at reducing the size of the equations before getting into the details of creating the subspace.

Consider again Equation 4.2 that describes a circuit macromodel ready for simulation using the Harmonic Balance technique. Performing a change of variables, $\hat{\boldsymbol{X}} = \boldsymbol{Q}\bar{\boldsymbol{X}}$ and then pre-multiplying by the congruence transformation matrix $\boldsymbol{Q}$, we can rewrite it as follows:

$$\begin{aligned} \hat{\boldsymbol{G}}\hat{\boldsymbol{X}} + \hat{\boldsymbol{C}}\hat{\boldsymbol{X}} + \hat{\boldsymbol{F}}(\hat{\boldsymbol{X}}) &= \hat{\boldsymbol{R}}\bar{\boldsymbol{U}} + \hat{\boldsymbol{B}} \\ \bar{\boldsymbol{I}} &= \hat{\boldsymbol{R}}^T\hat{\boldsymbol{X}} \end{aligned} \tag{4.5}$$

where

$$\hat{\boldsymbol{G}} = \boldsymbol{Q}^T \bar{\boldsymbol{G}} \boldsymbol{Q} \tag{4.6a}$$

$$\hat{\boldsymbol{C}} = \boldsymbol{Q}^T \bar{\boldsymbol{C}} \boldsymbol{Q} \tag{4.6b}$$

$$\hat{\boldsymbol{R}} = \boldsymbol{Q}^T \bar{\boldsymbol{R}} \tag{4.6c}$$

$$\hat{\boldsymbol{B}} = \boldsymbol{Q}^T \bar{\boldsymbol{B}} \tag{4.6d}$$

$$\hat{\boldsymbol{F}}(\hat{\boldsymbol{X}}) = \boldsymbol{Q}^T \boldsymbol{F}(\bar{\boldsymbol{X}}) \tag{4.6e}$$

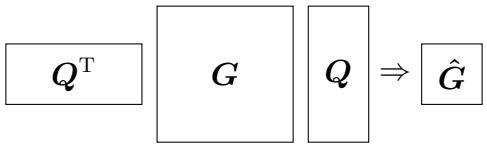

Fig. 4.2 A visual representation of POD

While Equation 4.5 looks similar to Equation 4.2, they have very different sizes. If the size of the congruence transformation matrix, q, is chosen carefully such that $q \ll N_{hb}$, then the reduced-order macromodel will be significantly smaller than the original. Figure 4.2 shows a visual representation of how this reduction occurs. It is worth noting here that while the sizes of the $\bar{\boldsymbol{G}}$, $\bar{\boldsymbol{C}}$, $\bar{\boldsymbol{R}}$, and $\bar{\boldsymbol{G}}$ have truly been reduced and replaced completely by $\hat{\boldsymbol{G}}$, $\hat{\boldsymbol{C}}$, $\hat{\boldsymbol{R}}$, and $\hat{\boldsymbol{G}}$, the same could not be said about the nonlinear vector $\boldsymbol{F}$. Anytime we run a simulation and we need to evaluate this vector, we will need to go back to the original system to evaluate the nonlinear vector and then return to the reduced one. This is a significant limitation of this method.

Congruence Transformation Matrix

As mentioned before, the heart of this method is the finding the congruence transformation matrix. In this section will address this issue. First, we need to realize that the newly formed reduced-order macromodels created using this technique will only be valid under specific conditions. These conditions should be chosen at the time the macromodel is created and generally depend on how the user intends to use the macromodel later. Generally speaking those conditions will include input power, input frequency, and loading conditions. To ensure that the macromodel is valid over those specific conditions, they need to be accounted for while constructing the reduced subspace $\boldsymbol{Q}$. For that to happen, the original macromodel should be simulated over the whole range of conditions desired by the user, and the solution of the system is then stored in a matrix $\boldsymbol{K}$ as follows:

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 & \cdots & \boldsymbol{X}_s \end{bmatrix} \tag{4.7}$$

Given the nature of electronic circuits, there will be many similarities in the responses above. This is only natural since we are simulating over a specific range of conditions and not randomly. Hence, the above subspace will surely contain many redundant directions. In order to capture the most dominant directions, singular value decomposition is used:

$$\boldsymbol{K} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{4.8}$$

The result of SVD is three matrices. The first is an orthogonal matrix, the second is the singular values in descending order, and the third is also an orthogonal matrix. The congruence transformation matrix $\boldsymbol{Q}$ is then constructed from the first few columns of the matrix $\boldsymbol{U}$ ensuring it contains the most dominant directions of the subspace. Figure 4.3

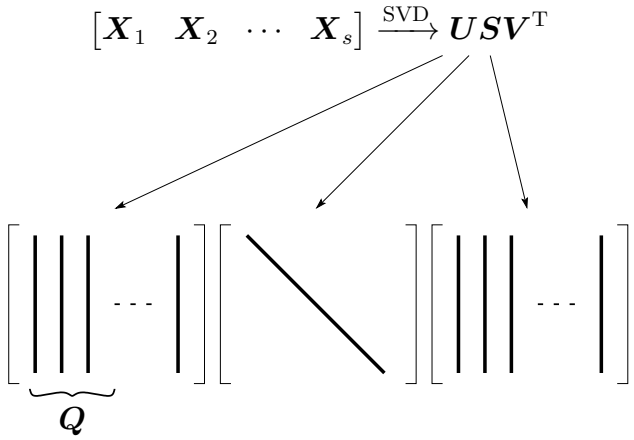

Fig. 4.3 Visual representation of the SVD process

shows a visual representation of this process.

The choice of the number of columns, and therefore the size of the reduced macromodel, is a critical one. Too small and the reduced-order macromodel might not be accurate enough. Too big and we risk not having any reduction or speedup. For linear circuits, researchers have come up with systematic methods to make this choice. Unfortunately, this does not exist for nonlinear circuits. In fact, trial and error must be used to determine what is sufficiently accurate. In the numerical results section, the user will be presented with more details on this since it varies from one circuit to another.

With $\boldsymbol{Q}$ formed, we can now perform proper orthogonal decomposition, as discussed in the previous section, and obtain the reduced-order macromodel. This macromodel can now be readily added to a larger system containing other macromodels (reduced or not reduced) and other linear and nonlinear elements:

$$\begin{bmatrix} \boldsymbol{G}_\phi & \hat{\boldsymbol{D}}_1\hat{\boldsymbol{R}}_1^T & \cdots & \hat{\boldsymbol{D}}_m\hat{\boldsymbol{R}}_m^T \\ \hat{\boldsymbol{R}}_1\hat{\boldsymbol{D}}_1^T & \hat{\boldsymbol{G}}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ \hat{\boldsymbol{R}}_m\hat{\boldsymbol{D}}_m^T & 0 & 0 & \hat{\boldsymbol{G}}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_\phi \\ \hat{\boldsymbol{X}}_1 \\ \vdots \\ \hat{\boldsymbol{X}}_m \end{bmatrix} + \begin{bmatrix} \boldsymbol{C}_\phi & 0 & \cdots & 0 \\ 0 & \hat{\boldsymbol{C}}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \hat{\boldsymbol{C}}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_\phi \\ \hat{\boldsymbol{X}}_1 \\ \vdots \\ \hat{\boldsymbol{X}}_m \end{bmatrix} + \begin{bmatrix} \boldsymbol{F}_\phi(\boldsymbol{X}_\phi) \\ \hat{\boldsymbol{F}}_1(\hat{\boldsymbol{X}}_1) \\ \vdots \\ \hat{\boldsymbol{F}}_m(\hat{\boldsymbol{X}}_m) \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}_\phi \\ \hat{\boldsymbol{B}}_1 \\ \vdots \\ \hat{\boldsymbol{B}}_m \end{bmatrix} \tag{4.9}$$

While the size of the above system is definitely much smaller when reduced-order macromodels are used, the challenge is how to deal with the nonlinear vector. Although the size of $\hat{\boldsymbol{F}}$ is small, we do not have a straightforward way of evaluating it without reverting to the original system that contains $\boldsymbol{F}$. Therefore at every iteration where the reduced nonlinear vector is needed, we must first return to the original system by multiplying $\hat{\boldsymbol{X}}$ with the congruence transformation matrix $\boldsymbol{Q}$, evaluate the original nonlinear vector $\boldsymbol{F}$, and then multiply by $\boldsymbol{Q}^T$ to get $\hat{\boldsymbol{F}}$. This also needs to be repeated for every reduced-order macromodel in the system:

$$\begin{bmatrix} \boldsymbol{F}_\phi(\boldsymbol{X}_\phi) \\ \hat{\boldsymbol{F}}_1(\hat{\boldsymbol{X}}_1) \\ \vdots \\ \hat{\boldsymbol{F}}_m(\hat{\boldsymbol{X}}_m) \end{bmatrix} = \begin{bmatrix} \boldsymbol{F}_\phi(\boldsymbol{X}_\phi) \\ \boldsymbol{Q}_1^T\boldsymbol{F}_1(\boldsymbol{Q}_1\boldsymbol{X}_1) \\ \vdots \\ \boldsymbol{Q}_m^T\boldsymbol{F}_m(\boldsymbol{Q}_m\boldsymbol{X}_m) \end{bmatrix} \tag{4.10}$$

This obviously is a major limitation for speedup given the cost of evaluating the original nonlinear vector due to its large size. Unfortunately, it does not stop there. While performing

the Newton-Raphson method in the Harmonic Balance technique, we need the derivative of the objective function and hence the derivative of the nonlinear vector. This is true for all the reduced-order macromodels present in the system. The general structure of the derivative of the nonlinear vector would be:

$$\begin{bmatrix} \frac{\partial \boldsymbol{F}_\phi(\boldsymbol{X}_\phi)}{\partial \boldsymbol{X}_\phi} & 0 & \cdots & 0 \\ 0 & \frac{\partial \hat{\boldsymbol{F}}_1(\hat{\boldsymbol{X}}_1)}{\partial \hat{\boldsymbol{X}}_1} & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \frac{\partial \hat{\boldsymbol{F}}_m(\hat{\boldsymbol{X}}_m)}{\partial \hat{\boldsymbol{X}}_m} \end{bmatrix}$$

We now are facing the same problem again: we do not have a way to evaluate the reduced-order nonlinear derivative without reverting to the original system. And so everytime we need this derivative, we must multiply $\hat{\boldsymbol{X}}$ by $\boldsymbol{Q}$ to get $\boldsymbol{X}$, evaluate the derivative $\frac{\partial \hat{\boldsymbol{F}}(\hat{\boldsymbol{X}})}{\partial \hat{\boldsymbol{X}}}$, and lastly multiply and pre-multiply by $\boldsymbol{Q}$ and $\boldsymbol{Q}^T$:

$$\frac{\partial \hat{\boldsymbol{F}}_m(\hat{\boldsymbol{X}}_m)}{\partial \hat{\boldsymbol{X}}_m} = \boldsymbol{Q}^T \frac{\partial \boldsymbol{F}_m(\boldsymbol{X}_m)}{\partial \boldsymbol{X}_m} \boldsymbol{Q} \tag{4.11}$$

Because of the above two required evaluations, it can easily be seen that while the newly formed system that contains the reduced-order macromodels is much smaller than the original one that contains the full-size macromodels, simulation speedup will remain limited. During every Newton-Raphson iteration, we need to revert back to the original system to evaluate the nonlinear vector and its derivative. This in fact becomes the bottleneck of the whole simulation and hinders any improvements in speedup.

Nevertheless, the first step of reduction achieved something significant: it reduced the size of the linear part of the MNA matrices. All that remains is figuring out how to reduce the nonlinear vector and its derivative. If we could figure out a way to avoid ever returning

to the original system to avoid the nonlinearities, it would remove this great limitation on speedup and opens up more reduction possibilities. This paves the way for the next step in reduction: artificial neural networks in the specific form of feedforward neural networks.

4.2.2 Feedforward Neural Networks

As discussed in the previous section, to overcome the speedup limitation, we would need to come up with a way to evaluate the nonlinear vector and its derivative without resorting to the need to revert back to the original system. Ideally, the reduced nonlinear vector would be replaced with something that would have similar behaviour without the need of the original vector or its derivative. More importantly, whatever the replacement is, it should also be simple enough such that it does not add too much overhead or require too much computation that might take away any possible speedup improvements.

Our proposed solution to this problem, which is the second step of our reduction technique, is artificial neural networks which can do exactly that. Artificial neural networks, specifically the feedforward ones, are known to have remarkable curve fitting abilities and when constructed carefully, they can mimic any nonlinear function to any desired accuracy, as discussed in the early introductory chapters. We intend to use this structure of the neural network to replace the reduced-order nonlinear vector. Furthermore, the derivative of the function that represents this neural network will serve as the new derivative of the nonlinear vector.

Since a three-layer neural network, shown in Figure 4.4, is a proven universal approximator and is capable of emulating any function to a desired accuracy, we will go ahead and use that here to replace the reduced-order nonlinear vector $\hat{\boldsymbol{F}}$. The advantage here is that if we are successful, then during the Newton-Raphson iterations, we will not need to revert to the original system to evaluate the nonlinear elements, but instead we can simply use the neural

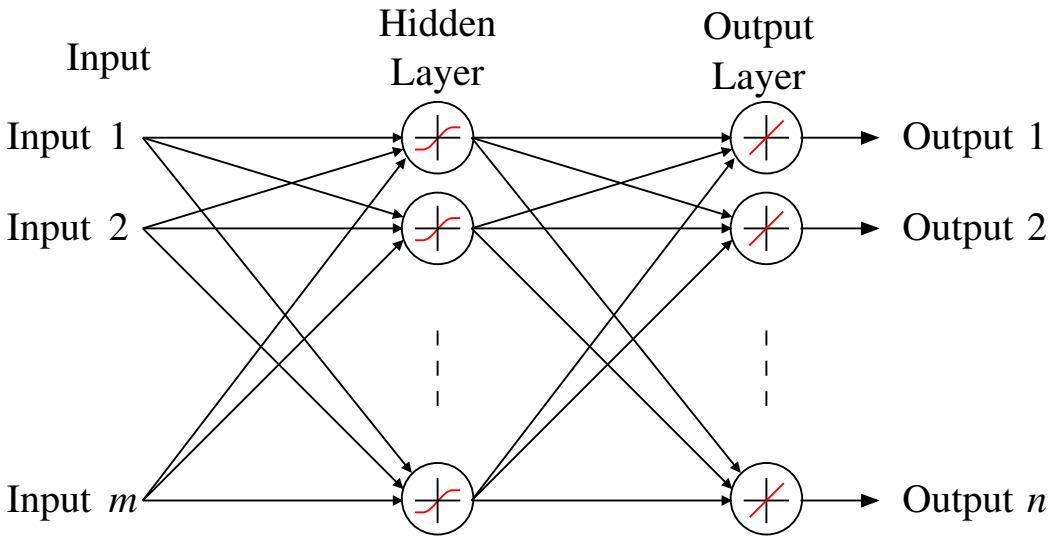

Fig. 4.4 Feedforward neural network

network to get the values that $\hat{\boldsymbol{F}}$ would have provided. If the neural network is accurate yet simple enough, then a great speedup could be achieved. It should also be noted that the derivative of the neural network function also needs to be evaluated because it too will be needed during the Newton-Raphson iterations.

Once the neural network is constructed, it can replace the nonlinear vector as such:

$$\boldsymbol{F_{nn}}(\hat{\boldsymbol{X}}) \approx \hat{\boldsymbol{F}}(\hat{\boldsymbol{X}}) \tag{4.12}$$

where $\boldsymbol{F_{nn}}$ is the function that describes the neural network. In other words, $\boldsymbol{F_{nn}}$ describes the activation functions, the weights and biases, and the layers of the neural network. The derivative of this function could also then be computed to replace the derivative of the original nonlinear vector as such:

$$\frac{\partial \boldsymbol{F_{nn}}(\hat{\boldsymbol{X}})}{\partial \hat{\boldsymbol{X}}} \approx \frac{\partial \hat{\boldsymbol{F}}(\hat{\boldsymbol{X}})}{\partial \hat{\boldsymbol{X}}} \tag{4.13}$$

It is evident that to be able to completely express $\boldsymbol{F_{nn}}$ and its derivative, we need to have all the details of the neural network, including its structure, layers, activation functions, weights, and biases.

Training of the Neural Network

Neural networks are only able to emulate the behaviour of other functions under two conditions. First, they need sufficient data for training, and second, a complex enough structure to learn this specific data and be able to predict its input-output behaviour or the input-output behaviour of similar data. Both are not trivial to find and require investigation. In most cases, they depend on the type and difficulty of the problem at hand.

On the amount of data provided, if too little data is provided then the neural network does not have enough information to learn the underlying functions and will fail to generalize or predict the input-output behaviour. The structure of the neural network is also critical. If it is too small, then again here the network will not have enough "flexibility" to learn the underlying functions. If it is too big, the user risks over-fitting, or in other words, the network "memorizes" the data. This is not advantageous because the network did not really learn the data and will fail to generalize its input-output behaviour. If the network "sees" this exact same data again, it will predict its input-output accurately, but if it "sees" something very close to it, then it will most likely fail.

In the context of neural networks, training or learning refers to the step when the weights and biases of the individual artificial neurons are adjusted such that a minimum error in the output of the network is achieved. In other words, the weights and biases are changed until the neural network gives an output which is considered close enough to the function it is trying to represent. The way those weights and biases are changed is through an algorithm known as the training algorithm. To do its job successfully, the training algorithm will

need access to input and output data. For feedforward neural networks, the most popular algorithm today is the back-propagation algorithm with all its different flavours.

For the neural network to accurately approximate the nonlinear function, it will need a large set of data to use for training. This training data should be extracted from the behaviour of the original system. The inputs in our data will be $\hat{\boldsymbol{X}}$ and the outputs will be $\hat{\boldsymbol{F}}(\hat{\boldsymbol{X}})$. This data will be taken from the same exact range where the reduced-order macromodel is considered to be valid. Therefore, we can guarantee that the neural network will be able to mimic the behaviour of the nonlinear vector within that specified range.

As for the choice of the training algorithm, since this is a feedforward neural network, the back-propagation algorithm was used. Specifically, we found that the Levenberg-Marquardt back-propagation algorithm worked best for our case and resulted in the best performance of the neural network. It converged faster, but did require more memory and CPU computations than the other back-propagation algorithms we tried.

Structure of the Neural Network

While the overall general structure of the feedforward network was described in Figure 4.4, in this section we will discuss it in a bit more detail. Specifically, we will express the function that completely describes the inside workings of our feedforward neural network. That includes the number of layers, the number of neurons, the weights and biases, and the activation functions. Figure 4.5 shows precisely that. Here the reader can see a representation of every layer of the neural network, with its weights, biases, and activation functions. In our case, since we are using a two-layer network, there will only be the input, the hidden layer, and the output layer.

In our case, $\hat{x}$ is the input, w_1 represents the weights of the neurons of the hidden layer, and b_1 represents the biases of these neurons. The activation function f_1 takes in a as an

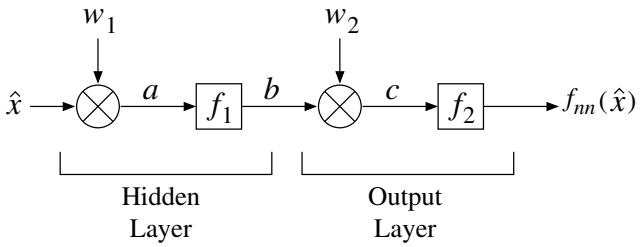

Fig. 4.5 Feedforward Neural Network

input and provides b as output, where a is the result of the weights multiplied by the inputs plus the biases. In our structure of the neural network, the activation function of the first layer is the *logsig* function:

$$f_1(x) = \frac{e^x - e^{-x}}{e^x - e^{-x}} \tag{4.14}$$

The same rationale applies to the output layer of the neural network. It has its own weights, biases, and activation function. The activation function we used for the output layer is simply the linear function:

$$f_2(x) = x \tag{4.15}$$

It can now be seen that the whole structure of the neural network could be described using one function $f_{nn}(\hat{x})$ that could be written as a series of the previous functions of the neural network:

$$f_{nn}(\hat{x}) = f_2(c) \tag{4.16}$$

and

$$c = b * W_2 + b_2 \tag{4.17}$$

where W_2 is the matrix containing the weights of the ouptput layer, b_2 is the matrix containing the biases of the output layer, and b is the output of the hidden layer:

$$b = f_1(a) \tag{4.18}$$

and

$$a = \hat{x} * W_1 + b_1 \tag{4.19}$$

where W_1 is the matrix containing the weights of the hidden layer and b_1 is the matrix containing the biases of the hidden layer.

Assuming the neural network has been trained carefully and successfully, we can now use the function that describes it, f_{nn}, to replace the reduced-order nonlinear vector in our calculations. If this function is accurate, yet simple enough, then a large speedup could be achieved.

It should be noted that in creating the above network, we did not start from scratch. We used available tools in Matlab which allowed us to create and train the above network. We then simply extracted the weights, biases, and activation functions to come up with a closed form for f_{nn}. Of course, we can not stop there. We now need to compute the derivative of this neural network to be able to use it properly in our Harmonic Balance simulations.

Derivative of the Neural Network

To be able to use the neural network during the Newton-Raphson iteration, we will need to not only evaluate the function describing it, but also the derivative of the function describing it with respect to the input $\hat{x}$:

$$\frac{\partial f_{nn}(\hat{x})}{\partial \hat{x}} \tag{4.20}$$

To be able to do that, we need to use the derivative chain rule and take into account every function inside the neural network. That includes the multiplication of the weights, the addition of the biases, and the activation functions. We also need to include any scaling functions used.

The chain derivative is:

$$\frac{\partial f_{nn}(\hat{x})}{\partial \hat{x}} = \frac{\partial f_{nn}}{\partial c} \times \frac{\partial c}{\partial b} \times \frac{\partial b}{\partial a} \times \frac{\partial a}{\partial \hat{x}} \tag{4.21}$$

where,

$$\frac{\partial f_{nn}}{\partial c} = \frac{\partial f_2(c)}{\partial c} = \frac{c}{\partial c} = 1 \tag{4.22}$$

$$\frac{\partial c}{\partial b} = \frac{\partial b * W_2 + b_2}{\partial b} = W_2 \tag{4.23}$$

$$\frac{\partial b}{\partial a} = \frac{\partial f_1(a)}{\partial a} = 1 - f^2(a) \tag{4.24}$$

and

$$\frac{\partial a}{\partial \hat{x}} = \frac{\partial \hat{x} * W_1 + b_1}{\partial \hat{x}} = W_1 \tag{4.25}$$

Lastly, the neural network uses a scaling function that maps its inputs and outputs to a specific range. This range is used in order to assist the training algorithm to converge faster. This scaling function becomes part of the neural network itself and has to be taken into account when creating the closed form function f_{nn} that describes its structure.

Using the above, we now have a closed form that describes the derivative of the neural network and can be used in the calculation of the nonlinear Jacobian matrix of the Harmonic Balance technique.

Calculating the Output of the Neural Network and its Derivative

One major advantage of the neural networks is not only that they simplify the evaluation of the nonlinear elements, but also the their derivatives requires minimal computation. Consider the *tansig* function that is used as the activation function of the hidden layer:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.26}$$

The derivative of this function can be rewritten as:

$$\frac{\partial f(x)}{\partial x} = 1 - f^2(x) \tag{4.27}$$

What this means is that we do not need to reevaluate any new functions to compute the derivative. We have already computed $f(x)$ when we evaluated the function of the neural network, and to compute its derivative we simply need to square it and subtract it from 1.

The same is true for the linear activation function:

$$f(x) = x \tag{4.28}$$

and its derivative will simply be:

$$\frac{\partial f(x)}{\partial x} = 1 \tag{4.29}$$

Furthermore, if we also consider the effects of the weights and biases, then at any input to any layer of the network, we can write:

$$\text{Output} = \text{Input} \times \text{Weights} + \text{Biases} \tag{4.30}$$

Once again the derivative is simple to calculate for every layer:

$$\frac{\partial \text{ Output}}{\partial \text{ Input}} = \text{Weights} \tag{4.31}$$

This is a quantity that is readily available and needs no extra computations to acquire. This is a major advantage for using neural networks. While in the past, the derivative of the nonlinear vector was not only complicated, but also had its own functions, now the function itself and its derivative are very closely related and require little extra computation. As will be clear from numerical results, this provides more speedup than expected.

4.3 Numerical Results

In this section, we will show the numerical results of our proposed method applied to various radio frequency circuits. For every example, we will address how we we decided on the size of $\boldsymbol{Q}$, the number of hidden neurons, and the size of training data.

4.3.1 Low-Noise Amplifier

The first example is the low noise amplifier shown in Figure 4.6. The macromodel has two ports: one at the input and one at the load. The goal is to generate a reduced-order macromodel of this system that is valid over a specific range of input power and loading conditions. The size of the original Harmonic Balance equations is 4862.

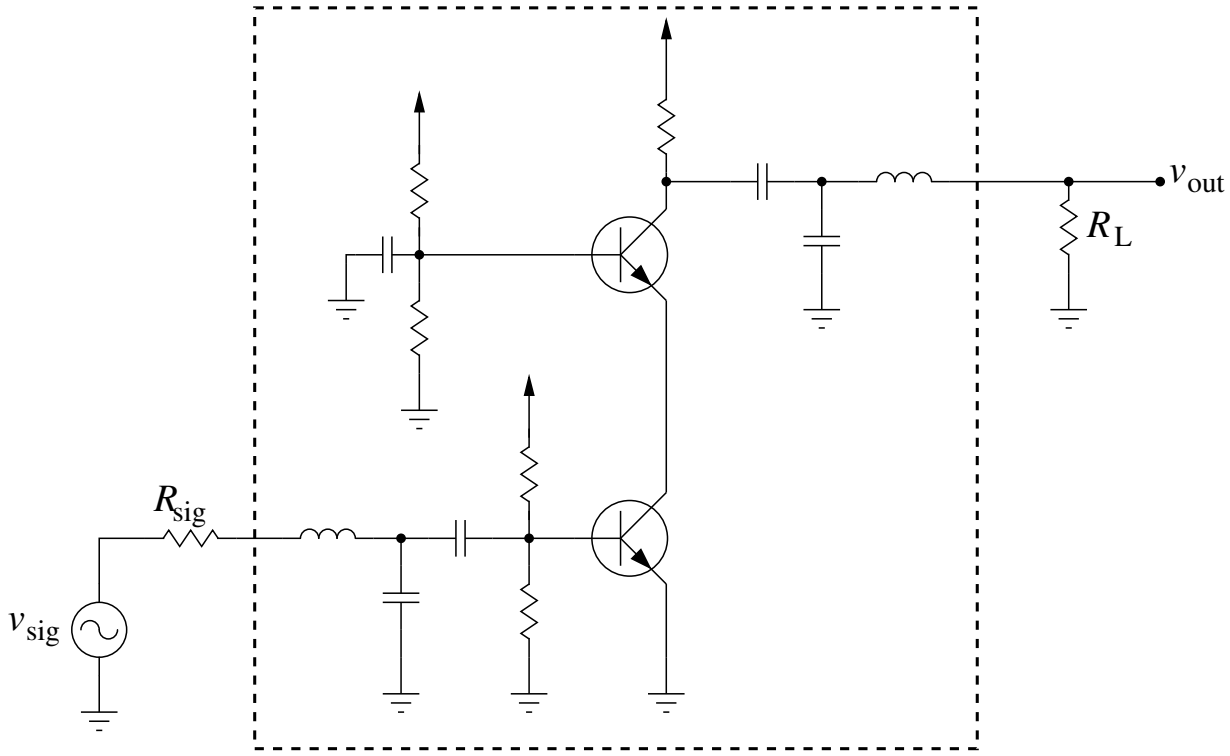

Fig. 4.6 Cascode amplifier

Congruence Transformation Matrix

The original system was simulated over the desired range. The input voltage was varied between 10µV and 1mV in steps of 10µV. The load was varied between 45Ω and 55Ω in steps of 1Ω. The total number of simulations carried out was 1100 and used to create the

subspace:

$$\boldsymbol{K} = [X_1 \; X_2 \; X_3 \ldots X_{1100}] \tag{4.32}$$

Singular value decomposition was then used to capture the most dominant directions of this subspace and the congruence transformation matrix was constructed. For this example, a $\boldsymbol{Q}$ size of 5 was sufficient to accurately capture the behaviour of the system.

Feedforward Neural Networks

To train the neural network accurately, more data points are needed than the ones used to generate the reduced subspace. To achieve this, the same points used to generate the subspace were used again in addition to points taken between them. The size of the data used to train the neural network was close to 10 thousand points.

As for the architecture of the neural network, 20 hidden neurons were used to ensure the network has enough "flexibility" to learn the data. Lastly, before training was done, two mapping functions were used to scale the data and make it more normalized in order to make the training easier.

Simulation

The reduced-order macromodel was simulated inside the valid range and compared to the original models. Figures 4.7, 4.8, and 4.9 show a comparison when the load was changed. Figures 4.10, 4.11, and 4.12 show a comparison when the input power was changed. As can be observed the reduced-order macromodels matched the original to a high degree of accuracy.

Overall, the speedup gain achieved by using the newly generated macromodels ranged

between 4 to 5 times.

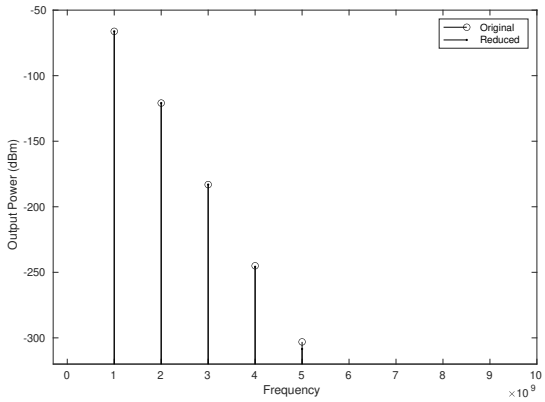

Fig. 4.7 A comparison between the reduced and original system when the load is 45Ω.

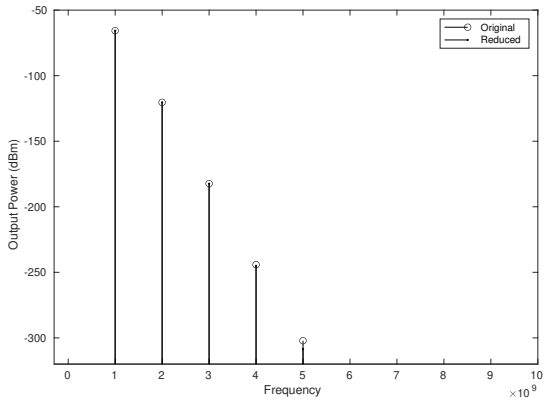

Fig. 4.8 A comparison between the reduced and original system when the load is 50Ω.

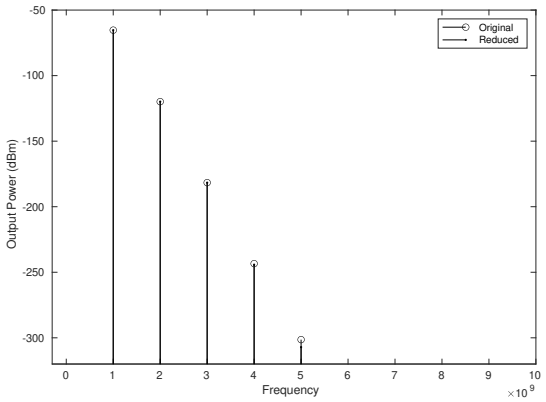

Fig. 4.9 A comparison between the reduced and original system when the load is 55Ω.

4.3.2 Frequency Mixer

In this example, we use the proposed algorithm to create a reduced-order macromodel of the RF mixer shown in Figure 4.13. Unlike the previous example, the mixer subcircuit has three

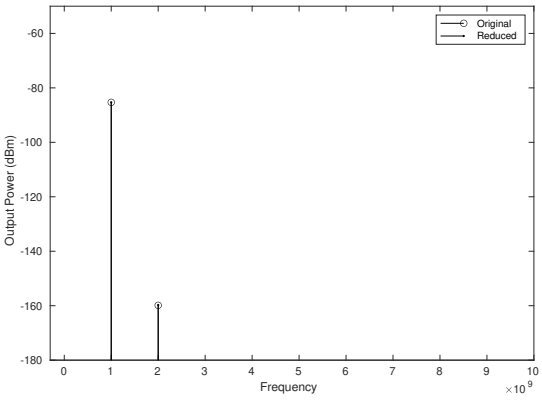

Fig. 4.10 A comparison between the reduced and original system when the input voltage 50µV.

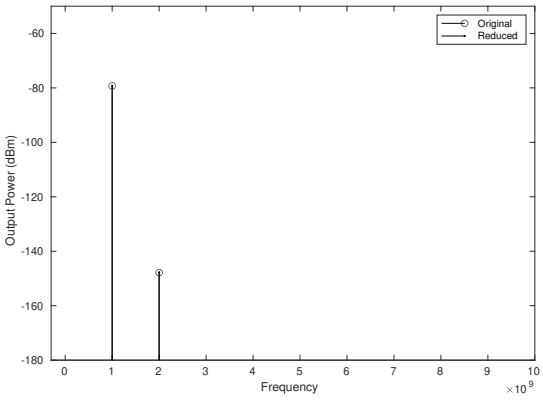

Fig. 4.11 A comparison between the reduced and original system when the input voltage 100µV.

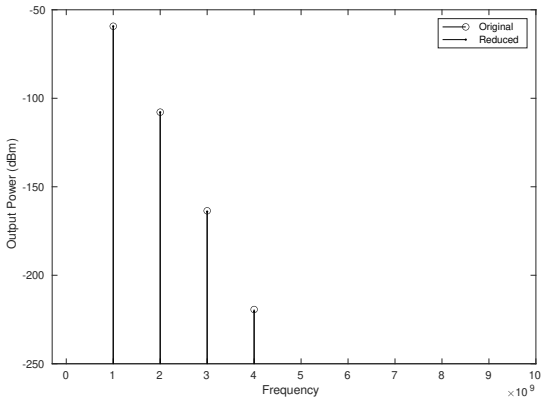

Fig. 4.12 A comparison between the reduced and original system when the input voltage 1mV.

ports: one port for the local oscillator v_{LO}, another port for the RF input v_{RF}, and lastly one port for the load v_{out}.

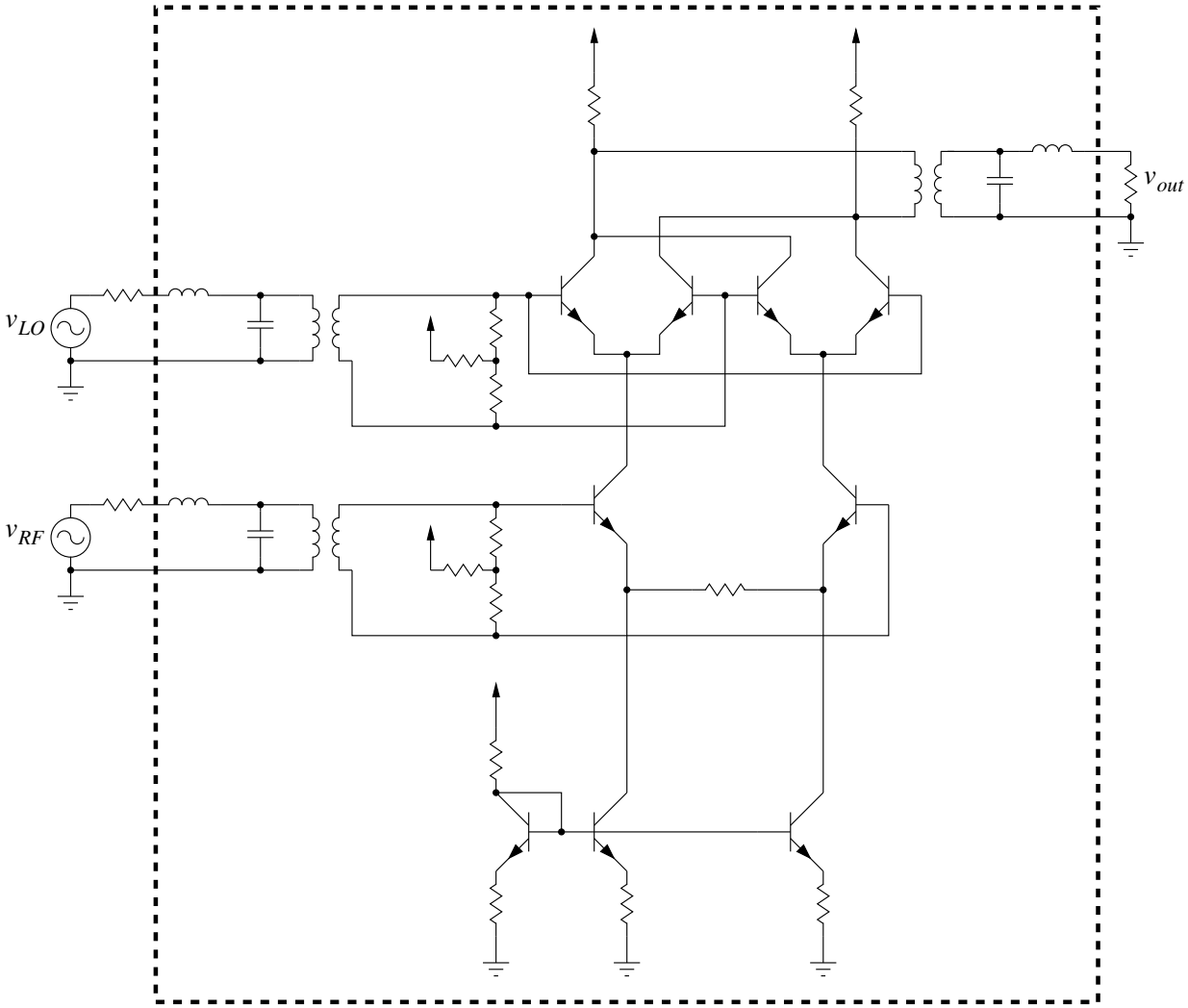

Fig. 4.13 Frequency mixer

The circuit is designed for a local oscillator frequency of 1 GHz and an RF frequency of 100 MHz. In this example, we create a macromodel of this mixer that is valid over a range of RF input power. Specifically, it is valid for an RF input of 10 mV to 100 mV.

Congruence Transformation Matrix

To construct the congruence transformation matrix for the macromodel, we need to simulate the original circuit over the desired range of which we would like it be valid. Therefore, the mixer was simulated with an RF input power varying from 10 mV to 100mV in steps of 1 mV, for a total of 91 simulations. The solution of every simulation is stored and the matrix containing the subspace would be:

$$\boldsymbol{K} = [X_1 \; X_2 \; X_3 \ldots X_{99}] \tag{4.33}$$

To capture the most dominant directions of this subspace, SVD is used and the congruence transformation matrix is constructed from the most dominant of those directions. For this example, a $\boldsymbol{Q}$ size of 5 was enough to capture accurately the behaviour of the mixer. Using this size of $\boldsymbol{Q}$, the circuit can be reduced from its original size of 21,097 to only 5.

Feedforward Neural Networks

Unlike the construction of $\boldsymbol{Q}$, where only a handful of simulations are required, the training of the neural network requires a significantly larger data set to accurately emulate the behavior of the nonlinear function. To accomplish this, a much larger number of points within the desired region of validity is taken, specifically 4,411 (as opposed to only the 91 simulation points needed to construct $\boldsymbol{Q}$).

The ANN has five inputs and five outputs. To ensure a high degree of accuracy, the ANN was constructed with 20 hidden neurons. The Levenberg-Marquardt backpropagation algorithm was used to train this network.

Simulation

The figures below show the a comparison between the reduced system constructed using the proposed method and the original method. The speedup achieved in all simulations ranged between 7 to 8 times.

Figure 4.14 shows a comparison between the harmonics of the original and reduced system when the input of the RF source was low, while Figure 4.15 shows a comparison between the harmonics of the original and reduced system when the input of the RF source is high. In both cases, the reduced-order macromodel constructed using the proposed method matches the original system to a high degree of accuracy while achieving a large speedup in simulation.

For a more comprehensive picture, Figure 4.16 shows a comparison between the harmonics of the original system and the reduced system for three harmonics over the whole range of predefined inputs. Figure 4.17 shows the output voltage in the time domain. Both figures illustrate that the reduced system matches the original one fairly well while still achieving speedup.

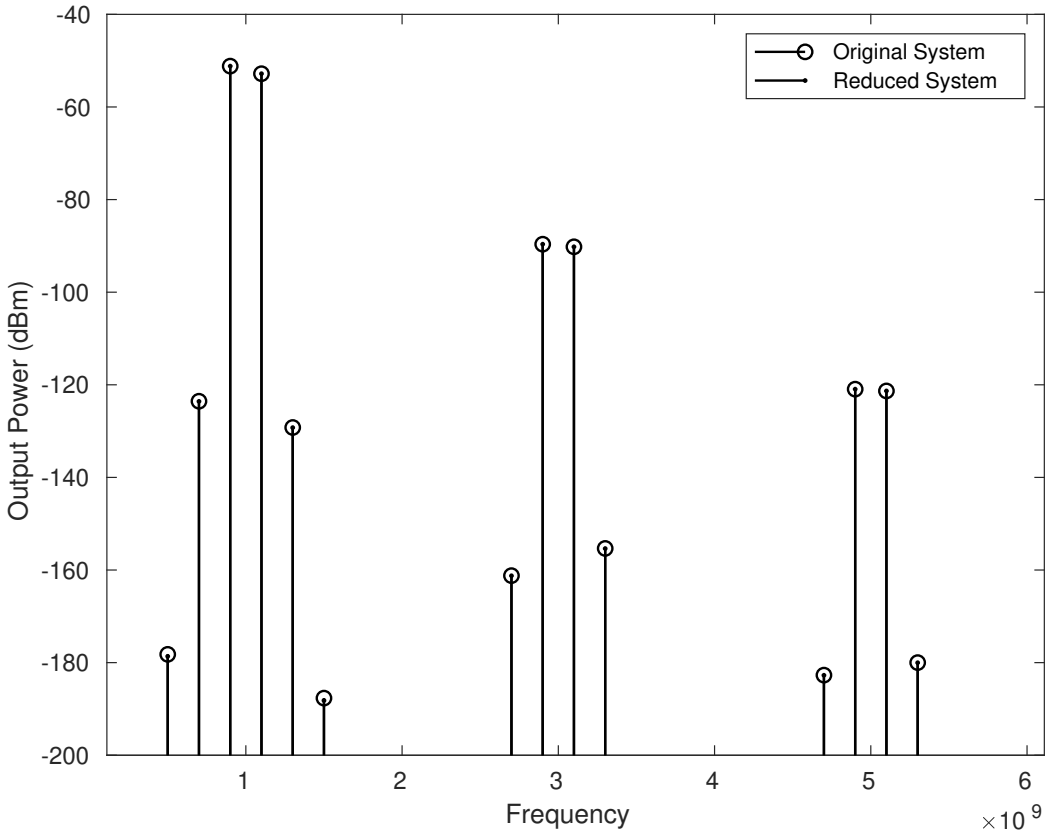

Fig. 4.14 A comparison between the original and reduced system at low RF input power.

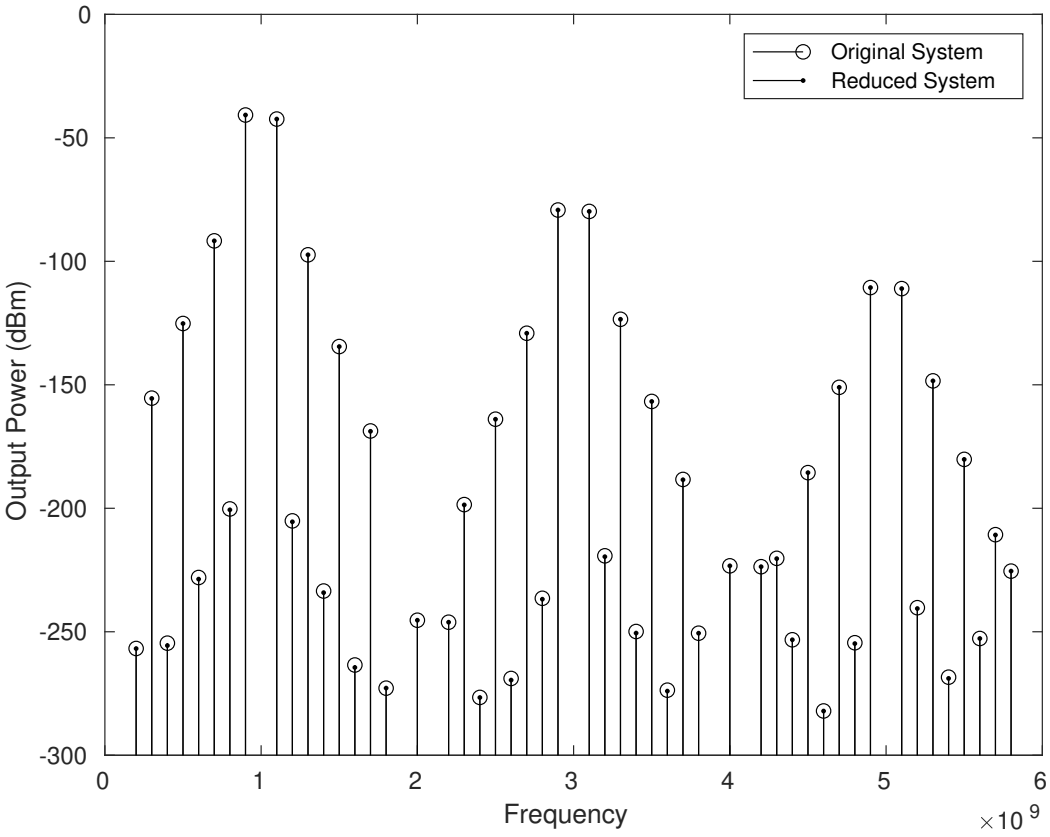

Fig. 4.15 A comparison between the original and reduced system at high RF input power.

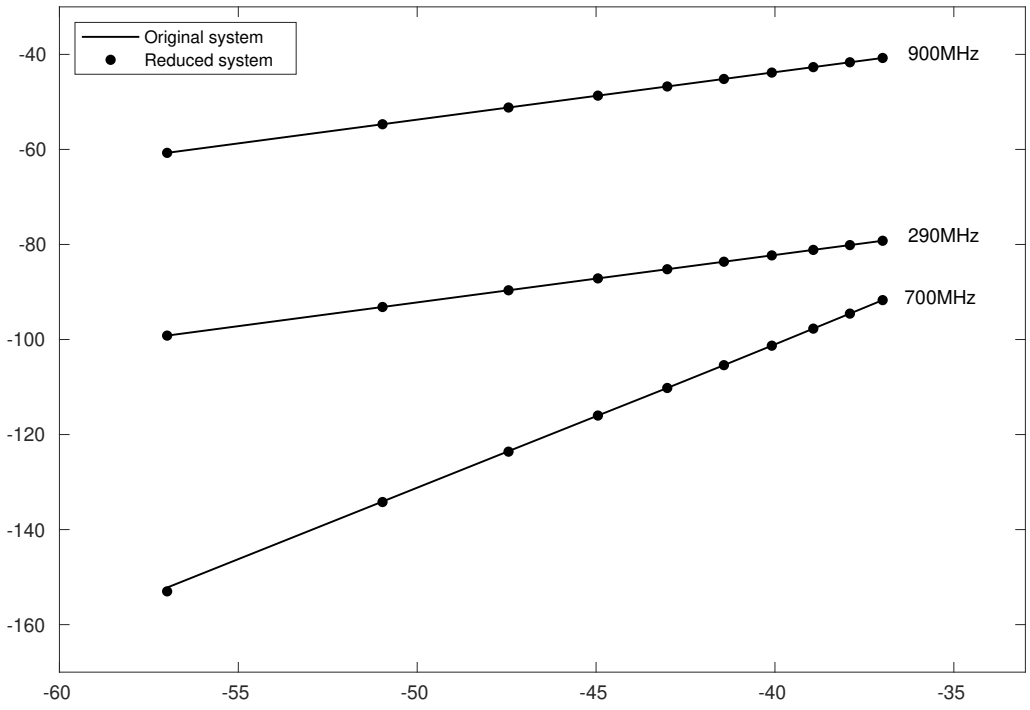

Fig. 4.16 A comparison between the original and reduced system for various harmonics at different input voltages.

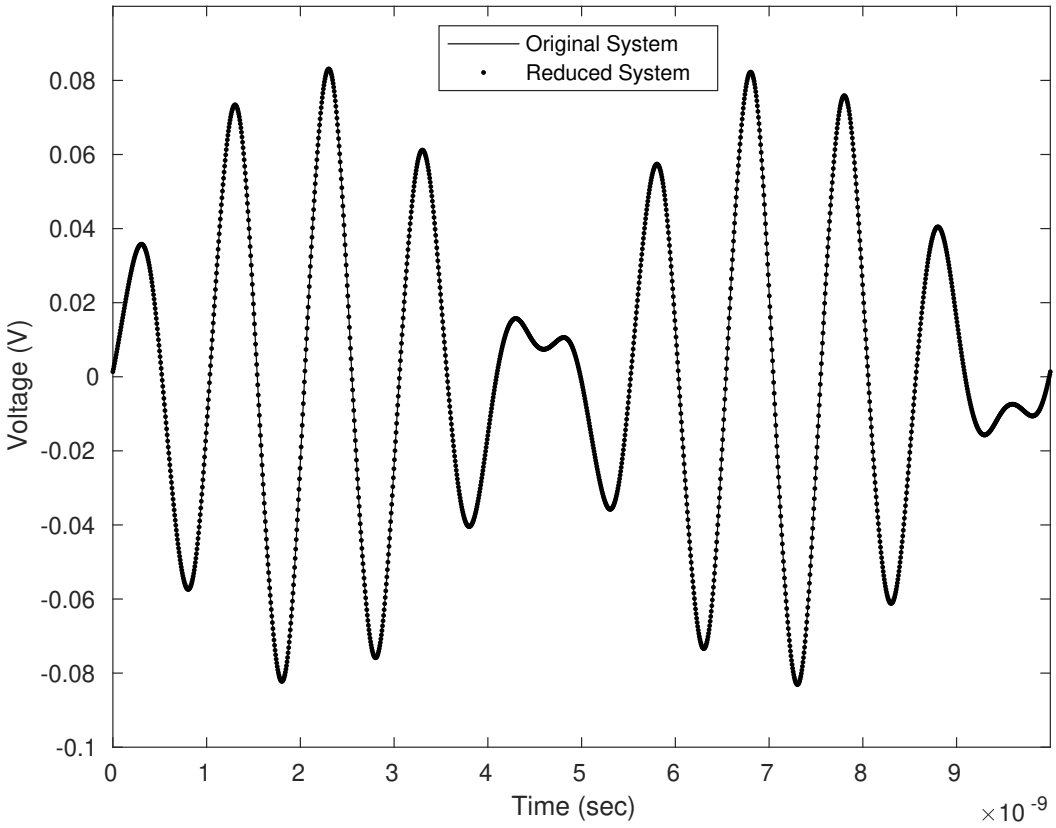

Fig. 4.17 A comparison between the original and reduced system in the time domain.

Chapter 5

Macromodeling Nonlinear Circuits in the Time Domain

In this chapter, we turn our focus to creating reduced-ordered macromodels of nonlinear circuits suitable for simulation in the time domain. The reduction technique presented in this chapter is similar to the one used for radio frequency circuits, however, it is adjusted to work for circuits to be simulated in the time domain.

The main difference between the two techniques is in the architecture of the neural network. While one feedforward neural network was used to represent the nonlinear vector of the Harmonic Balance equations, here we will be using one feedforward neural network for each entry of this vector. The reason is that in the Harmonic Balance equations, all entries in the nonlinear vector had a similar behaviour and range. That is because these RF circuits are usually operated around a point where they behave almost linearly. On the other hand, the nonlinear circuits that we will encounter in this chapter exhibit very strong nonlinearities. This means that the entries in the nonlinear vector of their equations have a much wider dynamic range, which makes it extremely challenging to represent the whole

vector with only one neural network. Our solution was to simply represent each entry with one neural network.

5.1 Time Domain Equations of a Nonlinear Subsection

Before we could tackle the problem of creating the reduced-order models of the nonlinear circuits, we have to reformulate the MNA equations into macromodel. This is similar to Section 4.2, but without the need for having the Harmonic Balance equations.

5.1.1 Network Formulation

Consider the multi-port circuit subsection, which contains several linear and nonlinear elements shown in Figure 5.1.

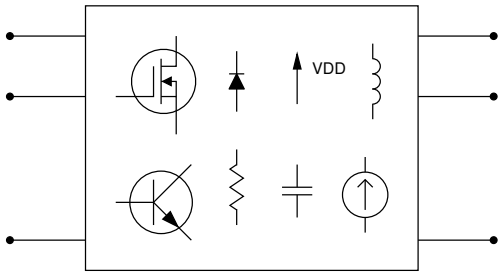

Fig. 5.1 A multi-port circuit subsection.

The MNA formulation for this subsection in the time domain can be written as:

$$\begin{aligned} \boldsymbol{G}\boldsymbol{x}(t) + \boldsymbol{C}\dot{\boldsymbol{x}}(t) + \boldsymbol{f}(\boldsymbol{x}(t)) &= \boldsymbol{R}\boldsymbol{u}(t) + \boldsymbol{b}(t) \\ \boldsymbol{i}(t) &= \boldsymbol{R}^T\boldsymbol{x}(t) \end{aligned} \tag{5.1}$$

where

- $\boldsymbol{u}(t) \in \mathbb{R}^{p\times 1}$ contains the ports' voltages,
- $\boldsymbol{i}(t) \in \mathbb{R}^{p\times 1}$ contains the ports' currents,
- $\boldsymbol{R} \in \mathbb{R}^{n\times p}$ is a selector matrix that maps the ports' voltages and currents into the state space of the circuit,
- p is the number of ports,
- and n is the size of the MNA equations.

The above equations, which can completely describe a nonlinear subsection, will be reduced to created the reduced-order macromodel. However, to be able to simulate the macromodel as part of a larger circuit, we need to also address how those equations could be added to a bigger system.

5.1.2 System Formulation

A nonlinear subsection such as the one defined in the previous section can be readily added to the overall equations of a circuit. Consider a circuit ϕ, similar to the one in Figure 5.2, containing several linear and nonlinear elements in addition to m subsections.

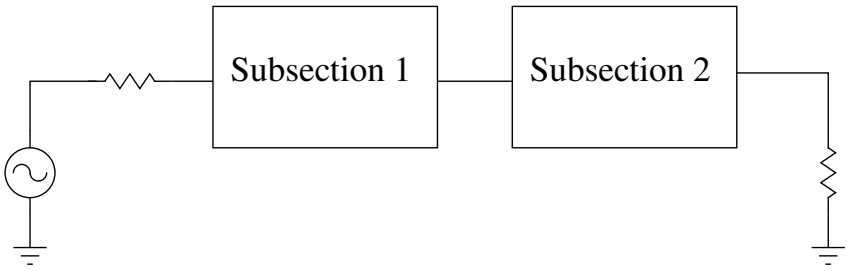

Fig. 5.2 A circuit that contains elements and subsections.

The contributions of the m subsections can be added to the overall equations of circuit ϕ as follows:

$$\begin{bmatrix} \boldsymbol{G}_\phi & \boldsymbol{D}_1\boldsymbol{R}_1^T & \cdots & \boldsymbol{D}_m\boldsymbol{R}_m^T \\ \boldsymbol{R}_1\boldsymbol{D}_1^T & \boldsymbol{G}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ \boldsymbol{R}_m\boldsymbol{D}_m^T & 0 & 0 & \boldsymbol{G}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_\phi(t) \\ \boldsymbol{x}_1(t) \\ \vdots \\ \boldsymbol{x}_m(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{C}_\phi & 0 & \cdots & 0 \\ 0 & \boldsymbol{C}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \boldsymbol{C}_m \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_\phi(t) \\ \dot{\boldsymbol{x}}_1(t) \\ \vdots \\ \dot{\boldsymbol{x}}_m(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{f}_\phi(\boldsymbol{x}_\phi(t)) \\ \boldsymbol{f}_1(\boldsymbol{x}_1(t)) \\ \vdots \\ \boldsymbol{f}_m(\boldsymbol{x}_m(t)) \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_\phi(t) \\ \boldsymbol{b}_1(t) \\ \vdots \\ \boldsymbol{b}_m(t) \end{bmatrix} \tag{5.2}$$

where

- $\boldsymbol{G}_\phi$, $\boldsymbol{C}_\phi$, $\boldsymbol{f}_\phi(\boldsymbol{x}_\phi(t))$ and $\boldsymbol{b}_\phi(t)$ are the vectors and matrices of the linear and nonlinear elements of circuit ϕ,
- $\boldsymbol{G}_1, \cdots, \boldsymbol{G}_m$, $\boldsymbol{C}_1, \cdots, \boldsymbol{C}_m$, $\boldsymbol{f}_1(\boldsymbol{x}_1(t)), \cdots, \boldsymbol{f}_m(\boldsymbol{x}_m(t))$, $\boldsymbol{b}_1(t), \cdots, \boldsymbol{b}_m(t)$ and $\boldsymbol{R}_1, \cdots, \boldsymbol{R}_m$ are the vectors and matrices of the m subsections,
- and $\boldsymbol{D}_1, \cdots, \boldsymbol{D}_m$ are selector matrices that map the port voltages and currents of the subsections to the state space of circuit ϕ.

With the ability to format any nonlinear circuit into macromodel form, we can now move to describing the reduction technique. The macromodel equations described above will be used as the starting point for reduction, and the newly reduced equations could be

readily plugged into a larger system. This reduction in the size of the equations will result in simulation speedup.

5.2 Reduction Technique

Similar to the previous chapter, the reduction technique is split into two steps. The first step aims to reduce the size of the MNA equations by reducing the size of the linear matrices. This is achieved by using proper orthogonal decomposition, where the MNA equations are projected onto a reduced subspace. While the reduction in the size of the equations is very significant, speedup will continue to be limited. The reason behind this is the presence of the nonlinear elements.

When simulating nonlinear circuits, we need to be able to evaluate the nonlinear vector at any point in time. However, we do not possess a way to do that if the system is reduced. In fact, the only possible thing to do is to revert back to the original system in every iteration of the simulation just to evaluate the nonlinear vector and its derivative. This greatly limits speedup.

This brings us to the next step of reduction: artificial neural networks. Artificial neural networks have been used, with great success, to model the behaviour of any nonlinear function to any desired accuracy. In our proposed method, we will use feedforward neural networks to emulate the behaviour of our nonlinear vector. Since the derivative is also needed to solve the Newton-Raphson method, we will also compute the derivative of the neural network.

5.2.1 Proper Orthogonal Decomposition

The first step of our proposed reduction technique is proper orthogonal decomposition. In this step, we tackle the large size of the matrices of the original system. Using POD, we can project the original system onto a reduced subspace $\boldsymbol{Q}$ using congruence transformation:

$$\begin{aligned} \hat{\boldsymbol{G}}\hat{\boldsymbol{x}}(t) + \hat{\boldsymbol{C}}\dot{\hat{\boldsymbol{x}}}(t) + \hat{\boldsymbol{f}}(\hat{\boldsymbol{x}}(t)) &= \hat{\boldsymbol{R}}\boldsymbol{u}(t) + \hat{\boldsymbol{b}}(t) \\ \boldsymbol{i}(t) &= \hat{\boldsymbol{R}}^T\hat{\boldsymbol{x}}(t) \end{aligned} \tag{5.3}$$

where

$$\hat{\boldsymbol{G}} = \boldsymbol{Q}^T\boldsymbol{G}\boldsymbol{Q} \tag{5.4a}$$

$$\hat{\boldsymbol{C}} = \boldsymbol{Q}^T\boldsymbol{C}\boldsymbol{Q} \tag{5.4b}$$

$$\hat{\boldsymbol{R}} = \boldsymbol{Q}^T\boldsymbol{R} \tag{5.4c}$$

$$\hat{\boldsymbol{b}}(t) = \boldsymbol{Q}^T\boldsymbol{b}(t) \tag{5.4d}$$

$$\hat{\boldsymbol{f}}(\hat{\boldsymbol{x}}(t)) = \boldsymbol{Q}^T\boldsymbol{f}(\boldsymbol{x}(t)) \tag{5.4e}$$

The congruence transformation matrix $\boldsymbol{Q}$ is an orthonormal matrix that is constructed and defined over a specific range. This range is usually a set of load or input conditions. If the size of this congruence transformation matrix, q, is chosen carefully such that $q << n$, where n is the size of the original system, then we can achieve a large speedup in simulation. The following section describes how this congruence transformation matrix is constructed.

Congruence Transformation Matrix

The subspace onto which the reduced-order system will be projected is defined over a specific range. This range could be over various input and loading conditions. The manner through which we get this subspace is by simulating the original system over this range. The circuit response over this range is then collected to construct the subspace:

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{x}_1(t) & \boldsymbol{x}_2(t) & \cdots & \boldsymbol{x}_s(t) \end{bmatrix} \tag{5.5}$$

Predictably, this subspace will contain many vectors that will be linearly dependent given the nature of electronic circuits and the way they behave over a similar range of inputs and loads. To capture the most dominant directions of this subspace, we can use singular value decomposition:

$$\boldsymbol{K} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{5.6}$$

In a similar fashion to what we did in the previous chapter, the congruence transformation matrix $\boldsymbol{Q}$ is constructed from the first few columns in $\boldsymbol{U}$ making it an orthonormal subspace. Figure 5.3 shows a graphical depiction of this process.

Circuit Simulation

With the reduced subcircuit ready to use, we can plug it into a larger circuit that contains other reduced or non-reduced subcircuits in addition to linear and nonlinear elements. The MNA equations for this overall circuit would look like:

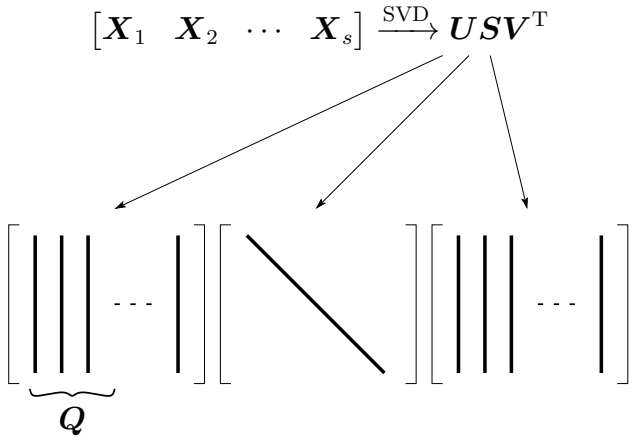

Fig. 5.3 Visual representation of the SVD process.

$$\begin{bmatrix} \boldsymbol{G}_\phi & \hat{\boldsymbol{D}}_1\hat{\boldsymbol{R}}_1^T & \cdots & \hat{\boldsymbol{D}}_m\hat{\boldsymbol{R}}_m^T \\ \hat{\boldsymbol{R}}_1\hat{\boldsymbol{D}}_1^T & \hat{\boldsymbol{G}}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ \hat{\boldsymbol{R}}_m\hat{\boldsymbol{D}}_m^T & 0 & 0 & \hat{\boldsymbol{G}}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_\phi(t) \\ \hat{\boldsymbol{x}}_1(t) \\ \vdots \\ \hat{\boldsymbol{x}}_m(t) \end{bmatrix} \tag{5.7}$$

$$+ \begin{bmatrix} \boldsymbol{C}_\phi & 0 & \cdots & 0 \\ 0 & \hat{\boldsymbol{C}}_1 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \hat{\boldsymbol{C}}_m \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_\phi(t) \\ \dot{\hat{\boldsymbol{x}}}_1(t) \\ \vdots \\ \dot{\hat{\boldsymbol{x}}}_m(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{f}_\phi(\boldsymbol{x}_\phi(t)) \\ \hat{\boldsymbol{f}}_1(\hat{\boldsymbol{x}}_1(t)) \\ \vdots \\ \hat{\boldsymbol{f}}_m(\hat{\boldsymbol{x}}_m(t)) \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_\phi(t) \\ \hat{\boldsymbol{b}}_1(t) \\ \vdots \\ \hat{\boldsymbol{b}}_m(t) \end{bmatrix}$$

Unfortunately, in order to simulate this circuit we need to evaluate the nonlinear vector. Since we do not possess a way to evaluate this vector directly when the circuit is in reduced form, we will have to revert back to the original system and evaluate it. After the original nonlinear vector is evaluated, we can then return to the reduced system using the congruence transformation matrix. If there are several reduced macromodels in the circuit, this has to be done for every one of them:

$$\begin{bmatrix} \boldsymbol{f}_{\phi}(\boldsymbol{x}_{\phi}(t)) \\ \hat{\boldsymbol{f}}_1(\hat{\boldsymbol{x}}_1(t)) \\ \vdots \\ \hat{\boldsymbol{f}}_m(\hat{\boldsymbol{x}}_m(t)) \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_{\phi}(\boldsymbol{x}_{\phi}(t)) \\ \boldsymbol{Q}_1^T \boldsymbol{f}_1(\boldsymbol{Q}_1 \boldsymbol{x}_1(t)) \\ \vdots \\ \boldsymbol{Q}_m^T \boldsymbol{f}_m(\boldsymbol{Q}_m \boldsymbol{x}_m(t)) \end{bmatrix} \tag{5.8}$$

Furthermore, since we will be solving those nonlinear equations using the Newton-Raphson method, we also need to evaluate the derivative of the nonlinear vector to use it in evaluating the Jacobian matrix. If several reduced-order subcircuits are used, this has to be done for every one of them:

$$\begin{bmatrix} \dfrac{\partial \boldsymbol{f}_{\phi}(\boldsymbol{x}_{\phi}(t))}{\partial \boldsymbol{x}_{\phi}(t)} & 0 & \cdots & 0 \\ 0 & \dfrac{\partial \hat{\boldsymbol{f}}_1(\hat{\boldsymbol{x}}_1(t))}{\partial \hat{\boldsymbol{x}}_1(t)} & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \dfrac{\partial \hat{\boldsymbol{f}}_m(\hat{\boldsymbol{x}}_m(t))}{\partial \hat{\boldsymbol{x}}_m(t)} \end{bmatrix}$$

where

$$\frac{\partial \hat{\boldsymbol{f}}_m(\hat{\boldsymbol{x}}_m(t))}{\partial \hat{\boldsymbol{x}}_m(t)} = \boldsymbol{Q}^T \frac{\partial \boldsymbol{f}_m(\boldsymbol{x}_m(t))}{\partial \boldsymbol{x}_m(t)} \boldsymbol{Q} \tag{5.9}$$

It can be easily observed that the evaluation of the nonlinear vector and its derivative quickly become the bottleneck in any simulation where these reduced-order subcircuits are used. Similar to what we did in the previous chapter, we will use artificial neural networks to replace the nonlinear vector and its derivative to overcome this problem.

5.2.2 Feedforward Neural Networks

Artificial neural networks, and specifically feedforward ones, are known for their ability to model functions. In this work, we will use them in order to emulate the behaviour of the nonlinear vector. If the function describing the neural network's structure, weights, and biases is expressed as $\boldsymbol{f_{nn}}(\hat{\boldsymbol{x}}(t))$, then our aim is to create a neural network such that:

$$\boldsymbol{f_{nn}}(\hat{\boldsymbol{x}}(t)) \approx \hat{\boldsymbol{f}}(\hat{\boldsymbol{x}}(t)) \tag{5.10}$$

Since we also need to evaluate the derivative of the nonlinear vector, then we also need to estimate it using the derivative of the neural network as well:

$$\frac{\partial \boldsymbol{f_{nn}}(\hat{\boldsymbol{x}}(t))}{\partial \hat{\boldsymbol{x}}(t)} \approx \frac{\partial \hat{\boldsymbol{f}}(\hat{\boldsymbol{x}}(t))}{\partial \hat{\boldsymbol{x}}(t)} \tag{5.11}$$

While in the previous chapter, we managed to used one neural network to emulate the behaviour of the whole nonlinear vector, this time it will be more difficult. When simulating the circuits using the Harmonic Balance technique, those circuits are almost always operating around a point where they are acting almost linearly. Therefore, the value at every entry of the nonlinear vector does not really vary much from other entries throughout the whole simulation. The same cannot be said about a simulation in the time domain. In fact, the value of every entry of the nonlinear vector could exhibit a very wide dynamic range.

In order to address this problem, and not make the training of the neural network more difficult than it needs to be, we will use one network for every entry in the nonlinear vector. This is in contrast to what we did in the previous chapter where we used one network for the whole vector.

Figure 5.4 shows the structure of the network we used. The input to this network is

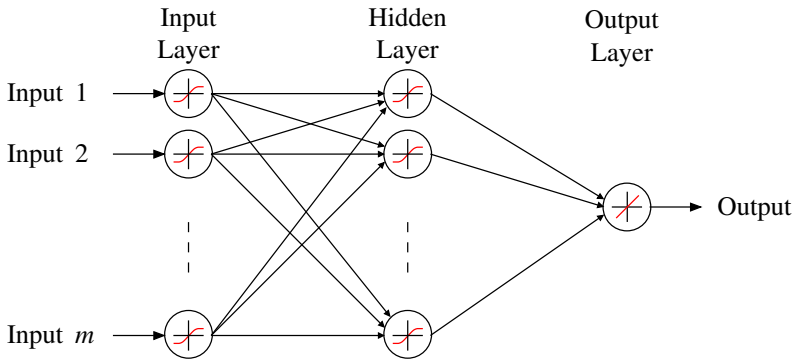

Fig. 5.4 Feedforward Neural Network.

the whole unknown vector $\hat{x}$. The output of this neural network is only one entry of the nonlinear vector $\hat{f}$.

Training of Neural Networks

For the neural networks to be able to emulate the behaviour of the nonlinear vector, they must be trained using the appropriate data. In this case, the reduced order subsection must be simulated over the predefined range once again. However, this time the data collected will be the reduced vector of unknowns, $\hat{x}$, and the reduced nonlinear vector $\hat{f}$. This data is collected for ever iteration in the simulation. The reduced vector of unknowns, $\hat{x}$, is the input to all the neural networks, while every entry of the reduced nonlinear vector, $\hat{f}$, will be the output of one of those neural networks.

Derivative of the Neural Network

To be able to use the neural network during the Newton-Raphson iteration, we will need to not only evaluate the function describing it, but also the derivative of the function describing it with respect to the input $\hat{x}$:

$$\frac{\partial f_{nn}(\hat{x})}{\partial \hat{x}} \tag{5.12}$$

To do that, we need to use the derivative chain rule and take into account every function inside the neural network. That includes the multiplication of the weights, the addition of the biases, and the activation functions. We also need to include any scaling functions used.

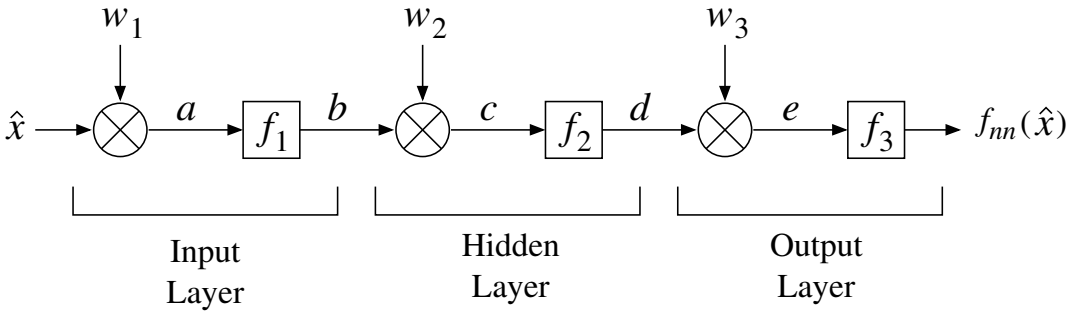

Fig. 5.5 Feedforward Neural Network.

Figure 5.5 shows in detail the internal structure of the neural network. Carefully, studying this structure, we can determine that the chain derivative is:

$$\frac{\partial f_{nn}(\hat{x})}{\partial \hat{x}} = \frac{\partial f_{nn}}{\partial e} \times \frac{\partial e}{\partial d} \times \frac{\partial d}{\partial c} \times \frac{\partial c}{\partial b} \times \frac{\partial b}{\partial a} \times \frac{\partial a}{\partial \hat{x}} \tag{5.13}$$

where,

$$\frac{\partial f_{nn}}{\partial e} = \frac{\partial f_3(e)}{\partial e} = \frac{e}{\partial e} = 1 \tag{5.14}$$

$$\frac{\partial e}{\partial d} = \frac{\partial (w_3 * d + b_3)}{\partial d} = w_3 \tag{5.15}$$

$$\frac{\partial d}{\partial c} = \frac{\partial f_2(c)}{\partial c} = 1 - f_2^2(c) \tag{5.16}$$

$$\frac{\partial c}{\partial b} = \frac{\partial (w_2 * b + b_2)}{\partial b} = w_2 \tag{5.17}$$

$$\frac{\partial b}{\partial a} = \frac{\partial f_1(a)}{\partial a} = 1 - f_1^2(a) \tag{5.18}$$

and

$$\frac{\partial a}{\partial \hat{x}} = \frac{\partial (w_1 * a + b_1)}{\partial a} = w_1 \tag{5.19}$$

Using the above, we now have a closed form that describes the derivative of the neural network that can be used in the calculation of the nonlinear Jacobian matrix.

5.3 Numerical Results

In this section, we provide a numerical example to show the accuracy and speedup achieved using the proposed reduction technique. The reader is provided with more details on specific decisions made regarding the size of the congruence transformation matrix, the number of hidden neurons, the training algorithms, and the training data among others.

The numerical example used to illustrate our method is a popular circuit used in literature and shown in Figure 5.6. This example is similar to the one in [96]. The number of nodes used is 50. All capacitors have a capacitance of 1F, and all resistors have a resistance of 1Ω. Lastly, the current through every diode is expressed as:

$$i_d(v) = e^{40v} - 1 \tag{5.20}$$

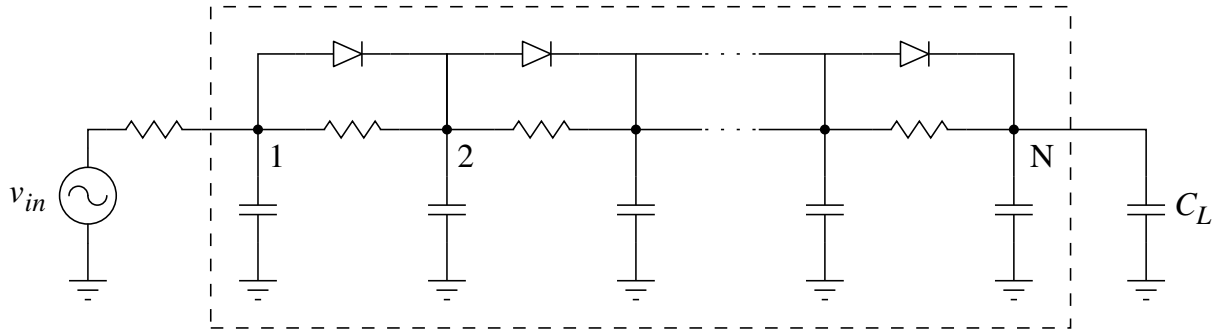

Fig. 5.6 Nonlinear transmission line.

5.3.1 First Step of Reduction

The size of the original non-reduced subsection is 50, while the size of the original circuit is 52. The first step of reduction requires the congruence transformation matrix $\boldsymbol{Q}$. To create this matrix, we must first construct the subspace $\boldsymbol{K}$.

To construct the subspace $\boldsymbol{K}$, the input voltage rise time was varied between 1s to 10s, the input voltage was varied between 1V to 2V, and finally the capacitive load was varied between 1F to 1F. For every simulation, a transient simulation was carried out till a stop time of 150s is reached. The total number of points collected to construct $\boldsymbol{K}$ was $12,016$.

Singular value decomposition was then applied to $\boldsymbol{K}$ to find the most dominant direction from which we can construct the congruence transformation matrix $\boldsymbol{Q}$. For this circuit, a $\boldsymbol{Q}$ of size 10 was chosen. Using this $\boldsymbol{Q}$, we can now perform the first step of the reduction where the sizes of the linear matrices can be reduced. This reduces the size of the original system from 50 to 10, or 20% of the original size.

5.3.2 Second Step of Reduction

The next step in the reduction is the creation of the feedforward neural networks. Since the size of $\hat{f}$ is 10, then we will need to create 10 different feedforward neural networks, one for every entry in $\hat{f}$. For this circuit we chose to use a three-layer neural network, with 30 input neurons, 30 hidden neurons, and 1 output neuron.

To ensure that the neural networks are emulating the behaviour of the nonlinear vector properly, they must be trained with input-output data that corresponds to the desired range of operation. This can be done by simulating the newly reduced circuit over the same range. Here again, $\hat{x}$ values will be collected for use as input, and $\hat{f}$ values will be collected for use as output. However, contrary to the first step and in order to ensure we collect as much data as possible, this time we will add the points taken in every Newton-Raphson iteration and not just the final solution. For this reason, the number of points collected for the training of the neural networks is 39,390.

5.3.3 Circuit Simulation

To test our reduced circuit subsection and measure its accuracy, it was simulated and compared to the original non-reduced system. Figure 5.7 shows a comparison between the original and reduced systems with the rise time varied from 1s to 30s. This comparison includes the risetimes 20s and 30s, both of which the subsection was not trained to model, but still was able to capture. It can be clearly seen in the figure that the original and reduced system responses are identical.

Next is testing whether the reduced system can capture the behaviour of the original under different voltage inputs. Figure 5.8 shows this comparison. For the range in which the circuit was trained, between 1V and 2V, the reduced matched the original system perfectly.

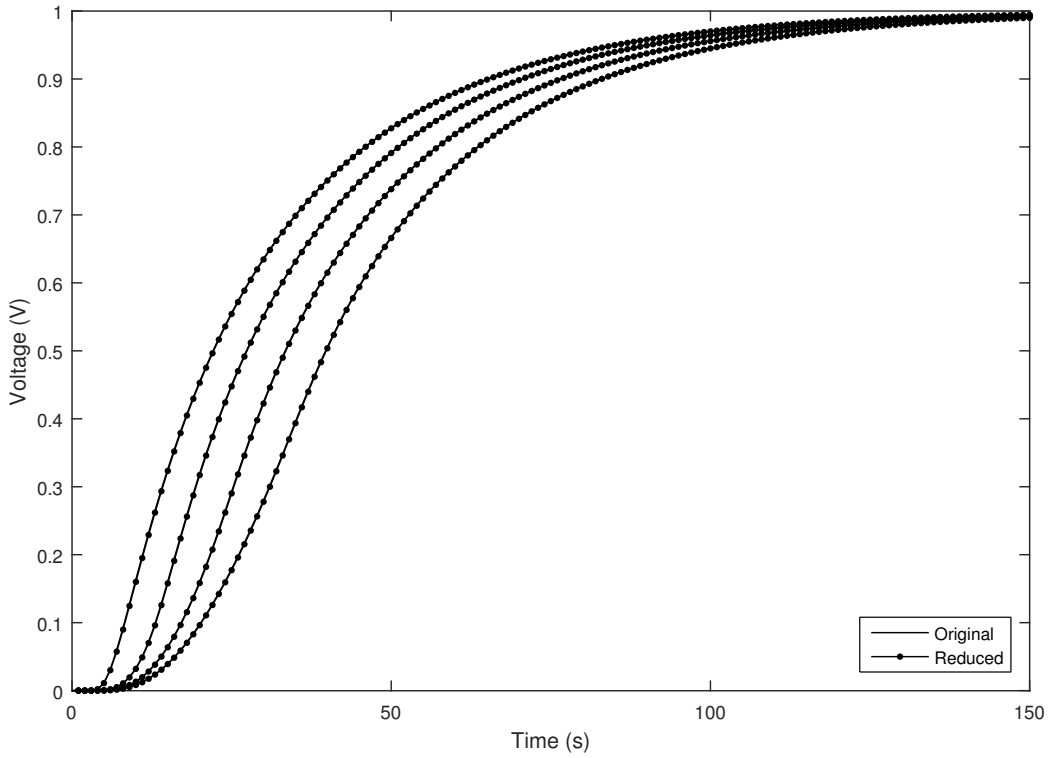

Fig. 5.7 Results for various rise times: 1s, 10s, 20s, and 30s.

For the 3V range, which the circuit was not trained to model, it was still able to match it, but with only small variations at the beginning.

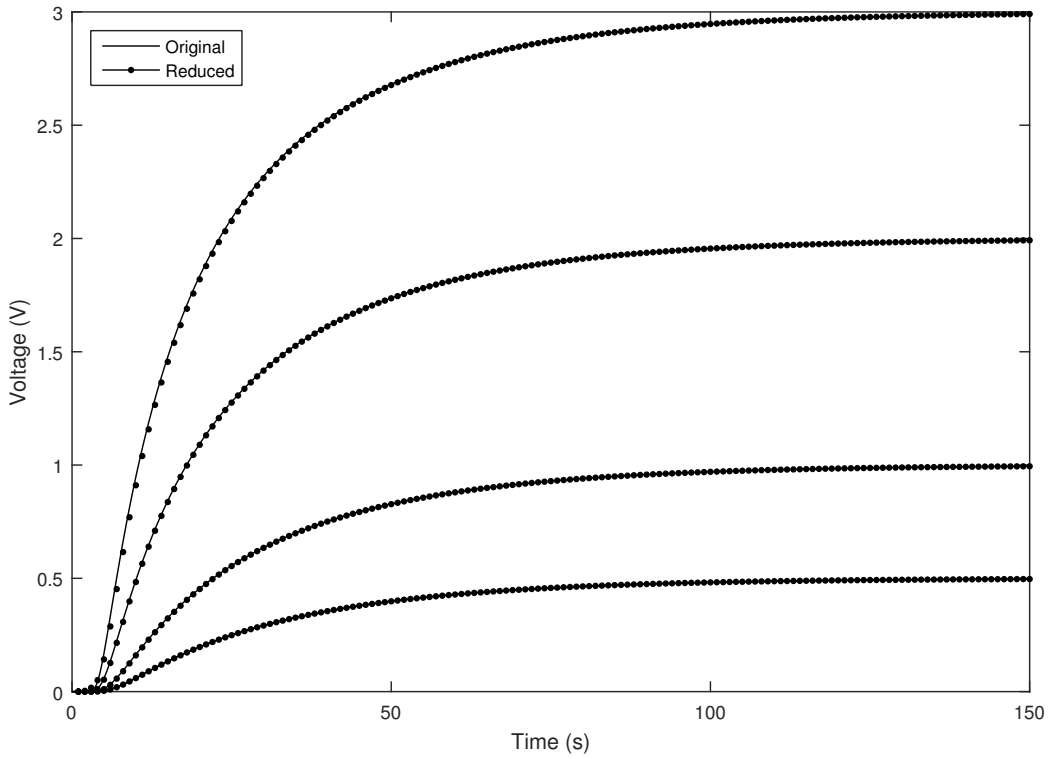

Fig. 5.8 Results for various voltage inputs 0.5V, 1V, 2V, and 3V.

Lastly, the reduced system was simulated and compared to the original one to check if they match when the capacitive load was varied. Figure 5.9 shows the comparison in the responses between the original and reduced system over a wide range of capacitances, some of which the original system was not trained to model. Here again, the reduced system was able to perfectly match the original.

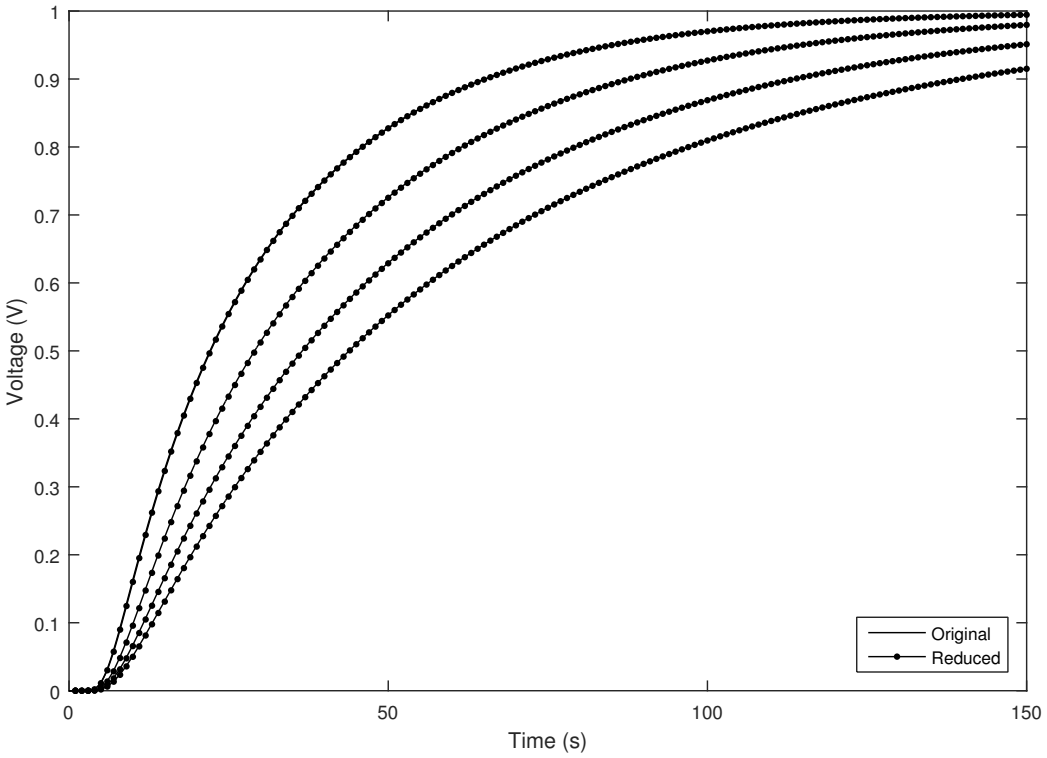

Fig. 5.9 Results for various capacitive loads: 1F, 10F, 20F, and 30F.

5.3.4 Speedup

The average speedup achieved by using the reduced circuit subsection over all the simulation that were run was five. One notable exception was when the input voltage was increased, where the speedup was much higher. For example, when the input voltage was only 1V, the speedup was five times, but when the input voltage was 2V and 3V, the speedup was 12 and 90 times respectively.

There are three main reasons for this speedup. The first reason is the reduction of the size of the matrices. While this reduction is significant, the speedup achieved by it is still very limited and on average was about 1.5 times only. The second, and most important reason for this speedup, was the substituting the nonlinear vectors with the neural networks. This eliminated the need to go back to the original system and therefore greatly sped up the simulation. The third reason for the speedup is the lack of need to do any calculation to compute the derivative of the neural networks. This derivative is needed during the Newton-Raphson iteration, but thanks to the activation functions used, their derivatives can be rewritten to be represented by the functions themselves as discussed in Section 2.2.1.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we presented a new algorithm to create reduced-order macromodels of nonlinear circuits suitable for addition to an overall Harmonic Balance or time domain system simulation. The algorithm is split into two steps, which are done consecutively. While each of those steps is an algorithm that can be used independently to create a reduced-order macromodel, in this work we combined them in such a way that allows us to realize the advantages of both algorithms.

The first step of reduction uses proper orthogonal decomposition to reduce the size of the matrices that represent the MNA equations of the macromodel. This method has been the cornerstone of many model-order reduction techniques, but is limited to only the linear parts of the system. Using congruence transformation, we can project the circuit equations onto an orthogonal reduced subspace. This subspace is constructed by simulating the original system over a predefined range of conditions. As a result, the macromodels are only guaranteed to be valid over this subspace.

The second step of reduction uses feedforward neural networks to replace the nonlinear vector and its derivative. While the first step greatly reduces the size of the MNA matrices, it remains limited because it lacks a way to reduce the size of the nonlinear vector. Therefore, a part of the original system, in this case the nonlinear vector, needs to be saved and used every time there is a need to evaluate it. A carefully trained neural network, however, can emulate the behaviour of this nonlinear vector and thus replace it completely. After this step, we can eliminate the need to save any part of the original model to resort to it during simulation, giving further speedup.

The macromodels generated using this algorithm are smaller in size and therefore faster to simulate relative to the original models, and they are accurate over a desired range of input or output conditions. Furthermore, it is possible, with some modifications, to generate reduced macromodels for different types of simulations. In this work, we focused on two types of simulations: Harmonic Balance and time domain.

In the case of Harmonic Balance, the congruence transformation matrix is constructed by applying singular value decomposition on the subspace containing the predefined range to obtain its most dominant directions. In the second step, one feedforward multi-layer neural network was used to represent the nonlinear vector. In the case of time domain, the congruence transformation matrix was constructed in a similar fashion, however, one neural network was used for every entry in the nonlinear vector. The reason for the two different approaches is the much wider dynamic range the nonlinear vector exhibits in the time domain compared to Harmonic Balance. In both cases, the derivative of the neural network replaces the derivative of the nonlinear functions of the original system.

Overall, this new algorithm has three main advantages. First, this algorithm is systematic. To generate a reduced-order macromodel of any circuit, the same steps of the algorithm are applied every time. If the user intends to create a macromodel for use in a Harmonic

Balance simulation, then the two steps of the algorithm are applied consecutively without any regard to other factors. There is no need for any modifications of any sort regardless of the situation.

Second, this algorithm is generic and is independent of the type of circuit under test. Whether it is a low noise amplifier, a frequency mixer, or a nonlinear transmission line has no effect on how the algorithm is applied. The internal structure of the circuit is of no concern. As long as the circuit is represented by its MNA equations and the type of simulation is known, the algorithm can be applied without any other considerations.

Lastly, this algorithm creates reduced-order macromodels. The circuits on which this algorithm is applied are formulated in macromodel format and the reduction technique is applied to the circuit in this specific format. This allows for the reduced-order macromodels to be added directly to other circuits without modifications. Anyone wishing to use these reduced macromodels can simply plug them into a larger system in the same way they would plug in a non-reduced macromodel or another circuit element.

6.2 Future Work

While the algorithm provides a promising new way of achieving speedup and maintaining accuracy, there are many areas which could explored for further improvement. Below we list some, where future work is possible and could be pursued:

- Connecting several macromodels together: In the numerical examples presented above, we only focused on one reduced macromodel at a time, however, it is entirely possible (and desirable) to connect several macromodels, whether reduced or not, together and run an overall system simulation. This could give greater insight into how much speedup is possible when all macromodels are replaced with their reduced-order form.

It will also give the user a better feel of how well this algorithm performs in "real-life" scenarios. Furthermore, it serves as an opportunity to discover possible areas of improvements or limitations.

- Establishing a more systematic way to choose the size of the congruence transformation matrix $\boldsymbol{Q}$: In this work, we relied on trial and error to decide on the size of this matrix, however, in the future it would be much more productive and faster to come up with a systematic way to decide on its size. While some work has been done in this field, there is still a lack of a comprehensive mathematical theory and therefore no clear way of making this decision. The singular values of the SVD could be taken into account to help with this decision. By investigating those values, it is possible to come up with some criteria to determine a cutoff after which increasing the size is no longer useful.

- Architecture of the neural networks: While in this thesis we focused on using feedforward neural networks only, there are many new emerging architectures in the literature today which could be promising. Using recurrent neural networks (which can save previous states), employing different types of activation functions, adding more layers, or exploring different types of training algorithms are just a few examples.

- Creating more general macromodels: the macromodels generated using this algorithm are guaranteed to work in the range that was defined at construction. In some cases, the macromodels do work outside this range with a high degree of accuracy, but that is not always the case. It would be interesting to see if it is possible to create macromodels that are general enough to work outside the predefined range.

- Macromodeling strongly nonlinear circuits: Some nonlinear circuits, such as inverters, exhibit very strong nonlinearities. An inverter or a ring oscillator could have many

transistors switching back and forth between different modes of operation many times in one simulation. That means there is a different function and model representing every transistor when this switch occurs and that could be very challenging to deal with from a reduction point of view. To capture this behaviour, the first step of reduction would have to have a large enough congruence transformation matrix. Not only is this a problem by itself since it limits size reduction, but also the bigger challenge here is that now the size of the neural network would have to be larger. The larger neural network is caused by the increased number of inputs and outputs in this case. Furthermore, since the underlying nonlinear functions are now much more complex and variant, it would be more challenging than ever to train those neural networks.

- Comparisons with other reduction techniques: In this work, we only compared the speedup gained by using the reduced-order macromodels with the original models serving as reference. However, in future it would be interesting to make a comparison between our algorithm and another nonlinear model-order reduction algorithms. Of course, to have a fair comparison both algorithms would have to be implemented in the same simulator and using the same environment.

References

[1] G. G. Gielen and R. A. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1825–1854, 2000.

[2] S. Borkar, R. Brodersen, J.-H. Chern, E. Naviasky, D. Saias, and C. Sodini, "Tomorrow's analog: Just dead or just different?," in *Proceedings of the 43rd annual Design Automation Conference*, pp. 709–710, ACM, 2006.

[3] M. J. Barragan and G. Leger, "A procedure for alternate test feature design and selection," Institute of Electrical and Electronics Engineers, 2015.

[4] K. Kundert and H. Chang, "Analog verification," in *Simulation and Verification of Electronic and Biological Systems*, pp. 157–171, Springer, 2011.

[5] K. Kundert, H. Chang, D. Jefferies, G. Lamant, E. Malavasi, and F. Sendig, "Design of mixed-signal systems-on-a-chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1561–1571, 2000.

[6] H. Chang and K. Kundert, "Verification of complex analog and RF IC designs," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 622–639, 2007.

[7] B. C. Lim, J.-E. Jang, J. Mao, J. Kim, and M. Horowitz, "Digital analog design: Enabling mixed-signal system validation," *IEEE Design & Test*, vol. 32, no. 1, pp. 44–52, 2015.

[8] G. W. Roberts, "Reducing the analog-digital productivity gap using time-mode signal processing," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 782–785, IEEE, 2014.

[9] B. A. Antao, "Trends in CAD of analog ICs," *IEEE Circuits and Devices Magazine*, vol. 12, no. 5, pp. 31–41, 1996.

[10] D. O. Pederson, "A historical review of circuit simulation," *Circuits and Systems, IEEE Transactions on*, vol. 31, no. 1, pp. 103–111, 1984.

[11] S. Madhu and R. Unnikrishnan, *Linear Circuit Analysis*. Prentice Hall, 1988.

[12] A. Vladimirescu, *The SPICE Book*. John Wiley & Sons, Inc., 1994.

[13] G. W. Roberts, *SPICE*. Oxford University Press, Inc., 1996.

[14] A. Vladirmirescu, "Shaping the history of SPICE," *Solid-State Circuits Magazine, IEEE*, vol. 3, no. 2, pp. 36–39, 2011.

[15] K. Kundert, *The Designer's Guide to Spice and Spectre®*. Springer Science & Business Media, 2006.

[16] C. Gu, "Algorithmic nonlinear macromodeling: Challenges, solutions and applications in analog/mixed-signal validation," in *Custom Integrated Circuits Conference (CICC), 2013 IEEE*, pp. 1–8, IEEE, 2013.

[17] M. Rewieński, "A perspective on Fast-SPICE simulation technology," in *Simulation and Verification of Electronic and Biological Systems*, pp. 23–42, Springer, 2011.

[18] J. Roychowdhury, "Algorithmic macromodelling methods for mixed-signal systems," in *VLSI Design, 2004. Proceedings. 17th International Conference on*, pp. 141–147, IEEE, 2004.

[19] S. K. Tiwary and R. A. Rutenbar, "Scalable trajectory methods for on-demand analog macromodel extraction," in *Proceedings of the 42nd annual design automation conference*, pp. 403–408, ACM, 2005.

[20] R. A. Rutenbar, G. G. Gielen, and J. Roychowdhury, "Hierarchical modeling, optimization, and synthesis for system-level analog and RF designs," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 640–669, 2007.

[21] M. Kanaan and R. Khazaka, "Model order reduction for nonlinear macromodeling of rf circuits," in *New Circuits and Systems Conference (NEWCAS), 2011 IEEE 9th International*, pp. 217–220, IEEE, 2011.

[22] M. Kanaan and R. Khazaka, "Nonlinear time-domain macromodeling using proper orthogonal decomposition and feedforward neural networks," in *Electrical and Computer Engineering (CCECE), 2017 IEEE 30th Canadian Conference on*, pp. 1–4, IEEE, 2017.

[23] C.-W. Ho, A. E. Ruehli, and P. A. Brennan, "The modified nodal approach to network analysis," *Circuits and Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 504–509, 1975.

[24] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design*. Springer Science & Business Media, 1983.

[25] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd ed., 1998.

[26] Q.-J. Zhang and K. C. Gupta, *Neural Networks for RF and Microwave Design*. Artech House, Inc., 2000.

[27] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pp. 318–362, Cambridge, MA, USA: MIT Press, 1986.

[29] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural computation*, vol. 3, no. 2, pp. 246–257, 1991.

[30] Q. Zhang and A. Benveniste, "Wavelet networks," *IEEE transactions on Neural Networks*, vol. 3, no. 6, pp. 889–898, 1992.

[31] G. R. Boyle, D. O. Pederson, B. M. Cohn, and J. E. Solomon, "Macromodeling of integrated circuit operational amplifiers," *Solid-State Circuits, IEEE Journal of*, vol. 9, no. 6, pp. 353–364, 1974.

[32] L. M. Brocco, S. P. Mccormick, and J. Allen, "Macromodeling CMOS circuits for timing simulation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 7, no. 12, pp. 1237–1249, 1988.

[33] G. Gielen, P. Wambacq, and W. M. Sansen, "Symbolic analysis methods and applications for analog circuits: A tutorial overview," *Proceedings of the IEEE*, vol. 82, no. 2, pp. 287–304, 1994.

[34] F. V. Fernandez and A. Rodriguez-Vazquez, "Symbolic analysis tools–The state-of-the-art," in *Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996 IEEE International Symposium on*, vol. 4, pp. 798–801 vol.4, May 1996.

[35] C. Borchers, "Symbolic behavioral model generation of nonlinear analog circuits," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 45, no. 10, pp. 1362–1371, 1998.

[36] Q. Yu and C. Sechen, "A unified approach to the approximate symbolic analysis of large analog integrated circuits," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 43, no. 8, pp. 656–669, 1996.

[37] A. H. Zaabab, Q.-J. Zhang, and M. Nakhla, "A neural network modeling approach to circuit optimization and statistical design," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 43, no. 6, pp. 1349–1358, 1995.

[38] F. Wang and Q.-J. Zhang, "Knowledge-based neural models for microwave design," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 45, no. 12, pp. 2333–2343, 1997.

[39] Q.-J. Zhang, K. C. Gupta, and V. K. Devabhaktuni, "Artificial neural networks for RF and microwave design–From theory to practice," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 51, no. 4, pp. 1339–1350, 2003.

[40] B. Gustavsen and A. Semlyen, "Rational approximation of frequency domain responses by vector fitting," *Power Delivery, IEEE Transactions on*, vol. 14, no. 3, pp. 1052–1061, 1999.

[41] B. Gustavsen and A. Semlyen, "A robust approach for system identification in the frequency domain," *Power Delivery, IEEE Transactions on*, vol. 19, no. 3, pp. 1167–1173, 2004.

[42] S. Lefteriu and A. C. Antoulas, "A new approach to modeling multiport systems from frequency-domain data," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 1, pp. 14–27, 2010.

[43] Y. Wang, C.-U. Lei, G. K. Pang, and N. Wong, "MFTI: Matrix-format tangential interpolation for modeling multi-port systems," in *Proceedings of the 47th Design Automation Conference*, pp. 683–686, ACM, 2010.

[44] M. Kabir and R. Khazaka, "Macromodeling of distributed networks from frequency-domain data using the Loewner matrix approach," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 60, no. 12, pp. 3927–3938, 2012.

[45] J. R. Phillips, "Projection-based approaches for model reduction of weakly nonlinear, time-varying systems," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 22, no. 2, pp. 171–187, 2003.

[46] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 9, no. 4, pp. 352–366, 1990.

[47] J. E. Bracken, V. Raghavan, and R. A. Rohrer, "Interconnect simulation with asymptotic waveform evaluation (AWE)," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 39, no. 11, pp. 869–878, 1992.

[48] T. K. Tang and M. S. Nakhla, "Analysis of high-speed VLSI interconnects using the asymptotic waveform evaluation technique," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 11, no. 3, pp. 341–352, 1992.

[49] E. Chiprout and M. S. Nakhla, *Asymptotic Waveform Evaluation*. Springer, 1994.

[50] E. Chiprout and M. S. Nakhla, "Analysis of interconnect networks using complex frequency hopping (CFH)," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 14, no. 2, pp. 186–200, 1995.

[51] P. Feldmann and R. W. Freund, "Efficient linear circuit analysis by Padé approximation via the lanczos process," in *Proceedings of the Conference on European Design Automation*, EURO-DAC '94, (Los Alamitos, CA, USA), pp. 170–175, IEEE Computer Society Press, 1994.

[52] P. Feldmann and R. W. Freund, "Efficient linear circuit analysis by Padé approximation via the Lanczos process," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 14, no. 5, pp. 639–649, 1995.

[53] P. Feldmann and R. W. Freund, "Reduced-order modeling of large linear subcircuits via a block Lanczos algorithm," in *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, pp. 474–479, ACM, 1995.

[54] L. M. Silveira, M. Kamon, and J. White, "Efficient reduced-order modeling of frequency-dependent coupling inductances associated with 3-D interconnect structures," *Components, Packaging, and Manufacturing Technology, Part B: Advanced Packaging, IEEE Transactions on*, vol. 19, no. 2, pp. 283–288, 1996.

[55] K. J. Kerns, I. L. Wemple, and A. T. Yang, "Stable and efficient reduction of substrate model networks using congruence transforms," in *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pp. 207–214, IEEE Computer Society, 1995.

[56] K. J. Kerns and A. T. Yang, "Stable and efficient reduction of large, multiport RC networks by pole analysis via congruence transformations," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 16, no. 7, pp. 734–744, 1997.

[57] K. J. Kerns and A. T. Yang, "Preservation of passivity during RLC network reduction via split congruence transformations," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 7, pp. 582–591, 1998.

[58] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: Passive reduced-order interconnect macromodeling algorithm," in *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pp. 58–65, IEEE Computer Society, 1997.

[59] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: Passive reduced-order interconnect macromodeling algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 645–654, Aug 1998.

[60] A. Odabasioglu, M. Celik, and L. T. Pileggi, "Practical considerations for passive reduction of RLC circuits," in *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pp. 214–220, IEEE Press, 1999.

[61] B. C. Moore, "Principal component analysis in linear systems: Controllability, observability, and model reduction," *Automatic Control, IEEE Transactions on*, vol. 26, no. 1, pp. 17–32, 1981.

[62] L. Pernebo and L. M. Silverman, "Model reduction via balanced state space representations," *Automatic Control, IEEE Transactions on*, vol. 27, no. 2, pp. 382–387, 1982.

[63] P. Rabiei and M. Pedram, "Model order reduction of large circuits using balanced truncation," in *Design Automation Conference, 1999. Proceedings of the ASP-DAC'99. Asia and South Pacific*, pp. 237–240, IEEE, 1999.

[64] M. Kamon, F. Wang, and J. White, "Generating nearly optimally compact models from Krylov-subspace based reduced-order models," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 47, no. 4, pp. 239–248, 2000.

[65] J. R. Phillips and L. M. Silveira, "Poor man's TBR: A simple model reduction scheme," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 1, pp. 43–55, 2005.

[66] D. Vasilyev, M. Rewienski, and J. White, "Macromodel generation for BioMEMS components using a stabilized balanced truncation plus trajectory piecewise-linear approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 285–293, Feb 2006.

[67] J. Phillips, L. Daniel, and L. M. Silveira, "Guaranteed passive balancing transformations for model order reduction," in *Design Automation Conference, 2002. Proceedings. 39th*, pp. 52–57, 2002.

[68] J. R. Phillips, L. Daniel, and L. M. Silveira, "Guaranteed passive balancing transformations for model order reduction," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 22, no. 8, pp. 1027–1041, 2003.

[69] J. R. Phillips, "Projection frameworks for model reduction of weakly nonlinear systems," in *Proceedings of the 37th Annual Design Automation Conference*, pp. 184–189, ACM, 2000.

[70] J. R. Phillips, "Automated extraction of nonlinear circuit macromodels," in *IEEE Custom Integrated Circuits Conference*, pp. 451–454, 2000.

[71] J. Roychowdhury, "Reduced-order modelling of linear time-varying systems," in *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pp. 92–95, ACM, 1998.

[72] J. Roychowdhury, "Reduced-order modeling of time-varying systems," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 46, no. 10, pp. 1273–1288, 1999.

[73] P. Li and L. T. Pileggi, "NORM: Compact model order reduction of weakly nonlinear systems," in *Proceedings of the 40th annual Design Automation Conference*, pp. 472–477, ACM, 2003.

[74] P. Li and L. T. Pileggi, "Compact reduced-order modeling of weakly nonlinear analog and RF circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 2, pp. 184–203, 2005.

[75] C. Gu, "QLMOR: A new projection-based approach for nonlinear model order reduction," in *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pp. 389–396, IEEE, 2009.

[76] C. Gu, "Qlmor: a projection-based nonlinear model order reduction approach using quadratic-linear representation of nonlinear systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 9, pp. 1307–1320, 2011.

[77] M. Ma and R. Khazaka, "Nonlinear macromodeling using model order reduction," in *Electrical Performance of Electronic Packaging, 2005. IEEE 14th Topical Meeting on*, pp. 131–134, IEEE, 2005.

[78] B. Nouri, M. S. Nakhla, and R. Achar, "A novel algorithm for efficient simulation of nonlinear transmission lines for RF applications via model order reduction," in *2015 IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO)*, pp. 1–3, IEEE, 2015.

[79] M. Rewienski and J. White, "A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices," in *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, pp. 252–257, Nov 2001.

[80] M. Rewieński and J. White, "A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 22, no. 2, pp. 155–170, 2003.

[81] S. K. Tiwary and R. A. Rutenbar, "Faster, parametric trajectory-based macromodels via localized linear reductions," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pp. 876–883, ACM, 2006.

[82] N. Dong and J. Roychowdhury, "Piecewise polynomial nonlinear model reduction," in *Design Automation Conference, 2003. Proceedings*, pp. 484–489, IEEE, 2003.

[83] N. Dong and J. Roychowdhury, "General-purpose nonlinear model-order reduction using piecewise-polynomial representations," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 2, pp. 249–264, 2008.

[84] C. Gu and J. Roychowdhury, "Model reduction via projection onto nonlinear manifolds, with applications to analog circuits and biochemical systems," in *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, pp. 85–92, IEEE, 2008.

[85] C. Gu and J. Roychowdhury, "Manifold construction and parameterization for nonlinear manifold-based model reduction," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, pp. 205–210, IEEE Press, 2010.

[86] Y. Hayashi, M. Sakata, and S. I. Gallant, "Multi-layer versus single-layer neural networks and an application to reading hand-stamped characters," in *International Neural Network Conference*, pp. 781–784, Springer, 1990.

[87] L. Zhang, Q.-J. Zhang, and J. Wood, "Statistical neuro-space mapping technique for large-signal modeling of nonlinear devices," *IEEE Transactions on Microwave Theory and Techniques*, vol. 56, no. 11, pp. 2453–2467, 2008.

[88] H. Kabir, Y. Wang, M. Yu, and Q.-J. Zhang, "Neural network inverse modeling and applications to microwave filter design," *IEEE Transactions on Microwave Theory and Techniques*, vol. 56, no. 4, pp. 867–879, 2008.

[89] P. M. Watson and K. C. Gupta, "Em-ann models for microstrip vias and interconnects in dataset circuits," *IEEE Transactions on Microwave Theory and Techniques*, vol. 44, no. 12, pp. 2495–2503, 1996.

[90] Y. Fang, M. C. Yagoub, F. Wang, and Q.-J. Zhang, "A new macromodeling approach for nonlinear microwave circuits based on recurrent neural networks," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 48, no. 12, pp. 2335–2344, 2000.

[91] Q. Zhang, Y. Cao, and I. Erdin, "Fast IO buffer modeling using neural network methods," in *Electronic Packaging Technology & High Density Packaging (ICEPT-HDP), 2010 11th International Conference on*, pp. 666–669, IEEE, 2010.

[92] B. Mutnury, M. Swaminathan, and J. P. Libous, "Macromodeling of nonlinear digital I/O drivers," *IEEE Transactions on Advanced Packaging*, vol. 29, no. 1, pp. 102–113, 2006.

[93] H. Sharma and Q.-J. Zhang, "Automated time domain modeling of linear and nonlinear microwave circuits using recurrent neural networks," *International Journal of RF and Microwave Computer-Aided Engineering*, vol. 18, no. 3, pp. 195–208, 2008.

[94] M. Isaksson, D. Wisell, and D. Ronnow, "Wide-band dynamic modeling of power amplifiers using radial-basis function neural networks," *IEEE transactions on microwave theory and techniques*, vol. 53, no. 11, pp. 3422–3428, 2005.

[95] F. Mkadem and S. Boumaiza, "Physically inspired neural network model for RF power amplifier behavioral modeling and digital predistortion," *IEEE Transactions on Microwave Theory and Techniques*, vol. 59, no. 4, pp. 913–923, 2011.

[96] Y. Chen and J. White, "A quadratic method for nonlinear model order reduction," in *Proc. Int. Conf. Modeling and Simulation of Microsystems*, pp. 477–480, 2000.