Model Order Reduction of Nonlinear Dynamical Systems

by

Chenjie Gu

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jaijeet Roychowdhury, Chair
Professor Robert Brayton
Professor Jon Wilkening

Fall 2011

Model Order Reduction of Nonlinear Dynamical Systems

Abstract

Model Order Reduction of Nonlinear Dynamical Systems

by

Chenjie Gu

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Science

University of California, Berkeley

Professor Jaijeet Roychowdhury, Chair

Higher-level representations (macromodels, reduced-order models) abstract away unnecessary implementation details and model only important system properties such as functionality. This methodology – well-developed for linear systems and digital (Boolean) circuits – is not mature for general nonlinear systems (such as analog/mixed-signal circuits). Questions arise regarding abstracting/macromodeling nonlinear dynamical systems: What are "important" system properties to preserve in the macromodel? What is the appropriate representation of the macromodel? What is the general algorithmic framework to develop a macromodel? How to automatically derive a macromodel from a white-box/black-box model? This dissertation presents techniques for solving the problem of macromodeling nonlinear dynamical systems by trying to answer these questions.

We formulate the nonlinear model order reduction problem as an optimization problem and present a general nonlinear projection framework that encompasses previous linear projection-based techniques as well as the techniques developed in this dissertation. We illustrate that nonlinear projection is natural and appropriate for reducing nonlinear systems, and can achieve more compact and accurate reduced models than linear projection.

The first method, ManiMOR, is a direct implementation of the nonlinear projection framework. It generates a nonlinear reduced model by projection on a general-purpose nonlinear manifold. The proposed manifold can be proven to capture important system dynamics such as DC and AC responses. We develop numerical methods that alleviates the computational cost of the reduced model which is otherwise too expensive to make the reduced order model of any value compared to the full model.

The second method, QLMOR, transforms the full model to a canonical QLDAE representation and performs Volterra analysis to derive a reduced model. We develop an algorithm that can mechanically transform a set of nonlinear differential equations to another set of equivalent nonlinear differential equations that involve only quadratic terms of state variables, and therefore it avoids any problem brought by previous Taylor-expansion-based methods. With the QLDAE representation, we develop the corresponding model order reduction algorithm that extends and generalizes previously-developed Volterra-based technique.

The third method, NTIM, derives a macromodel that specifically captures timing/phase responses of a nonlinear system. We rigorously define the phase response for a non-autonomous system, and derive the dynamics of the phase response. The macromodel emerges as a scalar, nonlinear time-varying differential equation that can be computed by performing Floquet analysis of the full model. With the theory developed, we also present efficient numerical methods to compute the macromodel.

The fourth method, DAE2FSM, considers a slightly different problem – finite state machine abstraction of continuous dynamical systems. We present an algorithm that learns a Mealy machine from a set of differential equations from its input-output trajectories. The algorithm explores the state space in a smart way so that it can identify the underlying finite state machine using very few information about input-output trajectories.

Acknowledgments

I would like to thank all those who supported me throughout my doctoral studies.

First and foremost, I would like to thank my advisor Professor Jaijeet Roychowdhury. Without his guidance, this dissertation would have been impossible. I really appreciate his insight into numerous aspects in numerical simulation and analog/RF computer-aided design, as well as his sharpness, enthusiasm, care and encouragement. Learning from and working with him was an invaluable and unique experience.

I would also like to thank dissertation and examination committee members, Professors Robert Brayton, Jon Wilkening, Andreas Kuehlmann. They have spent a lot of time examining and guiding my research directions, and have provided valuable comments and suggestions along my research in the last two years of my doctoral studies.

In addition, I would like to express my special thanks to several other professors: Professor Alper Demir for many fruitful discussions on phase macromodeling, Professor Sanjit Seshia for his insights into verification and model checking, Professor Claire Tomlin for her series of valuable lectures in linear, nonlinear and hybrid control systems, Professor Sachin Sapatnekar and Kia Bazargan for their early introduction to me the field of computer-aided design and electronic design automation.

I would also like to extend my thanks to those who have been helping me in my career planning and job applications, including but not limited to Joel Phillips, Sani Nassif, Eric Keiter, Heidi Thornquist, Noel Menezes, Chirayu Amin, Qi Zhu, Yuan Xie, Yu Wang.

Working in Professor Jaijeet Roychowdhury's group has been a rewarding experience, partly because of all the wonderful people I have worked with. Especially I would like to thank Ning Dong, Xiaolue Lai, Ting Mei for their friendly sharing of knowledge and experiences, as well as many helpful suggestions in research and life. I would also like to thank Prateek Bhansali with whom I have worked for more than three years, and David Amsallem who has been a post-doctoral researcher in the group.

The DOP Center (Donald O. Pederson Center for Electronics Systems Design) in Berkeley is without doubt a legendary research center and a great place to work in. My thanks also extend to friends in the DOP center, including but not limited to Baruch Sterin, Sayak Ray, Dan Holcomb, Wenchao Li, Susmit Jha, Bryan Brady, Pierluigi Nuzzo, Alberto Puggelli, Xuening Sun, Liangpeng Guo, Jia Zou.

My gratitude also goes to friends in other research groups in Berkeley and Minnesota, including but not limited to Jiashu Chen, John Crossley, Lingkai Kong, Yue Lu, Sriramkumar Venugopalan, Chao-Yue Lai, Humberto Gonzale, Nicholas Knight, Chunhui Gu, Zhichun Wang, Weikang Qian, Qunzeng Liu.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Macromodels and Motivating Applications

Engineers and scientists build models for many systems in various domains. Computer simulation of these models predicts system behaviors, and thus helps us understand existing systems and design new systems.

Typically, we have a hierarchy of models of different levels of abstraction. Higher-level models abstract away unimportant details of lower-level models, and thus are easier to understand and more efficient to simulate. For example, in digital circuits, we have models at the gate level describing the gate's truth-table and delay/power performance, models at the RTL level describing the register-level timing and dynamics, models at the finite state machine level describing the functionality of the circuit, *etc.*; in analog circuits, we have models at the device level describing how each electron move, models at the transistor level describing the current-voltage (I-V) and charge-voltage (Q-V) relationship of a transistor, models at the sub-circuit level describing sub-circuit performances such as the gain and bandwidth of an amplifier, models at the system level describing power/functionality of the circuit, *etc.*.

In the process of circuit design, a top-down synthesis approach is commonly employed. That is, starting with system specifications, we map from higher-level models to lower-level models iteratively until we reach the transistor-level implementation.

Macromodeling, or model order reduction, is the reverse procedure of synthesis. It refers to the generation of higher-level models from lower-level models.

Traditionally, macromodeling is done manually. It usually involves a lot of domain knowledge in a specific field, and is rather ad hoc than systematic. As a result, the macromodeling process takes long time. Moreover, while the manual macromodeling process usually captures responses of major interest, it may miss equally important corner cases, and may create a defective macromodel.

1.1.1 Mathematical Models

Depending on applications, different mathematical models are used to model a dynamical system describing how the state variables (such as voltages and currents in a circuit) evolve with time and inputs. Table 1.1 enumerates a few commonly used models. We classify these models according to continuity of state variables, continuity of time, and deterministicity, as shown in Table 1.1 and Fig. 1.1.

Table 1.1: Commonly used mathematical models

Model	State Variables	Time	Deterministicity
Differential Algebraic Equations (DAE)	Cont.	Cont.	Deterministic
Ordinary Differential Equations (ODE)	Cont.	Cont.	Deterministic
Partial Differential Equations (PDE)	Cont.	Cont.	Deterministic
Difference Equations (DE)	Cont.	Disc.	Deterministic
Stochastic Differential Equations (SDE)	Cont.	Cont.	Stochastic
Continuous-Time Markov Chain (CTMC)	Disc.	Cont.	Stochastic
Discrete-Time Markov Chain (DTMC)	Disc.	Disc.	Stochastic
Finite State Machines (FSM)	Disc.	Disc.	(Non-)Deterministic
Hybrid Systems (HS)	Both	Cont.	(Non-)Deterministic

While these models are useful in modeling different kinds of systems, and the reduction of these models all deserve further investigation, we mainly focus on ODE and DAE models in this dissertation, because such models have been widely recognized as standard models in circuit simulation as well as several other fields. We will only mention and comment the model reduction of a few other models in Table 1.1.

In terms of DAE models, we further restrict ourselves in input-output systems in the form of

$$\frac{d}{dt}q(x) + f(x) + Bu(t) = 0, \quad y = h(x), \tag{1.1}$$

where $x \in \mathbb{R}^N$ are the state variables; $q(x), f(x) : \mathbb{R}^N \to \mathbb{R}^N$, $h(x) : \mathbb{R}^N \to \mathbb{R}^{N_o}$ are functions; $B \in \mathbb{R}^{N\times N_i}$; $u \in \mathbb{R}^{N_i}$ are inputs, $y \in \mathbb{R}^{N_o}$ are outputs; N is the dimension of the state space (the order, or the size of the model); N_i is the number of inputs; N_o is the number of outputs.

For simplicity, sometimes we consider ODE models by choosing $q(x) = -x$ in (1.1), *i.e.*,

$$\frac{d}{dt}x = f(x) + Bu(t), \quad y = h(x). \tag{1.2}$$

However, DAE/ODE models (1.1) and (1.2) are most commonly used in many practical systems, and the techniques we develop easily extends to other forms of DAE/ODE models, such as the ones listed in Table 1.2.

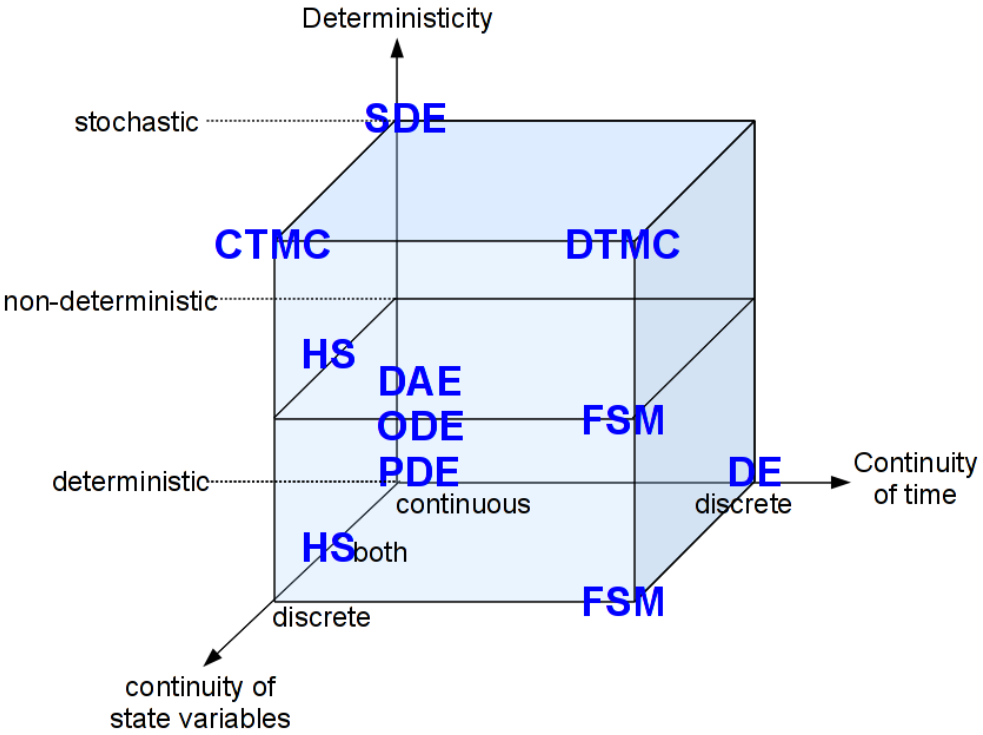


Figure 1.1: Classification of mathematical models.

Table 1.2: Other DAE/ODE models

Name	Model
Nonlinear descriptor system	$\frac{d}{dt}q(x) = f(x,u), \quad y = h(x)$
Implicit nonlinear system	$f(\frac{d}{dt}x, x, u, y) = 0$

1.1.2 Applications

The macromodels, or reduced order models, can be used in various contexts to help understanding and designing a system.

The straightforward application is to speedup simulations of a large system. For example, circuits nowadays can easily involve millions of transistors and even more number of parasitic resistors and capacitors. While analog circuits used to be stand-alone and have at most hundreds of transistors, they are now extensively coupled to digital circuits for feedback and calibration, thus also leading to a large number of equations. With reduced-order models of each sub-circuit block, we may replace the original sub-circuits by their reduced-order models to obtain a smaller system of equations with little accuracy compromise. Therefore the speedup of the simulation may be significant, and we can afford to simulate larger systems that are otherwise impossible to simulate within a reasonable amount of time.

Similarly, model order reduction enables a bottom-up approach to create a hierarchy of models, and to finally estimate higher-level performance metrics of a large system. For example, in aggressive memory designs, device-level PDE simulations are necessary to capture detailed nonlinear behaviors of transistors. This can involve a large number of equations by applying finite element methods on the underlying PDEs. Model order reduction can therefore automatically generate a smaller model for each transistor which can then be used to estimate circuit performances such as read/write access time of a memory cell.

Reduced order models can also be used in design optimization (*e.g.*, transistor sizing). In particular, parameterized reduced models [4–9] capture important system response with dependence on parameters. Therefore, they simplify the optimization problem and make the problem computationally more tractable. Some reduced models also naturally carries sensitivity information with respect to inputs/parameters, and this is very useful in gradient-based optimization methods.

Reduced order models may also help understanding nonlinear systems. To the best of our knowledge, there does not exist techniques to analyze behaviors of a system modeled by a large number of differential equations, except for extensive numerical simulation. Bifurcation theory and systems theory are developed but they are usually applied in very low dimensional systems (*e.g.*, 2-dimensional). Therefore, rather than a full-size model, a reduced order model may be systematically analyzed, and can provide useful intuition of how the large system works.

Since the MOR techniques are developed over the underlying mathematical models, they are generally applicable to many disciplines, including electronic circuits (interconnects [5,7,

10–14], electromagnetics [15–19], power grids [20–22], RF circuits [23–28], oscillators [29–32]), micro-electro-mechanical systems [33–39] fluid dynamics [40–46], biological systems [41, 47–51], neuroscience [52–59], chemical engineering [60–67], mechanical engineering [68–71], *etc.*.

1.2 Problem Overview

Loosely speaking, model order reduction aims to generating "simpler" models that capture "important" dynamics of original full models.

Here, "model" refers to a mathematical model that describes system behaviors. The model can be linear or nonlinear, continuous or discrete value, continuous or discrete time, deterministic or stochastic, as shown in Table 1.1.

"Simple" typically means computationally efficient, but may also represent other properties of the model, such as sparsity of matrices in the model, number of parameters, number of state variables, correlation among state variables, *etc.*.

What "important" dynamics mean depends highly on applications. For examples, it can be transfer functions over a certain frequency range (for LTI systems), the time-domain waveforms, stability, passivity, *etc.*.

For DAEs (1.1), model order reduction typically means deriving a system of reduced DAEs

$$\frac{d}{dt}q_r(x_r) + f_r(x_r) + B_r u(t) = 0, \quad y = h_r(x_r), \tag{1.3}$$

where $x_r \in \mathbb{R}^{N_r}$ are the state variables of the reduced model; $q_r(x_r), f_r(x_r) : \mathbb{R}^{N_r} \to \mathbb{R}^{N_r}$, $h_r(x_r) : \mathbb{R}^{N_r} \to \mathbb{R}^{N_o}$ are functions; $B \in \mathbb{R}^{N_r \times N_i}$; N_r is the order of the reduced model.

1.2.1 Challenges

While we will have more detailed discussions about challenges in nonlinear MOR in Chapter 2 and other chapters, we highlight the major ones in this section.

1. Difficulties intrinsic in the current projection framework. LTI MOR techniques generally are formulated within a linear projection framework. However, it may not be efficient/reasonable to use linear projection for nonlinear models, as it leads to less computational speedup and lacks enough theoretical support, compared to LTI cases. It is worthwhile to investigate how nonlinear projections may work better than linear projections, and that casts even more questions than answers regarding nonlinear model order reduction.

2. Lack of canonical representation of a nonlinear model. LTI models (linear differential equations) $C\frac{d}{dt}x + Gx + Bu = 0$ have a very special and compact matrix representation (C, G, B). There are also other equivalent canonical representations such as controllable/observable canonical forms [72, 73]. However, there have been hardly any

literature on canonical forms of general nonlinear systems. The nonlinear functions can be arbitrary and complicated, making the system hard to analyze and manipulate, and even harder to reduce.

3. Lack of characterization of system responses. LTI systems can be characterized by their transfer functions which are extensively used as a performance metric to preserve in linear model order reduction. However, for nonlinear systems, there is no simple way to derive analytical expressions of system responses such as harmonic distortion, output frequency, *etc.*. This further complicates the reduction problem.

1.3 Dissertation Contributions and Overview

In this dissertation, we formulate the model order reduction problem, explore and develop nonlinear model order reduction techniques, and apply these techniques in practical examples. In particular, the main contributions are as follows:

1. We formulate the nonlinear model order reduction problem as an optimization problem and present a general nonlinear projection framework that encompasses previous linear projection-based techniques as well as the techniques developed in this dissertation.

2. We develop a nonlinear model order reduction technique, ManiMOR, which is a direct implementation of the nonlinear projection framework. It generates a nonlinear reduced model by projection on a general-purpose nonlinear manifold. The proposed manifold can be proven to capture important system dynamics such as DC and AC responses. We develop numerical methods that alleviates the computational cost of the reduced model which is otherwise too expensive to make the reduced order model of any value compared to the full model.

3. We develop a nonlinear model order reduction technique, QLMOR, which transforms the full model to a canonical QLDAE representation and performs Volterra analysis to derive a reduced model. We develop an algorithm that can mechanically transform a set of nonlinear differential equations to another set of equivalent nonlinear differential equations that involve only quadratic terms of state variables, and therefore it avoids any problem brought by previous Taylor-expansion-based methods. With the QLDAE representation, we develop the corresponding model order reduction algorithm that extends and generalizes previously-developed Volterra-based technique.

4. We develop a nonlinear phase/timing macromodeling technique, NTIM, that derives a macromodel that specifically capture timing/phase responses of a nonlinear system. We rigorously define the phase response for a non-autonomous system, and derive the dynamics of the phase response. The macromodel emerges as a scalar, nonlinear time-varying differential equation that can be computed by performing Floquet analysis of

the full model. With the theory developed, we also present efficient numerical methods to compute the macromodel.

5. We develop a finite state machine abstraction technique, DAE2FSM, that learns a Mealy machine from a set of differential equations from its input-output trajectories. The algorithm explores the state space in a smart way so that it can identify the underlying finite state machine using very few information about input-output trajectories.

The remainder of the dissertation is organized as follows. Chapter 2 formulates the MOR problem, reviews previous work and summarizes challenges in nonlinear MOR. Chapter 3 provides a deeper view of the linear projection framework that is used by most previous MOR methods. Chapter 4 generalizes the linear projection framework to a nonlinear one, and summarizes the skeleton of an algorithm within this framework. Chapter 5 presents the method ManiMOR that is a direct implementation of the nonlinear projection framework. Chapter 6 presents the method QLMOR that generates a quadratic-linear reduced order model. Chapter 7 presents the method NTIM that generates a phase/timing reduced model. Both QLMOR and NTIM can be interpreted in the nonlinear projection framework. Chapter 8 looks at a slightly different problem of deriving a finite state machine model from a set of differential equations, and presents the method DAE2FSM. Chapter 9 concludes the dissertation and discusses the related future work.

1.4 Notations and Abbreviations

Except for special mention, we will use use the following notations and abbreviations throughout the dissertation.

Table 1.3: Mathematical notation

$\mathbb{R}^{M\times N}$	sets of real matrices of size M by N
$\mathbb{R}^{N}$	sets of real matrices of size M by N
x	state variables in the full model
$\dot{x}$	time derivative of x
x_r	state variables in the reduced model
u	inputs
y	outputs
$f(\cdot)$	nonlinear functions in the full model
$f_r(\cdot)$	nonlinear functions in the reduced model
$f'(\cdot)$	Jacobian of function f
r	residual of the differential equations
e_i	the column vector $[0,\cdots,0,1,0,\cdots,0]$ with the i-th element being 1 and other elements being 0
$H(s)$	transfer function of an LTI system
$\text{span}(A)$	column space of matrix A
$\text{span}(v_1,\cdots,v_N)$	span of vectors $v_1,\cdots,v_N$
$\otimes$	Kronecker product

Table 1.4: Abbreviations

DAEs	differential algebra equations
FSM	finite state machine
LTI	linear, time-invariant
LTV	linear, time-varying
LUT	look-up tables
LPTV	linear periodic time-varying
MIMO	multiple-input multiple-output
MOR	model order reduction
MSE	mean squared error
ODEs	ordinary differential equations
PDEs	partial differential equations
POD	proper orthogonal decomposition
PWL	piecewise linear
SIAO	single-input all-output
SISO	single-input single-output
SVD	singular value decomposition
TBR	truncated balanced realization
TPWL	trajectory piecewise linear

Chapter 2

Background and Previous Work

In this chapter, we formally formulate the model order reduction problem, trying to encompass all previous methods under the same formulation. We review previous linear and nonlinear model order reduction techniques, and summarize a few challenges in nonlinear MOR, some of which are addressed in this dissertation.

2.1 Problem Formulation

Suppose that we are given a full model $M \in \mathcal{M}$, where $\mathcal{M}$ is the set of a specific type of model, (*e.g.*, ordinary differential equations). The goal of model order reduction is to derive a reduced model $M_r \in \mathcal{M}_r$, where $\mathcal{M}_r$ is the set of another type of models (*e.g.*, ordinary differential equations), so that M_r approximates M in some sense.

Mathematically, we can define the problem of computing M_r from M as the following optimization problem

$$\begin{aligned} \underset{M_r}{\text{minimize}} \quad & F(M, M_r) \\ \text{subject to} \quad & G(M, M_r) = 0, \\ & H(M, M_r) \leq 0, \end{aligned} \tag{2.1}$$

where $F(\cdot,\cdot)$, $G(\cdot,\cdot)$, $H(\cdot,\cdot)$ are functions on models M and M_r. $F(\cdot,\cdot)$ specifies how close two models are. $G(\cdot,\cdot)$ and $H(\cdot,\cdot)$ specify the equality and inequality constraints.

Choices of $\mathcal{M}$, $\mathcal{M}_r$, $F(\cdot,\cdot)$, $G(\cdot,\cdot)$ and $H(\cdot,\cdot)$ lead to different problems. Usually, if M_r has less number of state variables (*i.e.*, the order of the system) than M, it is called a reduced order model, and M_r is expected to be computationally more efficient than M.

2.1.1 Types of Models

As listed in Table 1.1, there are many types of mathematical models. However, in the MOR problem, not all pairs of models $(\mathcal{M}, \mathcal{M}_r)$ are reasonable. For example, it may be

unreasonable to reduce an LTI model to a nonlinear model because they may exhibit fundamentally different behaviors; it may also be unreasonable to convert a deterministic model to a non-deterministic one and vise versa.

In this dissertation, unless otherwise mentioned, we focus on the problem where $\mathcal{M}$ and $\mathcal{M}_r$ are both linear ODEs/DAEs or nonlinear ODEs/DAEs.

We list possible model reduction problems with other choices of $\mathcal{M}$ and $\mathcal{M}_r$ in Table 2.1.

Table 2.1: Choices of mathematical models in MOR

Original Model $\mathcal{M}$	Reduced Model $\mathcal{M}_r$	Applications	References
ODEs/DAEs	ODEs/DAEs	Traditional MOR	[74–78]
ODEs/DAEs	FSMs	FSM abstraction/learning	[78–80]
ODEs/DAEs	Hybrid Systems	Hybridization	[79, 81, 82]
Hybrid Systems	FSMs	FSM abstraction/learning	[83, 84]
ODEs/DAEs	Difference Equations	Time Discretization	[85]
PDE	ODE/DAE	Time/Space Discretization	[85]

Remark In Table 2.1, the mappings $\mathcal{M} \to \mathcal{M}_r$ can be one-to-one, one-to-many, many-to-one or many-to-many. For example, for continuous ODEs, we can apply different integration/discretization schemes to obtain different difference equation models. Another example, widely used in IC design, is to map a finite state machine to a circuit implementation which corresponds to a set of ODEs/DAEs. With different circuit topologies, we obtain different ODEs/DAEs that realize the same FSM functionality. It is sometimes also interesting to look at the reverse problem – to map $\mathcal{M}_r$ to $\mathcal{M}$. For example, the mapping from FSMs to ODEs may correspond to the synthesis problem [86–88].

2.1.2 Model Fidelity

In (2.1), $F(\cdot, \cdot)$ defines a measure describing how close two models are, or how well M_r approximates M.

For an input-output system whose inputs are u and outputs are y, a straightforward choice of $F(\cdot, \cdot)$ is

$$F(M, M_r) \triangleq \sum_{u \in \mathcal{U}} ||y - y_r||, \tag{2.2}$$

where $\mathcal{U}$ is a given set of input trajectories/traces; y and y_r are outputs of M and M_r under inputs u, respectively. In general, y and u may be described in different forms, depending on applications.

For example, since LTI systems are typically described by their transfer functions, y can be chosen as the transfer function and u can vary over a range of frequencies. For nonlinear systems, often u is chosen as a set of (training) input signals, and y is chosen to be the

corresponding time-domain responses, or the first few harmonic distortions when the system is settled at the periodic steady state. In specific applications where other special responses or higher-level performance metrics (such as phase/timing responses or bit error rate) are of interest, y can also be set to these quantities.

There have also been many techniques in reducing dynamical systems with no inputs (*e.g.*, the ODE model $\dot{x} = f(x)$). In this case, we may still view it as an input-output system, with the virtual input to the system being the set of all possible impulse functions that set the initial condition of the system when applied. These methods basically compute reduced order models that approximate outputs of full models for arbitrary initial conditions.

$G(\cdot,\cdot)$ and $H(\cdot,\cdot)$ in (2.1) impose equality and inequality constraints on the reduced order model, respectively. They are usually used to guarantee certain important behaviors of reduced models.

For example, in moment-matching methods [89] for LTI systems, $G(\cdot,\cdot)$ is defined to be

$$m_i(M) = m_{r,i}(M_r), \tag{2.3}$$

where $m_i(M)$ is the i-th moment of the transfer function of model M (see Section 2.3.2 for details). This guarantees that the first few moments of the reduced model $m_{r,i}$ match those of the full model m_i.

In stability-guaranteed reduced-order modeling techniques [90,91], $H(\cdot,\cdot)$ may be defined as a linear matrix inequality

$$P_r A_r + A_r^T P_r \preceq 0 \tag{2.4}$$

where A_r is the matrix in the reduced model $\frac{d}{dt}x = A_r x$ and P_r is a positive definite matrix. To further guarantee the passivity of the reduced model, we may simply expand $G(\cdot,\cdot)$ by one more equality constraint [91].

While there are many fidelity metrics that prove to be useful in different applications, we try to summarize the most important ones that reduced models should satisfy, as follows:

1. Attractor preservation. Attractors, such as equilibria (DC states), limit cycles, or even strange attractors, are system responses as time t goes to infinity. Many engineering systems are designed to work in a region close to its DC states or limit cycles, and therefore the attractors are the most important system response to preserve in the reduced model.

   Nonlinear systems present much richer attractors than linear systems, such as multiple DC equilibria and limit cycles. They correspond to important circuits such as memory cells and oscillators.

2. Stability preservation. Stability describes how the state of a dynamical system moves under small perturbation at the input. It is a key system property of interest, and has many variants of definitions [73, 90, 91]. The stability (*e.g.*, of equilibria) should

be preserved in the reduced model since it ensures that the qualitative behavior of the system is correct, given that the attractors are preserved in the model.

3. Passivity preservation. Passivity of a system refers to the property that the system only consumes energy and never produces energy. This is commonly seen in practical systems, such as RLC interconnects. This is another qualitative behavior of a system, and is desirably preserved in reduced models.

4. Structure preservation. Reduced models are often written as a set of dense (in terms of Jacobian matrices) and less structured differential equations. However, full models usually have certain structures. For example, the system matrices may be sparse, symmetric or have block patterns; the system may be composed of a cascade of several sub-systems, or a feedback; the system may be be a second order system; the system may obey certain conservation laws; the system may correspond to a realistic circuit implementation. Therefore, it is sometimes desirable for reduced models to preserve these structures, even with certain accuracy loss.

5. Linearity preservation. This refers to the property that the reduced model of a linear system should remain linear. This is rather a necessary condition that should be appreciated in practice, especially for nonlinear MOR techniques. Linear systems have limited behaviors than nonlinear systems, and should not exhibit any nonlinear effects induced by model reduction methods.

   Example 2.1.1 *The TPWL [2] reduced model, reviewed in Section 2.4.3, for the ODE model (1.2) is*

$$\frac{d}{dt}x_r = \sum_i w_i(x_r)(V^T f(x_i) + V^T G(x_i)V(x_r - x_{r,i})). \tag{2.5}$$

   When $f(x) = Gx$, i.e., the full model is linear, then (2.5) becomes

$$\frac{d}{dt}x_r = \sum_i w_i(x_r)(V^T G x_i + V^T G V(x_r - x_{r,i})). \tag{2.6}$$

   Because $Vx_{r,i} = x_i$ does not always hold, the TPWL model (2.6) of an LTI model becomes nonlinear.

6. Composability. One application of reduced models is to replace complex system blocks by their reduced models so that simulation/validation of larger systems can be computationally more efficient. When the full models are replaced by reduced models and these reduced models are composed together, we hope to retain the coupling effects, and "important" behaviors (*e.g.*, stability) of the composed system. Composability

also allows one to generate reduced models of sub-systems separately and obtain reduced models of larger sub-systems by composition. It can lead to a hierarchical model order reduction framework.

7. Identification of simpler models under coordinate transformation. A nonlinear system, in some cases, can be converted to a simpler model (*e.g.*, with less nonlinearity), or even to an LTI model. In such cases, the nonlinear MOR problem may be reduced to a linear MOR problem, and therefore becomes much easier. One such example is as follows:

   Example 2.1.2 ("True" Linearization of Nonlinear Models) *Consider the nonlinear system*

$$\begin{aligned} \dot{x_1} &= -x_1 + u, \\ \dot{x_2} &= -\frac{x_2}{x_1}u, \end{aligned} \tag{2.7}$$

   where x_1, x_2 are state variables, and u is the input. Define $x_3 = x_1x_2$, we obtain by chain rule the differential equation for x_3. Thus, we obtain a set of differential equations in terms of x_1 and x_3, and they are linear.

$$\begin{aligned} \dot{x_1} &= -x_1 + u, \\ \dot{x_3} &= -x_3. \end{aligned} \tag{2.8}$$

   This simple example shows the possibility of finding appropriate coordinate transformations to simplify nonlinear models.

   Such capability is desirable in a nonlinear MOR technique.

8. Separation of independent/repetitive sub-systems. There are many cases where the full model is composed of several independent sub-systems. Some sub-systems may be redundant in terms of controllability, observability, repetition, *etc.*. It is therefore another desirable property for MOR techniques to separate these sub-systems automatically.

   Example 2.1.3 *Fig. 2.1, Fig. 2.2 and Fig. 2.3 show three special examples where the full model is composed of two sub-systems.*

   (a) Shown in Fig. 2.1, the output y depends solely on the sub-system A. Therefore sub-system B is redundant from an input-output perspective, and should be identified and removed in model order reduction.

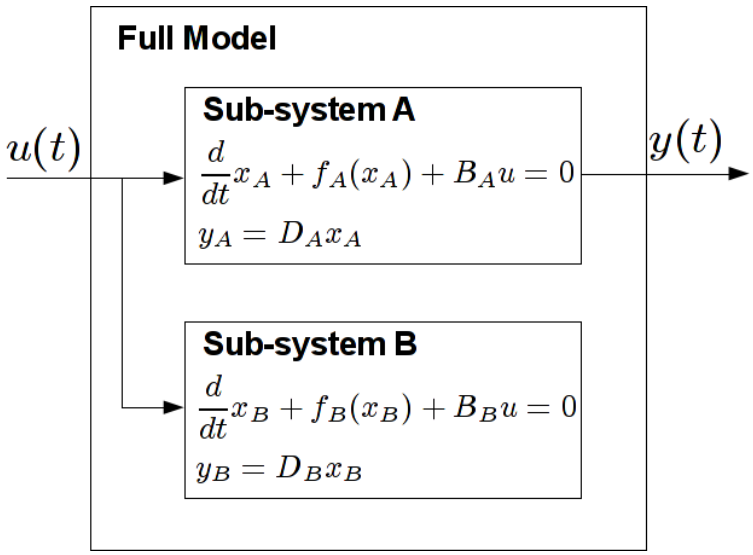


Figure 2.1: A model consisting of two separate sub-systems.

(b) *Shown in Fig. 2.2, the full system is composed of a nonlinear sub-system A and a linear sub-system B. In this case, it would be desirable to find a meaningful separation (e.g., minimizing the number of coupling variables) so that two sub-systems can be reduced separately and lead to "good" reduced models after composition.*

(c) *Shown in Fig. 2.3, the two sub-systems A and B are identical. (e.g., circuit blocks, such as inverter chains or memory cells, are often repeated.) It would be great if model order reduction techniques can identify these identical blocks, based on which we may simply reduce one sub-system, and repeat the reduced sub-system twice to get the full reduced model. This way, we also preserve the repetitive structure in the original full model.*

9. Lower computational cost. It is well recognized that in many cases, even if the order of the model is reduced, the computational cost is not reduced, and may even be higher than the original full model, making the reduced order model of little use. This is especially more likely to happen in nonlinear model order reduction.

10. Sparsity preservation. Full models of physical systems are normally sparse – in terms of linear systems, it means that the matrices in the model are sparse; in terms of nonlinear systems, it means that the Jacobian matrices of nonlinear functions are sparse. In contrast, the reduced models are typically dense. However, if sparsity can be preserved by sacrificing little accuracy, the reduced model may inherit certain structures in the full model, and the computational cost of the model can also be

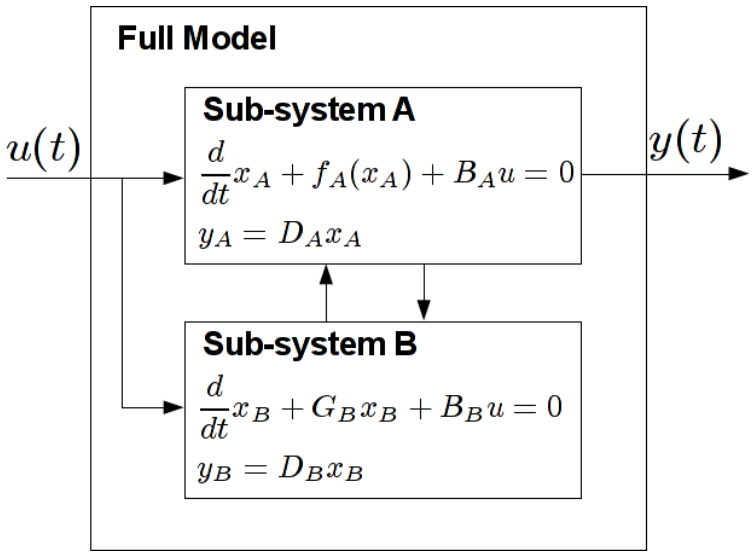


Figure 2.2: A model consisting of a nonlinear sub-system and a linear sub-system.

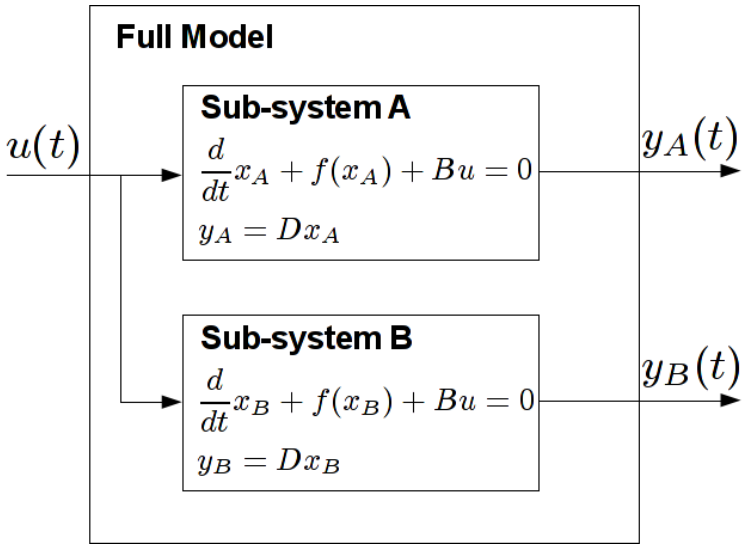


Figure 2.3: A model consisting of two identical sub-systems.

reduced due to sparsity. Also, a sparse projection may lead to reduced models whose state variables are associated with physical quantities.

Summarizing the above discussion, in Table 2.2[1], we give a relatively complete list of the model fidelity that has been, or potentially can be, used in model order reduction.

Table 2.2: Model fidelity and examples

Model Fidelity	Example in terms of $F(\cdot,\cdot)$, $G(\cdot,\cdot)$ and $H(\cdot,\cdot)$
Moment matching	$G(M, M_r) = m_i(M) - m_{r,i}(M) = 0$
Minimum output-MSE	$F(M, M_r) = \sum_{u\in\mathcal{U}} \|\|y(t) - y_r(t)\|\|_2$
Attractor preservation	$G(M, M_r) = \lim_{t\to\infty}(y(t) - y_r(t)) = 0$
Stability preservation	$H(M, M_r) = P_r A_r + A_r^T P_r \preceq 0$
Passivity preservation	$G(M, M_r) = P_r B - D = 0$ with stability condition
Structure preservation	$G(M, M_r)$: M_r corresponds to a realizable RLC circuit.
Linearity preservation	$G(M, M_r)$: M_r is linear, given M is linear.
Composability	$G(M, M_r)$ and/or $H(M, M_r)$: suppose $M = (M_A, M_B)$, $M_r = (M_{r,A}, M_{r,B})$, then M_r preserves certain properties of M.
Redundancy removal	M_r removes non-observable and non-controllable sub-systems.
Lower computational cost	$H(M, M_r)$: Computational cost of M_r is less than that of M.
Sparsity preservation	$F(M, M_r)$: number of non-zero elements in Jacobians in M_r.

In Table 2.3 in Section 2.5, we also summarize a selected set of previous model order reduction techniques in terms of the optimization problem (2.1), *i.e.*, in terms of the functions $F(\cdot,\cdot)$, $G(\cdot,\cdot)$ and $H(\cdot,\cdot)$.

2.1.3 Model Verification

In (2.1), if $F(\cdot,\cdot)$ is null, the problem is called the *feasibility problem* in literature. It can be viewed as a verification problem to check whether two models M and M_r are equivalent (under the constraints imposed by $G(\cdot,\cdot)$ and $H(\cdot,\cdot)$).

The solution of this problem renders a model M_r that satisfies all the constraints $G(\cdot,\cdot)$ and $H(\cdot,\cdot)$. If the (perfect) algorithm cannot find such a model, then such an equivalent model $M_r \in \mathcal{M}_r$ does not exist.

This problem has much wider engineering applications. For example, in the digital circuit synthesis and verification flow, abstraction/models of different levels are verified against each other (equivalence checking) to make sure that the abstraction/synthesized circuit is

[1]There are many choices of constraints that may lead to the same fidelity criteria, but we only provide one example for each to illustrate the idea. The symbols used in the table are straightforward and therefore without detailed explanation.

functionally correct. Solving such problems is therefore the key enabler of the digital circuit design flow.

However, in model order reduction literature, this equivalence checking procedure is typically skipped, or is done by heuristics. We believe that this is an important problem to solve for employing MOR in practice, and some initial studies [79, 92–96] in analog/mixed-signal verification may be used.

2.2 MOR Overview

According to the description of the full system, model order reduction methods can be classified into black-box methods and white-box methods.

In black-box methods, the internal structure or representation of the model is not accessible. Rather, we have restricted access to only a few controllable inputs (*e.g.*, stimulations) and observable outputs (*e.g.*, from measurements or simulations). This leads to various "fitting" algorithms, including regression, neural network, machine learning, *etc.*. These methods are often also called *system identification* methods. Due to the nature of such methods, we normally cannot assert any deterministic relationship between the full model and the reduced model beyond the training data.

In white-box methods, we have full access to the model. This gives us a complete view of the full model, and thus opportunities to guarantee certain properties can be preserved from full models. Depending on the type of the full model, the behavior can be qualitatively different, and therefore different methods have been proposed to handle different types of models, including LTI, LTV and nonlinear models. The most common framework encompassing most of these methods is the linear projection framework, to be introduced in this section, and detailed in Chapter 3.

2.2.1 System Identification

System identification methods [97,98] view the system as a black/grey box. Usually only limited number of input-output data/trajectory is given, either by simulation or measurement. Sometimes, partial information of the system is known (such as the connectivity of a graph). These methods can be classified into two categories: parametric methods and non-parametric methods.

In parametric methods, we make assumptions of the model structure for the system, and parameterize the model by certain key parameters. With the parameterized model, the data are used to fit the model by solving for parameters that minimizes the error between the fitted model and the data. For example, we may use methods such as least squares, maximum likelihood, maximum a posteriori probability estimation, compressed sensing, *etc.*.

In non-parametric methods, we construct the model directly from data. For example, to find an LTI model of a system, we can measure directly the impulse response or step response

to reconstruct the LTI system. To compute the transfer function at specific frequency points, we can measure directly the response to a sinusoidal input. Often the experiments are carefully designed for the identification purpose in non-parametric methods.

2.2.2 Linear Projection Framework

In the linear projection framework, we consider the problem of reducing a model $M \in \mathcal{M}$ that is a set of differential algebraic equations

$$\frac{d}{dt}q(x) + f(x) + b(t) = 0, \tag{2.9}$$

to a reduced model $M_r \in \mathcal{M}_r$ that is another set of differential algebraic equations

$$\frac{d}{dt}q_r(x_r) + f_r(x_r) + b_r(t) = 0, \tag{2.10}$$

by projection.

That is, the nonlinear functions in (2.10) are defined by projection, *i.e.*,

$$\begin{aligned} q_r(x_r) &= W^T q(V x_r), \\ f_r(x_r) &= W^T f(V x_r), \\ b_r(t) &= W^T b(t), \end{aligned} \tag{2.11}$$

where $V, W \in \mathbb{R}^{N \times N_r}$ are two projection matrices.

Therefore, the problem of MOR amounts to finding two appropriate projection matrices.

2.3 MOR for Linear Time-Invariant Systems

For an LTI system, $q(x) = Cx$ and $f(x) = Gx$ are linear functions of x, and often the input is written in the form of $b(t) = Bu(t)$, *i.e.*

$$C\frac{d}{dt}x + Gx + Bu = 0, \quad y = D^T x, \tag{2.12}$$

where $C \in \mathbb{R}^{N \times N}$, $G \in \mathbb{R}^{N \times N}$, $B \in \mathbb{R}^{N \times N_i}$, $D \in \mathbb{R}^{N \times N_o}$, N_i is the number of inputs and N_o is the number of outputs.

LTI reduced models are therefore in the form of

$$C_r\frac{d}{dt}x_r + G_r x_r + B_r u = 0, \quad y = D_r^T x_r, \tag{2.13}$$

where $C_r \in \mathbb{R}^{N_r \times N_r}$, $G_r \in \mathbb{R}^{N_r \times N_r}$, $B_r \in \mathbb{R}^{N_r \times N_i}$, $D_r \in \mathbb{R}^{N_r \times N_o}$, N_r is the order of the

reduced model.

2.3.1 Transfer Function Fitting

Since LTI models can be equivalently defined by their transfer functions $H(s)$, a straightforward model order reduction method is to fit a reduced transfer function $H_r(s)$ that gives a good approximation to $H(s)$.

The famous approach is the Padè approximation [99], *i.e.*, to find a rational reduced transfer function of the form

$$H_r(s) = \frac{a_0 + a_1 s + a_2 s^2 + \cdots + a_m s^m}{b_0 + b_1 s + b_2 s^2 + \cdots + b_n s^n} \tag{2.14}$$

which satisfies

$$\begin{aligned} H_r(0) &= H(0), \\ \frac{d^k}{ds^k} H_r(0) &= \frac{d^k}{ds^k} H(0), \quad k = 1, 2, \cdots, m+n, \end{aligned} \tag{2.15}$$

where $\frac{d^k}{ds^k} H(0)$ is also referred to as the k-th moment of the transfer function. Therefore, any LTI MOR method that leads to a reduced model satisfying (2.15) is also called a **moment-matching** method.

Intuitively, Padè approximation is good for MOR because a rational function is a good fit for LTI transfer functions – the original transfer function $H(s)$ is typically a rational function given that there is no repeating poles. In fact, we may analytically write out $H(s)$ of (2.12) as

$$H(s) = D^T (sC + G)^{-1} B, \tag{2.16}$$

which is a rational function in s if the matrix $G^{-1}C$ has no repeating eigenvalues.

There have been a series of work developed based on this idea, among which two major ones are AWE [100] and PVL [101]. AWE [100] explicitly computes the moments and finds a Padè approximation $H_r(s)$ with $m = n-1$ that matches these moments. PVL [101] avoids numerical inaccuracy in the explicit moment computation in AWE, and uses the Lanczos process to compute the Padè approximation of the transfer function.

2.3.2 Krylov-Subspace Methods

The explicit moment calculation in the Padè approximation methods can have serious numerical inaccuracy, and this inaccuracy will be inherited to the reduced model. Even with PVL, since it uses a transfer function representation of the LTI system, it is not directly clear what the state-space ODE/DAE model of the reduced model is.

The source of error in explicit moment computation comes from the fact that higher-order moments tend to decay/grow at an exponential rate. To see that, we may derive the expression of the i-th moment to be

$$m_i = D^T (G^{-1}C)^i B. \tag{2.17}$$

Therefore, as i becomes large, m_i depends approximately only on the eigenvalue of $G^{-1}C$ with the largest absolute value, and therefore will decay/grow exponentially, leading to significant numerical inaccuracy.

Krylov-subspace methods [89] avoids the explicit moment calculation. It turns out that if we choose V and W such that

$$\begin{aligned} \text{span}(V) &= \mathcal{K}_{N_r}(G^{-1}C, B) = \text{span}(G^{-1}CB, (G^{-1}C)^2 B, \cdots, (G^{-1}C)^{N_r} B, \\ \text{span}(W) &= \mathcal{K}_{N_r}((G^{-1}C)^T, D) = \text{span}((G^{-1}C)^T D, ((G^{-1}C)^T)^2 B, \cdots, ((G^{-1}C)^T)^{N_r} D), \end{aligned} \tag{2.18}$$

then the projected reduced models with $C_r = W^T CV$, $G_r = W^T GV$, $B_r = W^T B$, $D_r = V^T D$ have the property that the first $2N_r$ moments of the reduced transfer function match those of the full transfer function.

Moreover, the projection matrices V and W can be efficiently and accurately computed (*e.g.*, Arnoldi and Lanczos algorithms [102, 103]). These algorithms are computationally efficient and numerically stable even for large scale problems (*e.g.*, sparse matrices of dimension on the order of millions), and therefore enables the moment-matching technique to be applicable to large-scale problems.

2.3.3 Truncated Balanced Realization

The method of truncated balanced realization (TBR) originated from control theory, based on the idea of controllability and observability [104]. Loosely speaking, certain states in a dynamical system are hard to control and some others are hard to observe. The balanced realization obtains a dynamical system whose state variables have equal controllability and observability. Then, the truncation of the states that are hard to control and observe leads to a reduced order model.

It turns out that the TBR method also falls in the linear projection framework. Ultimately, it defines two matrices V and W, and the reduced model is a projected model.

2.3.4 Proper Orthogonal Decomposition

Proper orthogonal decomposition (POD) is also a method that derives reduced models by linear projection. It takes several data/trajectories of the state variables, and constructs a linear subspace that minimizes certain error between the projected reduced model and the

full model.

On the one hand, POD is general because it can take any trajectory of state variables. Therefore, prior information of input signals and system dynamics can help generating a better reduced order model. In can also be shown that if the trajectory is chosen to be the impulse response of an LTI system, then one version of the POD is equivalent to the TBR method.

On the other hand, POD is less systematic than other methods in the sense that there is no universal guideline in choosing a "good" trajectory. Therefore, it is possible to have "bad" trajectories that lead to "bad" reduced order models.

2.4 MOR for Nonlinear Systems

Nonlinear MOR methods are mostly built upon the LTI MOR methods. For so-called weakly-nonlinear systems, we may approximate the nonlinear model by a local model, such as LTI, LTV, Volterra models. The reduction is then performed on the approximated model to obtain an approximated reduced model. For general nonlinear systems, we may approximate the nonlinear model around several different regions in the state space, and reduce each linearized model using LTI MOR methods.

2.4.1 Linear Time-Varying Approximation and MOR

There are many practical systems that may be well-approximated by linear time-varying systems, partly because these systems are engineered to work under time-varying operating points. For example, in RF circuits such as mixers, the local frequency input is fixed, rendering an approximate periodic time-varying system. Similarly, in switched-capacitor circuits, the clock input is fixed, and the circuit is approximately working periodically under two modes (clock high and clock low).

The MOR methods in [25, 28] extends the notion of transfer function for LTI system to time-varying transfer functions. Then it can be shown that using a Padè-like approximation, or an appropriate projection, we can obtain reduced order models whose time-varying transfer functions approximate the original one.

2.4.2 Volterra-Based Methods

Generally, for a nonlinear system, we may derive a Volterra series approximation [105] which approximates its local behavior. Therefore, instead of matching the moments of transfer functions, we may match the moments of Volterra kernels (higher-order transfer functions), and this leads to the Volterra-based MOR methods such as NORM [106] and [107]. Since the Volterra series only converges when input is small, Volterra-based models are "small-signal" or "local" models.

Similar to Krylov subspace methods, in this case, we may also choose projection matrices V and W that cover a series of Krylov subspaces so that the projected models match the specified moments of Volterra kernels.

2.4.3 Trajectory-Based Methods

Unlike Volterra-based methods, trajectory-based MOR methods generate "global" reduced models, instead of "local" ones.

Among several variants of trajectory-based methods, the representative one is the TPWL method [2] whose algorithm is shown in Algorithm 1. The TPWL method identifies two linear subspaces V and W, and generates the reduced model by linear projection. The subspaces V and W are the aggregated subspaces of many reduced models of linearized models (*e.g.*, Krylov subspaces of linearized models), therefore making sure that the moments of transfer functions of reduced linearized models match those of the full linearized models. In order to achieve the computational efficiency of the reduced model, *i.e.*, the computation of the term $W^T f(Vz)$, it uses a weighted-summation of linear functions to approximate $W^T f(Vz)$.

Algorithm 1 TPWL (Trajectory Piece-Wise Linear) Model Order Reduction

Inputs: Differential equations of the full model $\frac{d}{dt}q(x) + f(x) + Bu = 0$.
Outputs: Two linear subspaces defined by the column span of V and W. Differential equations of the reduced model $\frac{d}{dt}q_r(x_r) + f_r(x_r) + B_r u = 0$. where $q_r(x_r) = \sum_{i=1}^{N_s} w_i(x_r)(W^T q(x_i) + W^T C(x_i)V(x_r - x_{r,i}))$, $f_r(x_r) = \sum_{i=1}^{N_s} w_i(x_r)(W^T f(x_i) + W^T G(x_i)V(x_r - x_{r,i}))$, $B_r = W^T B$, $w_i(x_r)$ are weighting functions described in Section 2.4.3, N_s is the number of sample points.

1: Simulate the full model using a set of "training" input signals, and obtain a set of trajectories.
2: Sample N_s points $\{x_i\}, i = 1, \cdots, N_s$ on these trajectories as expansion points of the reduced model.
3: **for all** $i = 1, \cdots, N_s$ **do**
4: Linearize the full model around each sample point x_i
5: Compute the reduced linearized models, and obtain projection matrices V_i and W_i for each reduced model.
6: **end for**
7: Compute the projection matrix for the nonlinear model $V = [V_1, x_1, \cdots, V_k, x_k]$, $W = [W_1, x_1, \cdots, W_k, x_k]$.
8: Compute $W^T f(x_i)$, $W^T G(x_i)V$, $W^T q(x_i)$, $W^T C(x_i)V$ and $W^T B$, and store them in the reduced model.

2.5 Summary of Previous Work

We summarize a selected set of methods in Table 2.3, relating them with our formulation (2.1).

Table 2.3: Summary of previous MOR methods in terms of the optimization problem

Methods	$F(\cdot,\cdot)$	$G(\cdot,\cdot)$	$H(\cdot,\cdot)$
AWE [100]	null	moment-matching $m_i(M) = m_{r,i}(M_r)$	null
PVL [101]	null	moment-matching $m_i(M) = m_{r,i}(M_r)$	null
PRIMA [74]	null	moment-matching $m_i(M) = m_{r,i}(M_r)$	null
		the full model is the MNA equation and is passive	
TBR [104]	null	controllability and observability are balanced	null
NORM [106]	null	moment-matching of Volterra kernels	null
TPWL [2]	null	moment-matching of linearized models	null

2.6 Major Challenges of Nonlinear MOR

Rethinking about the nonlinear MOR problem, we summarize the following major challenges:

1. Lack of canonical form for the reduced model. LTI systems can be completely represented by several matrices C, G, B, D. However, nonlinear systems generally do not have a universal canonical form, making it hard to efficiently represent a nonlinear system as well as to develop algorithms manipulating the full model.

2. Lack of analytical expressions for system responses. Unlike LTI systems which are characterized by transfer functions, nonlinear systems generally do not have such explicit analytical formula for system responses. Volterra kernels of polynomial systems alleviate this problem.

3. Limited guarantee of reduced order model behaviors. Often we are only able to quantify the behavior of the reduced model locally or with respect to given training data. However, global properties such as stability and multiple equilibria are extremely important for nonlinear systems.

4. Potentially expensive computational cost. In linear projection-based methods, the nonlinear function in the reduced model $W^T f(Vx_r)$ can be as expensive to compute as the full model, thus making the reduced model of less use compared to the full model.

2.7 Benchmark Examples

We have created and collected a few benchmark systems for testing MOR methods. Some of them are copied from existing literatures. A summary of these benchmarks is shown in Table 2.4. Detailed description of these benchmarks are presented in Appendix Chapter B.

Table 2.4: Benchmarks for MOR

Benchmark	Size	Description
NTL1	variable	nonlinear transmission line circuit [2]
NTL2	variable	nonlinear transmission line circuit (quadratic model) [2]
INVC	variable	inverter chain circuit
NEURON_ML	2	Morris-Lecar neuron model
NEURON_FN	variable	FitzHugh-Nagumo neuron model
LATCH1	3	latch circuit 1

Most of these examples are from circuit applications. Some of them describe the chemical reactions, biological pathways, or neuron dynamics. We briefly review the circuit MNA equations and chemical rate equations.

2.7.1 Circuit MNA Equations

In circuit simulation, the MNA equation formulation [108, 109] is most widely used. Briefly, in the MNA equations, the state variables (unknowns) are mainly the node voltages, together with a few branch currents of non-voltage-controlled devices. The differential equations mainly consist of KCL equations, *i.e.*, each equation defines the sum of currents flowing into one node to be 0. For non-voltage-controlled devices, the BCR equations are added to describe the voltage-current relationship of these devices.

2.7.2 Chemical Rate Equations

Chemical kinetics can be modeled by a set of differential equations, called rate equations. They describe the dynamics/rate-of-change of concentrations of reactants. They are usually in the form of ODEs $\frac{d}{dt}x = f(x, u)$, where x are the concentrations of reactants, and u are the controllable parameters such as reaction rates, temperatures.

There are **rate laws** that determine the function $f(x, u)$. For a generic reaction $A+B \xrightarrow{K} C$, the f function has expressions such as $K[A]^m[B]^n$, where K is a constant, and $[A]$, $[B]$ are the concentrations of A and B, respectively. Some other rate laws include models of Henri, Michaelis & Menten, and Briggs Haldane [110]. For example, in Michaelis & Menten model, the f function has expressions such as $\frac{x}{k+x}$.

Chapter 3

Linear Projection Framework for Model Order Reduction

While many previous MOR methods have used the linear projection framework, the major motivation seems to be the fact that the projection of the full model on certain subspaces leads to a reduced model that matches certain accuracy metrics (*e.g.*, moments or Hankel singular values). However, no literature has discussed why such such a framework is a good one for MOR. For example, some key questions that need to be answered are: can any reduced model be written as a projected model; what is the degree of freedom in choosing projection matrices; what might be other choices of projections; *etc.*.

In this chapter, we take a deeper look at the linear projection framework. We discuss the application of the linear projection framework in LTI MOR – the representation power of the linear projection framework, the degree of freedom in choosing projection matrices, the other potential criteria in choosing projection matrices. We also discuss the application of the linear projection framework in nonlinear MOR, and discuss potential problems and solutions.

3.1 Linear Projection Framework

The basic idea of the linear projection work has already been presented in Section 2.2.2. Here, we focus on the algorithmic aspect of the linear projection framework. The problem is to reduce a model M,

$$\frac{d}{dt}q(x) + f(x) + b(t) = 0, \tag{3.1}$$

to a reduced model M_r,

$$\frac{d}{dt}q_r(x_r) + f_r(x_r) + b_r(t) = 0. \tag{3.2}$$

The algorithm is generally in the form of Algorithm 2. Obviously, the choice of V and W plays a key role in the linear projection framework.

Algorithm 2 MOR based on Linear Projection

Inputs: $\frac{d}{dt}q(x) + f(x) + b(t) = 0$
Outputs: $\frac{d}{dt}q_r(x_r) + f_r(x_r) + b_r(t) = 0$
1: Construct a matrix $V \in \mathbb{R}^{N\times N_r}$;
2: Construct a matrix $W \in \mathbb{R}^{N\times N_r}$;
3: Define $q_r(x_r) = W^T q(Vx_r)$, $f_r(x_r) = W^T f(Vx_r)$, $b_r(t) = W^T b(t)$.

3.2 LTI MOR based on the Linear Projection Framework

For simplicity, we consider the LTI system where $q(x) = x$, $f(x) = Gx$, $b(t) = Bu$, *i.e.*,

$$\frac{d}{dt}x + Gx + Bu = 0, \quad y = D^T x. \tag{3.3}$$

Therefore, under the linear projection framework, we obtain a reduced model where $q_r(x_r) = W^T V x_r$, $f_r(x_r) = W^T G V x_r$, $b_r(t) = W^T B u$.

To further simplify the analysis and discussion, we assume that the left projection matrix W is chosen as

$$W^T = (V^T V)^{-1} V^T, \quad (W^T V = I), \tag{3.4}$$

and therefore, the reduced model is

$$\frac{d}{dt}x_r + G_r x_r + B_r u = 0, \quad y = D_r^T x_r \tag{3.5}$$

where

$$\begin{aligned} G_r &= W^T G V, \\ B_r &= W^T B, \\ D_r &= V^T D. \end{aligned} \tag{3.6}$$

3.2.1 Set of Projected Reduced Models

Under the optimization formulation (2.1), the LTI reduced model is in the set $\mathcal{M}_{r,LTI}$ of all differential equations in the form of (3.5), and is determined by all choices of (G_r, B_r, D_r),

i.e.,

$$\mathcal{M}_{r,LTI} = \left\{ \frac{d}{dt}x_r + G_r x_r + B_r u = 0, y = D_r^T x_r \middle| G_r \in \mathbb{R}^{N_r \times N_r}, B_r \in \mathbb{R}^{N_r \times N_i}, D_r \in \mathbb{R}^{N_r \times N_o} \right\}. \tag{3.7}$$

In contrast, the reduced model in the set of projected models $\mathcal{M}_{r,V}$, (3.5) with (3.6), is determined by all pairs of (V, W) where $W^T = (V^T V)^{-1} V^T$, *i.e.*,

$$\mathcal{M}_{r,V} = \left\{ \frac{d}{dt}x_r + W^T G V x_r + W^T B u = 0, y = D^T V x_r \middle| V \in \mathbb{R}^{N \times N_r}, W^T = (V^T V)^{-1} V^T \right\}. \tag{3.8}$$

Furthermore, instead of choosing the projection matrix V directly, we may choose a N_r-dimensional subspace $\mathcal{V}$ in $\mathbb{R}^N$, *i.e.*, an element in the Grassmann manifold [111]. Mathematically, $\mathcal{V} \in \text{Grass}(N_r, N) = \mathbb{R}^{N \times N_r}/GL_{N_r}$ where GL_{N_r} is the general linear group, *i.e.*, the set of all $N_r \times N_r$ invertible matrices. The dimension of the Grassmann manifold $\text{Grass}(N_r, N)$ is therefore $N_r(N - N_r)$. Using $\mathcal{V}$, the set of projected models is

$$\mathcal{M}_{r,\mathcal{V}} = \left\{ \frac{d}{dt}x_r + W^T G V x_r + W^T B u = 0, y = D^T V x_r \middle| V \in \text{Grass}(N_r, N), W^T = (V^T V)^{-1} V^T \right\}. \tag{3.9}$$

From (3.7), (3.8), (3.9), we may conclude that $\mathcal{M}_{r,V}$ is a subset of $\mathcal{M}_{r,LTI}$, and that there is a many-to-one mapping from $\mathcal{M}_{r,V}$ to $\mathcal{M}_{r,\mathcal{V}}$, as illustrated in Fig. 3.1. It is also tempting to say that the degrees of freedom in choosing a model in $\mathcal{M}_{r,LTI}$, $\mathcal{M}_{r,V}$, $\mathcal{M}_{r,\mathcal{V}}$ are $N_r^2 + N_r(N_i + N_o)$, $N_r N$, $N_r(N - N_r)$, respectively, by counting the number of variables.

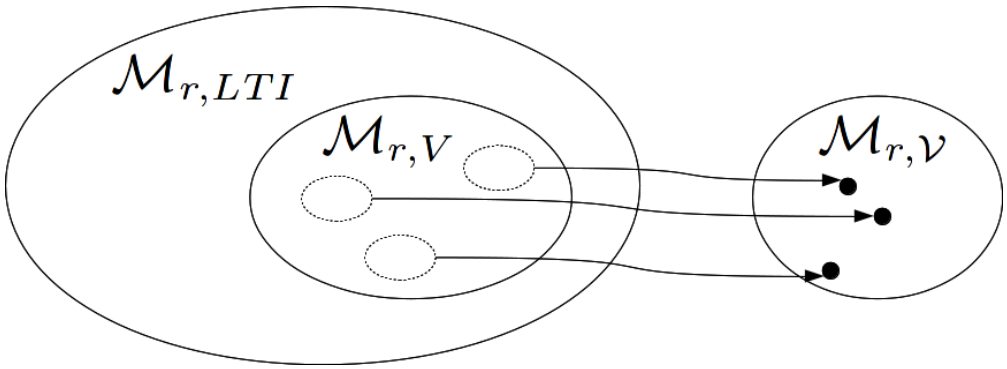


Figure 3.1: Relationship among $\mathcal{M}_{r,LTI}$, $\mathcal{M}_{r,V}$ and $\mathcal{M}_{r,\mathcal{V}}$.

This discussion raises two important questions:

1. What is the degree of freedom in choosing an LTI reduced model?

2. Are the projected models good candidates to use in MOR (*i.e.*, do they encompass all possible LTI reduced models)? If not, why are the projected models useful?

3.2.2 Degree of Freedom in Choosing SISO LTI Models

To determine the degree of freedom in choosing an LTI model in the form of (3.3), we study its canonical forms. Mathematically, given a set and an equivalent relation, the canonical form is a unique representation of an equivalence class, *i.e.*, all equivalent elements can be mapped to the same canonical form. Therefore, by studying the canonical forms, we may determine its degree of freedom.

For simplicity, we consider SISO systems in the following, *i.e.*, $N_i = N_o = 1$. For SISO LTI systems, the canonical forms include transfer function, controllable canonical form, observable canonical form, *etc.*. In particular, we study the transfer function and controllable canonical form.

Definition 3.2.1 (Transfer function) *The transfer function of (3.3) is*

$$\begin{aligned} H(s) &= D^T(sI+G)^{-1}B \\ (\textit{Let } G = P\Lambda P^{-1} \textit{ be its eigen-decomposition}) &= D^T P(sI+\Lambda)^{-1}P^{-1}B \\ (\textit{Assuming } \Lambda \textit{ is diagonal}) &= \sum_{i=1}^{N} \frac{r_i}{s+\lambda_i}, \end{aligned} \tag{3.10}$$

where

$$\begin{aligned} &r_i = \tilde{d}_i \tilde{b}_i^T \in \mathbb{R}, \\ &\tilde{D} = [\tilde{d}_1, \cdots, \tilde{d}_N] = P^T D, \\ &\tilde{B} = [\tilde{b}_1^T; \cdots; \tilde{b}_N^T] = P^{-1}B. \end{aligned} \tag{3.11}$$

To fully specify (3.10), we need to choose $\mathrm{diag}(\Lambda) \in \mathbb{R}^N$, $r = [r_1, \cdots, r_N] \in \mathbb{R}^N$, resulting in a degree of freedom $2N$.

Theorem 3.2.1 (Controllable canonical form) *Let (by Cayler-Hamilton theorem)*

$$G^n B = -\sum_{i=0}^{n-1} \alpha_i G^i B. \tag{3.12}$$

Then we can find a matrix $T \in \mathbb{R}^{N\times N}$ so that by defining $x = T\xi$, the LTI model in terms of ξ is in the controllable canonical form, i.e.,

$$\frac{d}{dt}\xi = T^{-1}GT\xi + T^{-1}Bu, \quad y = D^T T\xi, \tag{3.13}$$

where

$$T^{-1}GT = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -\alpha_0 & -\alpha_1 & -\alpha_2 & \cdots & -\alpha_{N-2} & -\alpha_{N-1} \end{bmatrix}, \quad T^{-1}B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{3.14}$$

Proof Define $M_c = [B, GB, \cdots, G^{N-1}B]$ to be the controllability matrix. As a consequence,

$$\begin{aligned} GM_c &= G[B, GB, \cdots, G^{N-1}B] \\ &= [GB, G^2B, \cdots, G^NB] \\ &= M_c \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -\alpha_0 \\ 1 & 0 & 0 & \cdots & 0 & -\alpha_1 \\ 0 & 1 & 0 & \cdots & 0 & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & 0 & -\alpha_{N-2} \\ 0 & 0 & 0 & \cdots & 1 & -\alpha_{N-1} \end{bmatrix} \\ &= M_cQ. \end{aligned} \tag{3.15}$$

Let $\hat{G} = T^{-1}GT$ and $\hat{B} = T^{-1}B$. Because the two systems have the same characteristic polynomial, we have

$$\hat{G}\hat{M}_c = \hat{M}_cQ. \tag{3.16}$$

Consequently, choosing

$$T = M_c\hat{M}_c^{-1}, \tag{3.17}$$

leads to (3.14). ■

To fully specify (3.13), we need to choose $\alpha = [\alpha_0, \cdots, \alpha_{N-1}] \in \mathbb{R}^N$, and $D^TT \in \mathbb{R}^N$, leading to a degree of freedom $2N$.

Therefore, we conclude the following result

Theorem 3.2.2 *The degree of freedom in determining an N-dimensional SISO LTI model is $2N$.*

From theorem 3.2.2, the degree of freedom in determining an N_r-dimensional reduced model is $2N_r$. We also see that the degrees of freedom in choosing $\mathcal{M}_{r,LTI}$, $\mathcal{M}_{r,V}$, $\mathcal{M}_{r,\mathcal{V}}$ are misleading because in each of these sets, a lot of elements are equivalent.

Corollary 3.2.3 (Modal form) *Let $G = P\Lambda P^{-1}$ where diag(Λ) are the eigenvalues of G and columns of P are eigenvectors of G. Define $\tilde{x} = P^{-1}x$, and replace $x = P\tilde{x}$ in (3.3), we obtain the decoupled form*

$$\frac{d}{dt}\tilde{x} = \Lambda\tilde{x} + P^{-1}Bu, \quad y = D^T P\tilde{x}, \tag{3.18}$$

which consists of N decoupled one-dimensional differential equations.

The decoupled form of an LTI model is not canonical.

Proof To fully specify the decoupled form (3.18), we need to choose diag$(\Lambda) \in \mathbb{R}^N$, $P^{-1}B \in \mathbb{R}^N$ and $P^T D \in \mathbb{R}^N$, leading to a degree of freedom $3N$.

However, any model defined by (G, B_1, D_1) so that $D_1^T P \cdot P^{-1}B_1 = D^T P \cdot P^{-1}B$ (element-wise multiplication) is equivalent (in the transfer function) to the model defined by (G, B, D), thus eliminating N degrees of freedom. ■

To be more general, we consider SIAO (Single-Input All-Output) systems:

Definition 3.2.2 (SIAO systems) *A SIAO system is described by (3.3) where $N_i = 1$ and $D = I_N$ (identity matrix of size N).*

Corollary 3.2.4 *The degree of freedom in determining an N-dimensional SIAO LTI model is $N + N^2$.*

The above discussion leads to the notion of equivalence between two SISO or SIAO systems.

Definition 3.2.3 (Equivalence of SISO LTI systems) *Two SISO LTI systems*

$$\frac{d}{dt}x + G_1 x + B_1 u, \quad y = D_1^T x \tag{3.19}$$

and

$$\frac{d}{dt}x + G_2 x + B_2 u, \quad y = D_2^T x \tag{3.20}$$

are equivalent if they have the same transfer function, i.e.,

$$H_1(s) = D_1^T(sI + G_1)^{-1}B_1 = H_2(s) = D_2^T(sI + G_2)^{-1}B_2. \tag{3.21}$$

According to this definition, we have the following theorem:

Theorem 3.2.5 *Two SISO LTI systems are equivalent if and only if they have the same controllable canonical form.*

Proof If two SISO LTI systems have the same controllable canonical form, then it is obvious that they have the same transfer function, and therefore are equivalent.

If two SISO LTI systems are equivalent, because the controllable canonical form is unique, they have the same controllable canonical form. ■

3.2.3 Degree of Freedom in Choosing Projected SISO LTI Models

At a first glance, a projected model in (3.8) is determined by $V \in \mathbb{R}^{N\times N_r}$, *i.e.*, NN_r variables.

However, an N_r-dimensional SISO model can be determined by $2N_r$ variables. This implies that there are many choices of V that lead to the same reduced model.

Suppose we obtain two reduced models by applying projection defined by V_1 and V_2, *i.e.*,

$$\frac{d}{dt}x_r + W_1^T G V_1 x_r + W_1^T B u = 0, \quad y = D^T V_1 x_r, \quad W_1^T = (V_1^T V_1)^{-1} V_1^T, \tag{3.22}$$

and

$$\frac{d}{dt}x_r + W_2^T G V_2 x_r + W_2^T B u = 0, \quad y = D^T V_2 x_r, \quad W_2^T = (V_2^T V_2)^{-1} V_2^T. \tag{3.23}$$

We derive conditions for (3.22) to be equivalent to (3.23), and then discuss the degree of freedom in choosing a projected model.

Theorem 3.2.6 *If span(V_1) = span(V_2), then (3.22) is equivalent to (3.23).*

Proof If span(V_1) and span(V_2) are the same, then there exists an invertible matrix $Q \in \mathbb{R}^{N_r\times N_r}$, such that $V_2 = V_1 Q$.

Therefore, we have

$$\begin{aligned} H_2(s) &= D^T V_1 Q (sI + (Q^T V_1^T V_1 Q)^{-1} Q^T V_1^T G V_1 Q)^{-1} (Q^T V_1^T V_1 Q)^{-1} Q^T V_1^T B \\ &= D^T V_1 Q (sQ^T V_1^T V_1 Q + Q^T V_1^T G V_1 Q)^{-1} Q^T V_1^T B \\ &= D^T V_1 (sV_1^T V_1 + V_1^T G V_1)^{-1} V_1^T B \\ &= H_1(s). \end{aligned} \tag{3.24}$$

That is, the two models are equivalent. ■

Theorem 3.2.6 shows that $\mathcal{M}_{r,V}$ can be reduced to $\mathcal{M}_{r,\mathcal{V}}$ – the subspace, rather than the projection matrix, matters in the behavior of the projected model.

A further question is: does there exist V_1 and V_2, span$(V_1) \neq$ span(V_2), but the projected reduced model are equivalent? For SISO systems, the answer is yes, and therefore the degree of freedom in choosing a projected model must be less than the dimension of Grass(N_r, N).

Representativeness of Projected Models

From the above discussion, we observe that to determine a projected model, we need to determine a subspace from $\mathcal{V} \in$ Grass(N_r, N) whose dimension is $N_r(N - N_r)$, while an arbitrary N_r-dimensional LTI SISO model is determined by $2N_r$ numbers.

However, the projected models have limited power in representing general N_r-dimensional models. Therefore, we have the following theorem.

Theorem 3.2.7 *Projected models cannot represent arbitrary N_r-dimensional LTI SISO models. In particular, they are highly dependent on the original full model.*

In the following, we show several counter examples in which projected models are not general enough to cover all possible N_r-dimensional models:

Example 3.2.8 *1. Suppose $N = N_r + 1$, we have only N_r degrees of freedom in choosing the projected model, while we have $2N_r$ degrees of freedom in choosing an arbitrary N_r-dimensional LTI SISO model. It is therefore not possible to cover all possible LTI SISO models.*

2. Consider deriving a one-dimensional projected LTI model by choosing $V \in \mathbb{R}^{N\times 1}$ and $V^TV = 1$. Let G be positive definite. Therefore, in the reduced model,

$$G_r = W^TGV = V^TGV \succ 0. \tag{3.25}$$

Therefore, the reduced model is always stable. While this is probably rather a good result in practice, it shows that the projected model cannot represent any non-stable model.

3. If G is identity, we have

$$G_r = W^TI_NV = I_{N_r}, \tag{3.26}$$

and therefore, the poles of the reduced transfer function can only be 1. The projected model is obviously not general.

In general, in order to be equivalent to a given N_r-dimensional LTI model, the projected model must satisfy $2N_r$ or $N_r^2 + N_r$ constraints for SISO or SIAO systems, respectively, (*i.e.*, by equating variables in the canonical forms). Therefore, the number of elements in the set of N_r-dimensional subspaces must be larger than the degree of freedom (*i.e.*, $N_r(N-N_r) \geq 2N_r$ $(N \geq N_r + 2)$ or $N_r(N - N_r) \geq N_r^2 + N_r$ $(N \geq 2N_r + 1)$, respectively), in order to make the projected model general. While there is still no guarantee that the projected model is general enough, this appears as a viable necessary condition.

While projected models are not general enough to represent all N_r-dimensional models, they are still most widely used, due to the following reasons:

1. Projection is a natural way to map from $G \in \mathbb{R}^{N\times N}$ to $G_r \in \mathbb{R}^{N_r\times N_r}$;

2. There are efficient algorithms to generate projected reduced models that are reasonably accurate.

3.2.4 Computational Complexity of Projected Models

One important application of reduced models is to speedup simulations. In this regard, the computational complexity is extremely important.

In particular, the time complexity of computing matrix-vector multiplication Gx and linear solve $Gx = b$ as well as the space complexity of storing G are important.

In the full model, we usually have a sparse matrix G in which the number of non-zero entries is of the order $O(N)$ (due to the fact that physical systems are only locally coupled). Let this number upper-bounded by KN with K small, we have that the matrix vector multiplication Gx takes $2KN$ flops, the linear solve takes $O(N^\alpha)$ with α being around 1.2, and the space complexity is KN.

In contrast, the projected matrix G_r is usually dense, and therefore has a space complexity N_r^2. The matrix-vector multiplication takes $2N_r^2$ flops, and the linear solve takes $O(N_r^3)$. These results are summarized in Table 3.1.

Table 3.1: Computational complexity of full model and reduced model

	matrix-vector multiplication	linear solve	memory usage
(sparse) full model	$2KN$	$\simeq O(N^{1.2})$	KN
(dense) projected reduced model	$2N_r^2$	$O(N_r^3)$	N_r^2

This exercise gives us a rough idea of what the reduced model size should be in order to achieve computational speedups. Take the matrix-vector multiplication for example, we will gain speedups only if $KN \gg N_r^2$.

3.3 Nonlinear MOR based on the Linear Projection Framework

For a nonlinear system

$$\frac{d}{dt}x + f(x) + Bu = 0, \quad y = D^T x, \tag{3.27}$$

the linear projection will lead to a reduced model of the form

$$\frac{d}{dt}x_r + f_r(x_r) + B_r u = 0, \quad y = D_r^T x_r, \tag{3.28}$$

where

$$\begin{aligned} f_r(x_r) &= W^T f(V x_r), \\ B_r &= W^T B, \\ D_r &= V^T D. \end{aligned} \tag{3.29}$$

3.3.1 Set of Projected Reduced Models

Similar the the linear case, the nonlinear reduced model is in the set $\mathcal{M}_{r,NL}$ of all differential equations in the form of (3.28), and is determined by all choices of $(f_r(\cdot), B_r, D_r)$, *i.e.*,

$$\mathcal{M}_{r,NL} = \left\{ \frac{d}{dt}x_r + f_r(x_r) + B_r u = 0, y = D_r^T x_r \middle| f_r : \mathbb{R}^{N_r} \to \mathbb{R}^{N_r}, B_r \in \mathbb{R}^{N_r \times N_i}, D_r \in \mathbb{R}^{N_r \times N_o} \right\}. \tag{3.30}$$

In contrast, the reduced model in the set of projected models $\mathcal{M}_{r,V}$, (3.28) with (3.29), is determined by all pairs of (V, W) where $W^T = (V^T V)^{-1} V^T$, *i.e.*,

$$\mathcal{M}_{r,V} = \left\{ \frac{d}{dt}x_r + W^T f(V x_r) + W^T B u = 0, y = D^T V x_r \middle| V \in \mathbb{R}^{N \times N_r}, W^T = (V^T V)^{-1} V^T \right\}. \tag{3.31}$$

Furthermore, instead of choosing a projection matrix V, but choosing a N_r-dimensional subspace $\mathcal{V}$ in $\mathbb{R}^N$, we have

$$\mathcal{M}_{r,\mathcal{V}} = \left\{ \frac{d}{dt}x_r + W^T f(V x_r) + W^T B u = 0, y = D^T V x_r \middle| V \in \mathrm{Grass}(N_r, N), W^T = (V^T V)^{-1} V^T \right\}. \tag{3.32}$$

We hope to answer the same two questions we have addressed in the linear case:

1. What is the degree of freedom in choosing a nonlinear reduced model?
2. Are the projected models good candidates to use in MOR (*i.e.*, do they encompass all possible nonlinear reduced models; if not what subset of models can be represented)?

However, we are not able to give exact answers, due to the lack of canonical form and the lack of analytical response functions such as transfer functions.

3.3.2 Computational Complexity of Projected Models

In the full model, the nonlinear function $f(x)$ is "sparse", meaning that each element $f_i(x)$ depends on only a few variables (K of them), and only takes at maximum L flops for each nonlinear kernel. Therefore, the function evaluation takes NKL flops, and the Jacobian matrix has only NK non-zero elements and is sparse.

In contrast, the projected nonlinear function $f_r(x_r)$ is "dense", meaning that $f_{r,i}(x_r)$ depends on all variables in x_r. Without any symbolic simplification, $f_r(x_r)$ can be computed by computing $x = Vx_r$, $f(x)$ and $W^T f(x)$ sequentially, resulting in $NKL + 2NN_r$ flops. This is always larger than the that of the full model.

On the other hand, unlike the LTI case where $W^T GV$ can be pre-computed and stored in the reduced model, there is no simple way to store a nonlinear function $W^T f(V\cdot)$, therefore resulting in difficulties of storing the reduced model.

We will discuss methods to handle such difficulties in Section 4.7.

Chapter 4

Nonlinear Projection Framework for Model Order Reduction

While the linear projection framework works very well for reducing LTI models, it is less so for reducing nonlinear models. In this chapter, we study how linear projections can fail in nonlinear MOR, from which we motivate and propose the nonlinear projection framework that provides an elegant alternative for nonlinear MOR. We analyze each component in this framework (*e.g.*, choosing manifolds and projections), and present general guidelines of deriving nonlinear projected reduced models under this framework.

4.1 Motivation

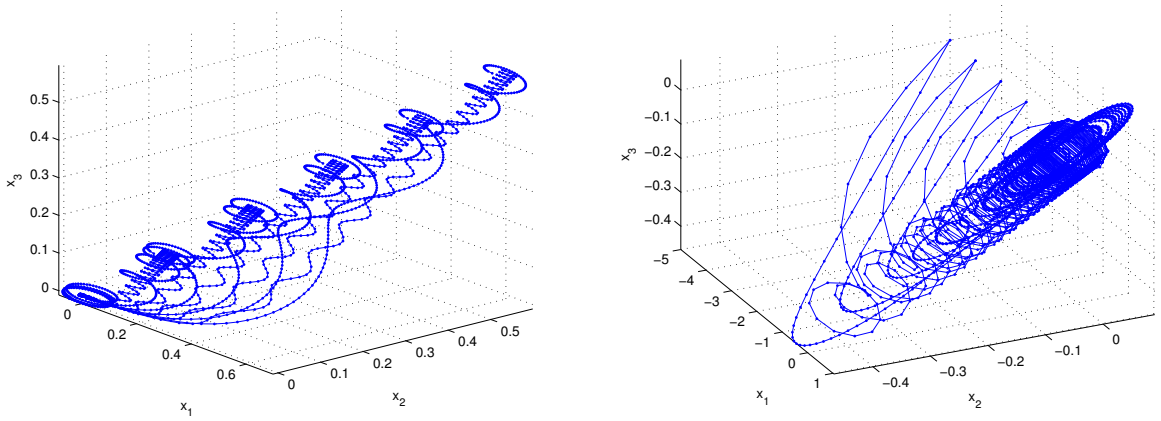
The linear projection framework amounts to finding two subspaces $\mathcal{V} = \text{span}(V)$ and $\mathcal{W} = \text{span}(W)$ and imposing constraints $x \in \mathcal{V}$, $r \perp \mathcal{W}$. Loosely speaking, we want to find two subspace $\mathcal{V}$ and $\mathcal{W}$ so that the solution of the full model stay very close to $\mathcal{V}$ and that the residual of the reduced model solution is orthogonal to $\mathcal{W}$.

Example 4.1.1 (A simple LTI system) *Consider a simple LTI system defined by*

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -10 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u(t), \tag{4.1}$$

where $x = [x_1, x_2, x_3]^T$ are the three state variables, $u(t)$ is the input.

We perform a series of simulations of (4.1) by applying several random sinusoidal inputs. The trajectories of $x(t)$ and $\frac{dx}{dt}(t)$ are plotted in Fig. 4.1(a) and Fig. 4.1(b), respectively. In this case, the trajectories lie exactly on a 2-dimensional linear subspace, and we expect that the MOR algorithm is able to identify that linear subspace.



(a) $x(t)$. (b) $\frac{dx}{dt}(t)$.

Figure 4.1: Trajectories of (4.1) in the state space.

While the condition that trajectories stay close to a linear subspace often (provably) holds for LTI systems, it often does not hold for nonlinear systems. Rather, trajectories of a nonlinear system tend to stay on a nonlinear manifold.

Example 4.1.2 (A simple nonlinear system) *Consider a nonlinear system*

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -10 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ x_1^2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u(t), \tag{4.2}$$

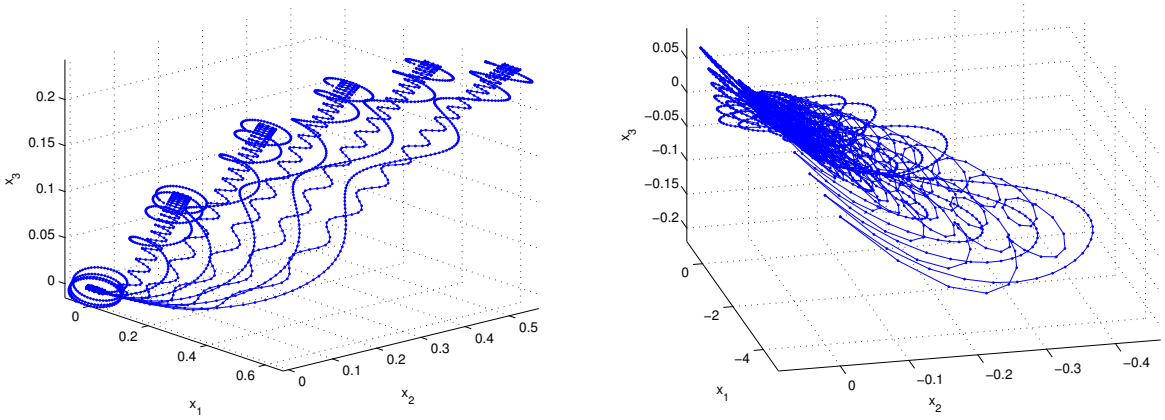
which adds a quadratic nonlinear term to the LTI system (4.1). Simple as it is, it exhibits very different behaviors from the LTI system (4.1).

We apply the same random inputs to (4.2) (as those to (4.1)), and the trajectories of $x(t)$ and $\frac{dx}{dt}(t)$ are plotted in Fig. 4.2(a) and Fig. 4.2(b), respectively. We observe that $x(t)$ and $\frac{dx}{dt}(t)$ no longer stay on a linear subspace. Rather, there seem to be a nonlinear manifold attracting the trajectories.

This example suggests that nonlinear projections might be better for nonlinear model order reduction, and motivates us to consider the nonlinear projection framework.

4.2 Nonlinear Projection Framework

In the nonlinear projection framework, we try to identify two manifolds $\mathcal{V}$ and $\mathcal{W}$ that are sub-manifolds of $\mathbb{R}^N$. To generalize the linear projection framework, we define $x = v(x_r)$



(a) $x(t)$. (b) $\frac{dx}{dt}(t)$.

Figure 4.2: Trajectories of (4.2) in the state space.

and constrain $w(r(x_r)) = 0$, where $v : \mathbb{R}^{N_r} \to \mathbb{R}^N$ and $w : \mathbb{R}^N \to \mathbb{R}^{N_r}$ define manifolds $\mathcal{V}$ and $\mathcal{W}$, respectively.

Therefore, the general skeleton of an MOR algorithm in the nonlinear projection framework is:

Algorithm 3 MOR based on Nonlinear Projection

Inputs: $r(x) = \frac{d}{dt}q(x) + f(x) + b(t) = 0$
Outputs: $w(r(v(x_r))) = w\left(\frac{d}{dt}q(v(x_r)) + f(v(x_r)) + b(t)\right) = 0$

1: Construct a mapping $v : \mathbb{R}^{N_r} \to \mathbb{R}^N$;
2: Construct a mapping $w : \mathbb{R}^N \to \mathbb{R}^{N_r}$;
3: Define the reduced model to be $w(r(v(x_r))) = 0$.

Algorithm 3 is very general. To be less general, we will consider $w(r) = W^T(x)r$, where $W^T : \mathbb{R}^N \to \mathbb{R}^{N_r \times N}$, thus leading to Algorithm 4.

Different from the linear projection framework, where the reduced model is determined by a pair of projection matrices (V, W) or subspaces $(\mathcal{V}, \mathcal{W})$, we now have a much wider choices of projection functions and manifolds. For example, if we consider the quadratic projection functions, *i.e.*,

$$v(x_r) = V_1 x_r + V_2 x_r \otimes x_r, \tag{4.3}$$

then the reduced model is parameterized by (V_1, V_2) (assuming $w(\cdot)$ is derived from $v(\cdot)$, for simplification).

The fundamental question in designing a nonlinear MOR method using Algorithm 3 is

Algorithm 4 MOR based on Nonlinear Projection with $w(r) = W^T(x)r$

Inputs: $\frac{d}{dt}q(x) + f(x) + b(t) = 0$
Outputs: $W^T(v(x_r))\left(\frac{d}{dt}q(v(x_r)) + f(v(x_r)) + b(t)\right) = 0$
1: Construct a mapping $v : \mathbb{R}^{N_r} \to \mathbb{R}^N$;
2: Construct a mapping $w(r) = W^T(x)r$ where $W^T : \mathbb{R}^N \to \mathbb{R}^{N_r \times N}$;
3: Define $w(r(v(x_r))) = 0$.

how to identify manifolds $\mathcal{V}$ and $\mathcal{W}$. Similar to LTI MOR, the manifold to be used in MOR depends on the reduction criterion $F(\cdot,\cdot)$, $G(\cdot,\cdot)$ and $H(\cdot,\cdot)$ in (2.1). In this section, rather than being very general to answer this question, we study some candidate manifolds, and try to draw insights and conclusions.

The notations and simplifications that will be used in this section include:

1. We consider reducing ODE models, instead of DAE models, *i.e.*,

$$\frac{d}{dt}x = f(x) + Bu. \tag{4.4}$$

2. Suppose $x \in \mathcal{V}$, *i.e.*, $x = v(x_r)$. We denote $V(x)$ for all $x = v(x_r)$ by

$$V(x) = \left.\frac{dv}{dx_r}\right|_{x=v(x_r)}. \tag{4.5}$$

3. Define $W^T(x) = W^T(v(x_r))$ such that

$$W^T(v(x_r))V(v(x_r)) = I. \tag{4.6}$$

In this case, the reduced model is simplified to

$$\frac{d}{dt}x_r = f_r(x_r) + B_r(x_r)u, \tag{4.7}$$

where

$$f_r(x_r) = W^T(v(x_r))f(v(x_r)), \quad B_r(x_r) = W^T(v(x_r))B. \tag{4.8}$$

Therefore, $\mathcal{W}$ is determined by $\mathcal{V}$.

4.3 Manifold Integrating Local Krylov Subspaces

The first candidate manifold we study is the one that "integrates" local Krylov subspaces.

For nonlinear systems, properties are often described in terms of linearized systems. For example, TPWL [2] tries to find a linear subspace which covers the local Krylov subspaces

around all expansion points. The linear subspace, however, is likely to grow very quickly as the number of expansion points increases.

In order to avoid the growth of the order of reduced models due to the inclusion of many linear subspaces, we expect to construct a manifold $\mathcal{V}$ so that at each point $x \in \mathcal{V}$, the tangent space around x, $T_x\mathcal{V}$ matches the local Krylov subspace of the linearized system at x.

In another word, the mapping $v : \mathbb{R}^{N_r} \to \mathbb{R}^N$ should satisfy

$$\text{span}(V(x)) = \text{span}\left(\left.\frac{\partial v}{\partial x_r}\right|_{x=v(x_r)}\right) = \mathcal{K}_{N_r,x}, \tag{4.9}$$

where $\mathcal{K}_{N_r,x}$ is the N_r-dimensional Krylov subspace of the linearized system around x. Such a manifold is called an **integral manifold**.

To see why the manifold satisfying (4.9) is useful, we show that responses of the full linearized model and the reduced linearized model are closely related.

By linearizing (4.4) around $x_i \in \mathbb{R}^N$, (*i.e.*, let $x = x_i + \Delta x$, with higher-order terms of Δx truncated), we obtain

$$\frac{d}{dt}\Delta x = G(x_i)\Delta x + f(x_i) + Bu, \tag{4.10}$$

where

$$G(x) = \frac{\partial f}{\partial x}(x). \tag{4.11}$$

In contrast, the linearized model of (4.7) around $x_{r,i}$

$$\frac{d}{dt}\Delta x_r = G_r(x_{r,i})\Delta x_r + f_r(x_{r,i}) + B_r(x_{r,i})u, \tag{4.12}$$

where[1]

$$\begin{aligned} G_r(x_r) &= \frac{\partial}{\partial x_r}\left[W^T(v(x_r))[f(v(x_r)) + Bu]\right] \\ &= W^T(v(x_r))G(x)|_{x=v(x_r)}V(v(x_r)) + \frac{\partial W^T(v(x_r))}{\partial x_r}\cdot(f(v(\hat{x})) + Bu), \\ B_r(x_r) &= \frac{\partial}{\partial u}\left[W^T(v(x_r))[f(v(x_r)) + Bu]\right] \\ &= W^T(v(x_r))B. \end{aligned} \tag{4.13}$$

To compare (4.10) with (4.12), let $x_i = v(x_{r,i})$, and consider x_i that is an equilibrium

[1]Note: $\frac{\partial W^T(\hat{x})}{\partial \hat{x}}$ is a 3-dimensional tensor, and therefore the notation $\frac{\partial W^T(v(x_r))}{\partial x_r}\cdot(f(v(\hat{x})) + Bu)$ denotes a matrix of size $N_r \times N_r$.

point, *i.e.*,

$$f(x_i) + Bu_{DC} = 0, \tag{4.14}$$

where u_{DC} is a DC (constant) input.

Given (4.14), the second part of $G_r(x_r)$ in (4.13) vanishes, and therefore (4.12) is a linear reduced model of (4.10) with projection matrices $V(v(x_r))$ and $W(v(x_r))$. Furthermore, since $\text{span}(V(v(x_r))) = \mathcal{K}_{N_r,v(x_r)}$, the first N_r moments of the transfer function of (4.12) match those of (4.10).

4.4 Existence of an Integral Manifold

The manifold defined in the last section would be useful for model reduction. However, the key question is: can we find an N_r-dimensional manifold satisfying conditions (4.9)? Unfortunately, generally the answer is no. To show that, we resort to a fundamental result in differential geometry [73, 112, 113], the Frobenius theorem.

To explain the result of Frobenius theorem, we introduce a few terminologies in differential geometry[2]:

1. Let X denote a manifold.

2. A **distribution** on X is a map Δ which assigns to each point p in X a subspace of the tangent space at this point, *i.e.*,

$$p \in X \to \Delta(p) \in T_pX. \tag{4.15}$$

3. A **foliation** $\{S_\alpha\}_{\alpha\in A}$ of X of dimension k is a partition

$$X = \bigcup_{\alpha\in A} S_\alpha \tag{4.16}$$

of X into disjoint connected sub-manifolds S_α, called **leaves**, which has the following property: for any $p \in X$ there exists a neighborhood U of p and a diffeomorphism $\Phi : U \to V \subset \mathbb{R}^N$ onto an open subset V such that

$$\Phi((U \bigcap S_\alpha)_{cc}) = \{p = (x_1, \cdots, x_N) \in V | x_{k+1} = c_\alpha^{k+1}, \cdots, x_N = c_\alpha^N\}, \tag{4.17}$$

where P_{cc} denotes a connected component of the set P.

4. A distribution of constant dimension $p \to \Delta(p)$ on X is **integrable** if there exists a

[2]For self-contained ness, we provide a short introduction to differential geometry that is relevant to our discussion in Appendix Chapter A.

foliation $\{S_\alpha\}_{\alpha\in A}$ on X such that for any $p \in X$,

$$T_pS = \Delta(p), \tag{4.18}$$

where S is the leaf passing through p.

This series of rather semi-formal definitions leads to the following important result: there exists a sub-manifold S that satisfies $T_pS = \Delta(p)$ for some distribution Δ if the distribution Δ is integrable.

Furthermore, the other important result that makes us able to tell the integrability of a distribution Δ is given by the Frobenius theorem.

Theorem 4.4.1 ((Global) Frobenius theorem) *A smooth distribution Δ of constant dimension is integrable if and only if it is involutive. The integral foliation of Δ is the partition of a manifold X into orbits of the family of vector fields $\{g|g \in \Delta\}$.*

Theorem 4.4.2 ((Local) Frobenius theorem) *A distribution $\Delta = span(g_1, \cdots, g_k)$ of constant dimension k is involutive if and only if there exists a diffeomorphism φ such that if $\xi = \varphi(x)$, then*

$$(D\varphi\Delta)(\xi) = span(D\varphi g_1, \cdots, D\varphi g_k) = span\left(\frac{\partial}{\partial\xi_1}, \cdots \frac{\partial}{\partial\xi_k}\right). \tag{4.19}$$

In Theorem 4.4.1 and 4.4.2, we encounter the definition of the **involutivity** of a distribution.

Definition 4.4.1 (Involutivity of a distribution) *A distribution Δ is involutive if for any vector fields $f, g \in \Delta$, their Lie bracket is also in Δ, i.e., $[f, g] \in \Delta$.*

Definition 4.4.2 (Lie bracket (coordinate-dependent)) *The Lie bracket of two vector fields f and g, denoted by $[f, g]$, is*

$$[f, g] = \frac{\partial g}{\partial x}(x)f(x) - \frac{\partial f}{\partial x}(x)g(x). \tag{4.20}$$

From the above discussion and theorems, we conclude:

1. We can define a sub-manifold by defining an integrable distribution Δ that specifies the tangent space at each point on the sub-manifold.
2. We can check the integrability of a distribution by checking its involutivity.
3. We can check the involutivity of a distribution by checking whether the Lie bracket of two vector fields in the distribution is still in the distribution.

In Section 4.3, we have specified the distribution Δ to be the one assigning Krylov subspaces to each point on the manifold, *i.e.*,

$$\Delta_{\mathcal{K}}(x) \triangleq \text{span}(\mathcal{K}_{N_r}) = \text{span}(G^{-1}(x)B, \cdots, G^{-N_r}(x)B), \quad \forall x = v(x_r), x_r \in \mathbb{R}^{N_r}. \tag{4.21}$$

One distribution that satisfies (4.21) is

$$\Delta_{\mathcal{K}}(x) \triangleq \text{span}(\mathcal{K}_{N_r}) = \text{span}(G^{-1}(x)B, \cdots, G^{-N_r}(x)B), \quad \forall x \in \mathbb{R}^{N}. \tag{4.22}$$

Unfortunately, **this distribution is generally not involutive**, and therefore such a manifold may not exist. A counter-example can be constructed as follows:

Example 4.4.3 *Consider a linearized model in the form of*

$$A(x)\frac{dx}{dt} + x + Bu = 0, \tag{4.23}$$

where $A(x) \in \mathbb{R}^{N\times N}$ *and* $B \in \mathbb{R}^{N\times 1}$.

They Krylov basis of (4.23) is

$$V(x) = [A(x)B, A(x)^2B, \cdots, A(x)^kB]. \tag{4.24}$$

Let $N = 3$, $N_r = 2$. *Let* $A(x)$ *be the Jacobian of a nonlinear function* $g(x)$, *given by*

$$g(x) = \begin{bmatrix} x_1x_3 \\ x_2x_3 \\ x_1 \end{bmatrix}, \quad A(x) = \frac{\partial g}{\partial x}(x) = \begin{bmatrix} x_3 & 0 & x_1 \\ 0 & x_3 & x_2 \\ 1 & 0 & 0 \end{bmatrix}, \tag{4.25}$$

and let B *be a constant vector*

$$B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \tag{4.26}$$

Therefore, the two-dimensional Krylov subspace is the span of $\{v_1 = A(x)B, v_2 = A(x)^2B\}$, *where*

$$v_1(x) = \begin{bmatrix} x_3 \\ 0 \\ 1 \end{bmatrix}, \quad v_2(x) = \begin{bmatrix} x_3^2 + x_1 \\ x_2 \\ x_3 \end{bmatrix}. \tag{4.27}$$

The Lie bracket $[v_1, v_2]$ *is*

$$[v_1, v_2] = \frac{\partial v_2}{\partial x}v_1 - \frac{\partial v_1}{\partial x}v_2 = \begin{bmatrix} 2x_3 \\ 0 \\ 1 \end{bmatrix}. \tag{4.28}$$

When $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, *we have*

$$v_1(x) = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad v_2(x) = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}, \quad [v_1, v_2] = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} \notin span(\{v_1, v_2\}), \tag{4.29}$$

and therefore Δ *defined by (4.21) is not involutive.*

4.5 Involutive Distribution Matching Krylov Subspaces

While the distribution defined by (4.21) is typically not involutive, we may induce an involutive distribution from it. Given a set of analytical vector fields $\{f_u\}$, $\Delta = \text{span}(\{f_u\})$, we may define

$$L(p) = \text{span}\{g(p) | g \in \text{Lie}\{f_u\}\} \tag{4.30}$$

where $\text{Lie}\{f_u\}$ is the smallest family of vector fields that contains $\{f_u\}$ and is closed under taking linear combinations and Lie bracket (or the **Lie algebra** of $\{f_u\}$).

According to corollary A.7.2, the orbit of $(p, \{f_u\})$ defines a sub-manifold whose tangent space

$$T_p X = L(p). \tag{4.31}$$

In another word, $L(p)$ defines an involutive distribution that covers the subspace defined by Δ at each point p. Therefore, $L(p)$ corresponds to a well-defined sub-manifold.

In order to compute $L(p)$, we can start with $L(p) = \text{span}(\{f_u\}) = \Delta$, and then iteratively add Lie brackets of generator functions in $L(p)$ into $L(p)$ until $L(p)$ is involutive.

The possible caveat of this approach is that there does not seem to have any bound of the dimension of $L(p)$. Therefore, even in the case where we want to match all 2-dimensional Krylov subspaces, $L(p)$ may end up being of a large dimension, making the manifold not so efficient in preserving accuracy of linearized models.

4.6 Constructing Integral Manifolds

We have seen that $\Delta_{\mathcal{K}}$ in (4.21) is generally not involutive, and $\Delta_{\mathcal{K}}$-induced $L(p)$ is involutive but may be of large dimension. However, the discussion gives us a better understanding on how we may define a manifold (more precisely, a sub-manifold in $\mathbb{R}^N$):

1. We may define an involutive distribution, or equivalently, its generator functions, from which a sub-manifold can be derived;

2. We may directly define the projection function $x = v(x_r)$ that specifies the manifold $\mathcal{V}$, and derive its tangent spaces $V(x) = \frac{\partial v}{\partial x_r}(v(x_r))$.

In this section, we consider several ways to construct integral manifolds, and consider special kinds of distributions/manifolds that may be analyzed or computed efficiently.

In the following discussion, we assume the projections $x = v(x_r)$ and $x_r = w(x)$ such that $x_r = w(v(x_r))$ for all x_r. Note that this definition leads to the reduced model

$$\frac{dx_r}{dt} = \frac{dw}{dx}\frac{dx}{dt} = \frac{dw}{dx}(v(x_r))(f(v(x_r)) + Bu), \tag{4.32}$$

which is the same as that in (4.7), and in Algorithm 4.

We denote $\frac{\partial v}{\partial x_r}(x_r)$ as $V(x_r)$, $V(w(x))$ as $V(x)$[3], $\frac{\partial w}{\partial x}(x)$ as $W^T(x)$, and $W^T(v(x_r))$ as $W^T(x_r)$.

Note that $x = v(x_r)$ has N equations, and $x_r = w(x)$ has N_r equations. Therefore, $x = v(x_r)$ specifies all the points on a manifold, while $x_r = w(x)$ specifies equivalent classes, *i.e.*, it specifies points $x \in \mathbb{R}^N$ that map/project to the same x_r. Equivalently, $x_r = w(x)$ defines the projection from x to x_r on the manifold (corresponding to $v(x_r)$ in $\mathbb{R}^N$). Therefore, the assumption $x_r = w(v(x_r))$ simply states that the projection of a point on the manifold is the same point.

Example 4.6.1

1. $[x_1, x_2, x_3] = [x_r, x_r, x_r]$ *specifies a one-dimensional manifold (a line) in* $\mathbb{R}^3$.
2. $x_r = x_1^2 + x_2^2 + x_3^2$ *defines equivalent points in* $\mathbb{R}^3$ *being points on spheres of radius* $\sqrt{x_r}$ *(*$x_r \geq 0$*).*

4.6.1 Involutivity Constraints

To enforce the involutivity of a distribution $\text{span}(V(x)) = \text{span}([v_1, \cdots, v_{N_r}])$ so that it corresponds to an integral manifold, we have involutivity constraints

$$[v_i, v_j] \in \text{span}(V(x)), \quad \forall i, j, i \neq j. \tag{4.33}$$

Using coordinate-dependent definition of the Lie bracket, we have

$$[v_i, v_j] = \frac{\partial v_j}{\partial x} v_i - \frac{\partial v_i}{\partial x} v_j \in \text{span}(V(x)), \quad \forall i, j, i \neq j. \tag{4.34}$$

To simplify (4.34), we note that from the proof of the Frobineous theorem, if $\text{span}(V(x))$ is involutive, then there exists a commuting basis of generators for $\text{span}(V(x))$. Therefore, without loss of generality, we assume that $v_i(x)$ and $v_j(x)$ are commuting, leading to algebraic equations

$$[v_i, v_j] = \frac{\partial v_j}{\partial x} v_i - \frac{\partial v_i}{\partial x} v_j = 0, \quad \forall i, j, i \neq j. \tag{4.35}$$

[3]This definition is an extension to the one in (4.5) for $x \notin \mathcal{V}$, with the assumption that $w(v(x_r)) = x_r$.

For each pair (i, j), (4.35) correspond to N equations. Since there are $\frac{1}{2}N_r(N_r-1)$ (i,j) pairs, the involutivity constraints consist of $\frac{1}{2}N_r(N_r-1)N$ equations.

4.6.2 Method 1

In method 1, we directly construct $v(\cdot)$ and $w(\cdot)$ with the constraint

$$x_r = w(v(x_r)), \tag{4.36}$$

which states that the projection from x_r to x and then x to x_r leads to the same point on the manifold.

With (4.36), we may immediately prove that for any $x = v(x_r)$, the distribution defined by $\text{span}(V(x))$ is involutive, and that the basis given by $V(x)$ is commuting.

Theorem 4.6.2 *If $x_r = w(v(x_r))$, then for all $x = v(x_r)$, $[v_i(x), v_j(x)] = 0$ for all i, j, $i \neq j$.*

Proof By definition,

$$v_i(x) = \left.\frac{\partial v}{\partial x_{r,i}}\right|_{x_r=w(x)}, \quad v_j(x) = \left.\frac{\partial v}{\partial x_{r,j}}\right|_{x_r=w(x)}. \tag{4.37}$$

Therefore,

$$\begin{aligned}[v_i, v_j] &= \frac{\partial v_j}{\partial x} v_i - \frac{\partial v_i}{\partial x} v_j \\ &= \frac{\partial^2 v}{\partial x_r \partial x_{r,j}} \frac{\partial w}{\partial x} \frac{\partial v}{\partial x_{r,i}} - \frac{\partial^2 v}{\partial x_r \partial x_{r,i}} \frac{\partial w}{\partial x} \frac{\partial v}{\partial x_{r,j}}.\end{aligned} \tag{4.38}$$

Take the derivative of $x_r = w(v(x_r))$ with respect to x_r, we obtain

$$I = \frac{\partial w}{\partial x} \frac{\partial v}{\partial x_r}, \quad \forall x = v(x_r), \tag{4.39}$$

and therefore

$$\frac{\partial w}{\partial x} \frac{\partial v}{\partial x_{r,i}} = e_i. \tag{4.40}$$

Therefore, (4.38) simplifies to

$$\begin{aligned}[v_i, v_j] &= \frac{\partial^2 v}{\partial x_r \partial x_{r,j}} e_i - \frac{\partial^2 v}{\partial x_r \partial x_{r,i}} e_j \\ &= \frac{\partial}{\partial x_{r,j}} \left(\frac{\partial v}{\partial x_{r,i}} \right) - \frac{\partial}{\partial x_{r,i}} \left(\frac{\partial v}{\partial x_{r,j}} \right) \\ &= 0 \end{aligned} \tag{4.41}$$

■

While we haven't shown that $V(x)$ is involutive for all $x \in \mathbb{R}^N$, we see that by definition, we have an integral manifold $\mathcal{V}$ that satisfies $T_{x_r}\mathcal{V} = \text{span}(V(x))$. Therefore, the projections $v(\cdot)$ and $w(\cdot)$ lead to a well-defined integral manifold.

However, in practice, it can still be hard or impossible to construct analytical expressions of the nonlinear functions $v(\cdot)$ and $w(\cdot)$ that satisfy (4.36), except for a few special cases.

Example 4.6.3 *Consider linear functions*

$$x = v(x_r) = x_0 + V x_r, \quad x_r = w(x) = x_{r,0} + W^T x, \tag{4.42}$$

where $x_0 \in \mathbb{R}^N$, $x_{r,0} \in \mathbb{R}^{N_r}$, $V \in \mathbb{R}^{N \times N_r}$ and $W \in \mathbb{R}^{N \times N_r}$.

The condition (4.36) for (4.42) translates to

$$x_r = x_{r,0} + W^T(x_0 + V x_r) = x_{r,0} + W^T x_0 + W^T V x_r \tag{4.43}$$

For (4.43) to hold for all x_r, we must have

$$\begin{aligned} x_{r,0} + W^T x_0 &= 0, \\ W^T V &= I. \end{aligned} \tag{4.44}$$

Therefore, given V, x_0, one of the many choices of W^T and $x_{r,0}$ is

$$\begin{aligned} W^T &= (V^T V)^{-1} V^T, \\ x_{r,0} &= -W^T x_0. \end{aligned} \tag{4.45}$$

Example 4.6.4 *Consider a nonlinear function $v(\cdot)$ and a linear function $w(\cdot)$:*

$$x = v(x_r) = [V, V_g][x_r; g(x_r)] = V x_r + V_g g(x_r), \quad x_r = w(x) = W^T x, \tag{4.46}$$

where $g(x_r) \in \mathbb{R}^k$, $V \in \mathbb{R}^{N \times N_r}$, $V_g \in \mathbb{R}^{N \times k}$, $W^T \in \mathbb{R}^{N_r \times N}$, with k being the number of nonlinear functions in $g(x_r)$.

In this case, (4.36) becomes

$$x_r = W^T(Vx_r + V_g g(x_r)). \tag{4.47}$$

Define W such that $W^TV = I$[4]. We obtain

$$W^T V_g g(x_r) = 0. \tag{4.48}$$

A sufficient condition for (4.48) to hold is $W^TV_g = 0$, or in another word, columns of V_g are orthogonal to columns of W.

For example, consider

$$x = v(x_r) = [x_{r,1}, x_{r,2}, x_{r,1}x_{r,2}], \quad x_r = w(x) = W^T x = [x_1, x_2]. \tag{4.49}$$

In (4.49), we define the manifold by $v(\cdot)$ which is the surface defined by $x_1 = x_{r,1}, x_2 = x_{r,2}, x_3 = x_{r,1}x_{r,2}$. On the other hand, we define the projection of each point x to the manifold by $w(\cdot)$: for any point (x_1, x_2, x_3), the projection leads to the point $v(w(x)) = (x_1, x_2, x_1x_2)$.

(4.48) corresponds to $V = W = [I_2; 0]$, $V_g = e_3$. As a result, $W^TV_g = 0$. Therefore, $V(x)$ defines an integral manifold.

Note also that in this example, the manifold (defined by $v(\cdot)$) is nonlinear, but the projection (defined by $w(\cdot)$) is linear.

Example 4.6.5 *Consider a linear function $v(\cdot)$ and a nonlinear function $w(\cdot)$:*

$$x = v(x_r) = Vx_r, \quad x_r = w(x) = W^T x + W_g^T g(x). \tag{4.50}$$

In this case, (4.36) becomes

$$x_r = W^T V x_r + W_g^T g(Vx_r). \tag{4.51}$$

Again, if we define W such that $W^TV = I$[5], we have

$$W_g^T g(Vx_r) = 0. \tag{4.52}$$

To simplify the problem, we further assume $g(x)$ are quadratic functions, i.e., $g(x) = x \otimes x$. Then,

$$W_g^T(Vx_r) \otimes (Vx_r) = W_g^T(V \otimes V)(x_r \otimes x_r) = 0, \tag{4.53}$$

for which a sufficient condition is

$$W_g^T(V \otimes V) = 0. \tag{4.54}$$

[4]This might not be the best choice, though.

[5]This might also not be a good choice.

For example, consider

$$V = [1; 0; 0], \quad W = [1; 0; 0]. \tag{4.55}$$

We then have

$$V \otimes V = [1; 0; 0; 0; 0; 0; 0; 0; 0], \tag{4.56}$$

and a valid choice of W_g^T *is*

$$W_g^T = [0, 0, 0, 0, 0, 0, 0, 0, 1]. \tag{4.57}$$

These choices correspond to

$$\begin{aligned} x &= v(x_r) = [x_r; 0; 0] \\ x_r &= w(x) = x_1 + x_3^2 \end{aligned} \tag{4.58}$$

In this case, the manifold is the x_1 *axis (a linear subspace), but the projection of any point to the* x_1 *axis is the parabola* $x_1 + x_3^2 = 0$ *(nonlinear).*

4.6.3 Method 2

In method 2, we start with an involutive and commuting basis $\tilde{V}(x)$, from which we induce the mapping $v(\cdot)$ by solving

$$\frac{dx}{dx_r} = \tilde{V}(x). \tag{4.59}$$

Specifically, the commuting basis $\tilde{V} = [\tilde{v}_1, \cdots, \tilde{v}_{N_r}]$ provides a natural definition of $v(\cdot)$ that also serves as a parameterization of the manifold

$$v : (x_{r,1}, \cdots, x_{r,N_r}) \mapsto \exp(x_{r,1}\tilde{v}_1) \circ \cdots \circ \exp(x_{r,N_r}\tilde{v}_{N_r}), \tag{4.60}$$

where $\exp(x_{r,i}\tilde{v}_i)$ is the solution to

$$\frac{dx}{dx_{r,i}} = \tilde{v}_i(x). \tag{4.61}$$

Since columns of $\tilde{V}(x)$ commute, we can change the order of the integration without changing the mapping. With $\tilde{V}(x)$, we can also induce a $W(x)$ by ensuring $W^T(x)\tilde{V}(x) = I$.

In this method, we have not explicitly defined the projection $x_r = w(x)$ although it may be induced by solving $\frac{dw(x)}{dx} = W^T(x)$. However, we do not have to compute $w(x)$ for the purpose of reduced order modeling – $W^T(x)$ alone is enough to construct a reduced order model.

One way to find such a $\tilde{V}(x)$ that also preserves system responses (such as moments of linearized models) is to start with a non-involutive and non-commuting "accurate" basis

(*e.g.*, the Krylov basis), and then try to find a "closest" distribution to it.

Another way is to consider specific basis functions, and derive simpler conditions for involutivity and commutativity.

Example 4.6.6 *Consider* $V(x) = [v_1, \cdots, v_{N_r}]$ *where* $v_i(x)$ *is in the form of*

$$v_i(x) = A_i x + b_i, \quad i = 1, \cdots, N_r, \tag{4.62}$$

where $A_i \in \mathbb{R}^{N\times N}$ *and* $b_i \in \mathbb{R}^{N\times 1}$.

Applying the involutivity constraint in (4.35) on the pair (v_i, v_j)*, we obtain*

$$\begin{aligned}[v_i, v_j] &= A_j(A_i x + b_i) - A_i(A_j x + b_j) \\ &= (A_j A_i - A_i A_j)x + (A_j b_i - A_i b_j) = 0.\end{aligned} \tag{4.63}$$

In order for (4.63) to be equal to 0 for all x*, we must have*

$$\begin{aligned}A_j A_i - A_i A_j &= 0, \\ A_j b_i - A_i b_j &= 0.\end{aligned} \tag{4.64}$$

For simplicity, consider

$$A_i = U\Lambda_i U^{-1}, \tag{4.65}$$

where columns of U *are the eigenvalues of* A *and* Λ_i *is diagonal with eigenvalues of* A_i *on the diagonal.*

A_i*'s defined by (4.65) are commuting:*

$$A_i A_j = U\Lambda_i U^{-1} U\Lambda_j U^{-1} = U\Lambda_i\Lambda_j U^{-1} = U\Lambda_j\Lambda_i U^{-1} = U\Lambda_j U^{-1} U\Lambda_i U^{-1} = A_j A_i. \tag{4.66}$$

We also obtain b_j *in terms of* b_i*:*

$$b_j = A_i^{-1} A_j b_i = U\Lambda_i^{-1} U^{-1} U\Lambda_j U^{-1} b_i = U\Lambda_i^{-1}\Lambda_j U^{-1} b_i. \tag{4.67}$$

4.6.4 Method 3

In method 3, we start with an arbitrary distribution defined by $U(x)$ that may be non-involutive or non-commuting. However, we can still define a mapping $v : x_r \mapsto x$ by

$$v : (x_{r,1}, \cdots, x_{r,N_r}) \mapsto \exp(x_{r,N_r} u_{N_r}) \cdots \exp(x_{r,1} u_1). \tag{4.68}$$

In another word, the mapping is obtained by integrating

$$\frac{dx}{dx_{r,i}} = u_i(x) \tag{4.69}$$

in a specified order ($i = 1, \cdots, N_r$).

From this mapping, we may then derive the distribution

$$V(x) = \left.\frac{dv(x_r)}{dx_r}\right|_{x_r=w(x)}, \tag{4.70}$$

by defining a projection $x_r = w(x)$.

In fact, the ManiMOR approach to be introduced in Chapter 5 uses this method to construct the manifold.

4.7 Reduced Model Data Structure and Computation

The storage of mapping functions $x = v(x_r)$, $x_r = w(x)$, or the associated distributions $V(x)$ and $W^T(x)$ can be difficult and expensive. Moreover, to compute the reduced model, we ultimately need to store and compute

$$f_r(x_r) = W^T(v(x_r))f(v(x_r)), \tag{4.71}$$

which may be even more complicated.

4.7.1 Symbolic/Analytical Representation

If $v(x_r)$, $w(x)$ have analytical expressions, we may simply store their symbolic expression. In particular, some special functions even have a matrix representation, such as quadratic functions

$$v(x_r) = G_1 x_r + G_2 x_r \otimes x_r, \tag{4.72}$$

for which we may just store the matrices G_1 and G_2.

Moreover, analytical expressions of $v(\cdot)$ and $w(\cdot)$ may help simplify (4.71). In the linear projection framework, we have

$$f_r(x_r) = W^T G V x_r, \tag{4.73}$$

where $W^T G V \in \mathbb{R}^{N_r \times N_r}$ can be precomputed and stored, and is N-independent. We may do the similar trick in the nonlinear case.

Example 4.7.1 *Consider the case where*

$$\begin{aligned} v(x_r) &= V_1 x_r + V_2 x_r \otimes x_r, \\ f(x) &= G_1 x + G_2 x \otimes x, \\ W^T(x) &= W^T. \end{aligned} \tag{4.74}$$

We obtain

$$\begin{aligned} f_r(x_r) =& W^T \left(G_1(V_1x_r + V_2x_r \otimes x_r) + G_2(V_1x_r + V_2x_r \otimes x_r) \otimes (V_1x_r + V_2x_r \otimes x_r)\right) \\ =& W^T G_1 V_1 x_r + W^T G_1 V_2 x_r \otimes x_r \\ &+ W^T G_2[(V_1 \otimes V_1)(x_r \otimes x_r) + (V_1 \otimes V_2)(x_r \otimes x_r \otimes x_r) \\ &+ (V_2 \otimes V_1)(x_r \otimes x_r \otimes x_r) + (V_2 \otimes V_2)(x_r \otimes x_r \otimes x_r \otimes x_r)] \\ =& W^T G_1 V_1 x_r + (W^T G_1 V_2 + W^T G_2(V_1 \otimes V_1))(x_r \otimes x_r) \\ &+ W^T G_2(V_1 \otimes V_2 + V_2 \otimes V_1)(x_r \otimes x_r \otimes x_r) \\ &+ W^T G_2(V_2 \otimes V_2)(x_r \otimes x_r \otimes x_r \otimes x_r), \end{aligned} \tag{4.75}$$

where we only have N-independent matrices to pre-compute and store in the reduced model.

4.7.2 Piecewise Linear Approximation

When no analytical expression is available, we have to use certain approximation for the nonlinear mappings and nonlinear functions. One such approximation is PWL approximation[6], *i.e.*, concatenating (interpolating) point-wise linearized functions to obtain a global function.

Approximating mapping functions

In PWL approximation, we assume that, instead of the actual nonlinear function $x = v(x_r)$, we are given a few pairs of $(x_{r,i}, x_i), i = 1, \cdots, N_s$ where N_s is the number of samples on the manifold. Therefore, we can approximate $v(\cdot)$ by

$$x = \sum_{i=1}^{N_s} K_i(x_r) x_{r,i}, \tag{4.76}$$

where $K_i(x_r)$ is a weighting (kernel) function that satisfies

$$\begin{aligned} K_i(x_r) &\geq 0, \\ \sum_{i=1}^{N_s} K_i(x_r) &= 1, \\ K_i(x_{r,i}) &= 1. \end{aligned} \tag{4.77}$$

When we also have information about the tangent space (*e.g.*, Krylov subspaces) at each

[6]While the method is traditionally called "piecewise-linear" in MOR community, it is also known as a special case of kernel methods.

point, *i.e.*,

$$V_i = \left.\frac{\partial v}{\partial x_r}\right|_{x=v(x_{r,i})}, \tag{4.78}$$

we may obtain a better PWL approximation of $v(\cdot)$ by

$$x = \sum_{i=1}^{N_s} K_i(x_r)(V_i(x_r - x_{r,i}) + x_i). \tag{4.79}$$

Similarly, we may approximate $w(\cdot)$ by

$$x_r = \sum_{i=1}^{N_s} K_i(x_r)(W_i^T(x - x_i) + x_{i,r}). \tag{4.80}$$

Approximating nonlinear functions

For PWL approximations of nonlinear functions, we assume that we are given $f_i = f(x_i)$ and $G_i = \frac{\partial f}{\partial x}(x_i)$ for each pair $(x_i, x_{r,i})$. Therefore, the PWL approximation of $f_r(x_r)$ is

$$f_r(x_r) \simeq W^T(x_r) \sum_{i=1}^{N_s} K_i(x_r) \left(f_i + G_i(v(x_r) - x_i)\right). \tag{4.81}$$

Combine (4.81) and (4.79), we obtain

$$\begin{aligned} f(x_r) \simeq & W^T(x_r) \sum_{i=1}^{N_s} K_i^f(x_r) \left(f_i + G_i(\sum_{j=1}^{N_s} K_j^v(x_r)(V_j(x_r - x_{r,j}) + x_j) - x_i) \right) \\ = & W^T(x_r) \sum_{i=1}^{N_s} K_i^f(x_r) \Big(f_i + K_i^v(x_r) G_i V_i(x_r - x_{r,j}) + G_i(K_i^v(x_r) x_i - x_i) \\ & + \sum_{j=1, j\neq i}^{N_s} K_j^v(x_r)(V_j(x_r - x_{r,j}) + x_j) - x_i) \Big). \end{aligned} \tag{4.82}$$

where $K_i^f(\cdot)$ and $K_i^v(\cdot)$ are the weighting functions for $f(\cdot)$ and $v(\cdot)$, respectively.

Note that when x_r is close to $x_{r,i}$, $K_i(x_r) \simeq 1$ and $K_j(x_r) \simeq 0, j \neq i$. Therefore, we have

$$f_r(x_r) \simeq W^T(x_r)(f_i + G_i V_i(x_r - x_{r,i})). \tag{4.83}$$

Therefore, the PWL approximation of $f(\cdot)$ is often simplified to

$$f_r(x_r) \simeq W^T(x_r) \sum_{i=1}^{N_s} K_i(x_r) \left(f_i + G_i V_i (x_r - x_{r,i})\right), \tag{4.84}$$

which can be viewed as approximately truncating second-order terms of $K_i^f(x_r) K_j^v(x_r)$.

To approximate $W^T(x_r)$, we differentiate (4.80) and obtain

$$W^T(x_r) = \left.\frac{\partial w}{\partial x}\right|_{x=v(x_r)}. \tag{4.85}$$

Similarly, a PWL approximation of $W^T(x_r)$ is

$$W^T(x_r) \simeq \sum_{i=1}^{N_s} w_i(x_r) W_i^T. \tag{4.86}$$

Combine (4.84) with (4.86), and use a simplified PWL approximation, we obtain

$$f_r(x_r) \simeq \sum_{i=1}^{N_s} K_i(x_r)(W_i^T f_i + W_i^T G_i V_i (x_r - x_{r,i})). \tag{4.87}$$

Therefore, we can pre-compute and store $W_i^T f_i \in \mathbb{R}^{N_r}$ and $W_i^T G_i V_i \in \mathbb{R}^{N_r \times N_r}$ in the reduced model. As a result, the computation cost of the reduced model (*i.e.*, (4.87)) is depends only on N_r and N_s, and is N-independent.

Approximating Jacobians

Taking the derivative of (4.87) with respect to x_r, we obtain the Jacobian

$$\frac{\partial f_r}{\partial x_r} \simeq \sum_{i=1}^{N_s} K_i(x_r) W_i^T G_i V_i + \sum_{i=1}^{N_s} \frac{\partial K_i(x_r)}{\partial x_r} \left[W_i^T f_i + W_i^T G_i V_i (x_r - x_{r,i})\right]. \tag{4.88}$$

Note that when $\frac{\partial K_i(x_r)}{\partial x_r}$ is small (which is true for x_r far away from x_i), we can further ignore the second part in (4.88), and obtain

$$\frac{\partial f_r}{\partial x_r} \simeq \sum_{i=1}^{N_s} K_i(x_r) W_i^T G_i V_i. \tag{4.89}$$

4.7.3 Reduced Order Model Generation

To generate the reduced order model, we pre-compute and store the reduced matrices/vectors (*e.g.*, those used in (4.87)) in a look-up table. They are summarized in Table 4.1.

Table 4.1: Look-up table stored in the PWL-based reduced model

Variable	Expression	Size
f_{r_i}	$W_i^T f(x_i)$	$\mathbb{R}^{N_r \times 1}$
q_{r_i}	$W_i^T q(x_i)$	$\mathbb{R}^{N_r \times 1}$
B_{r_i}	$W_i^T B$	$\mathbb{R}^{N_r \times 1}$
G_{r_i}	$W_i^T \frac{\partial f}{\partial x}(x_i) V_i$	$\mathbb{R}^{N_r \times N_r}$
C_{r_i}	$W_i^T \frac{\partial q}{\partial x}(x_i) V_i$	$\mathbb{R}^{N_r \times N_r}$

Therefore, if these is N_s samples, the time and space complexity of the reduced model (*i.e.*, computation and storage of (4.87)) is $O(N_s N_r^2)$ – it is N-independent and therefore is more efficient than computing the full model given $N_r \ll N$ and $N_s \ll N$.

Furthermore, instead of using all N_s samples for the computation at a specific point x_r, we may use k-nearest neighbors of x_r in the computation of (4.87) because the weight function $w_i(x_r)$ evaluates approximately to 0 when x_r is far away from x_i. Therefore, the computational complexity is further reduced to $O(kN_r^2)$ plus the complexity of computing the k-nearest neighbors.

Chapter 5

ManiMOR: General-Purpose Nonlinear MOR by Projection onto Manifolds

In this chapter, we present a general-purpose nonlinear MOR technique based on the nonlinear projection framework presented in Chapter 4. The algorithm explicitly constructs and parameterizes a manifold in the state space, and projects the original nonlinear model to the manifold to obtain a nonlinear reduced order model.

We describe in detail how we construct and parameterize a general-purpose manifold that captures typical system responses (such as DC and AC responses). We validate and illustrate the MOR technique using several examples.

5.1 Construction of the Manifold

Many dynamical systems derived from engineering problems work in certain highly constrained region in the state space, either because the fast dynamics die out very quickly and are not easily observable, or because the systems are designed/engineered to work under certain mode (*e.g.*, amplifiers are designed to work around a DC steady state / equilibrium, and oscillators are designed to work around a limit cycle). In another word, state variables typically stay close to the attractors, and that part of the state space covers important dynamics. In order for the reduced model to reproduce the attractors, a necessary condition is that the manifold must cover the attractors.

5.1.1 DC Manifold

We first consider the manifold covering equilibria (also known as DC operating points in circuit community) of a system.

For the nonlinear system (1.2), the DC equilibria are the solution to

$$f(x) + Bu = 0, \quad u \in [u_{min}, u_{max}], \tag{5.1}$$

where u_{min} and u_{max} are bounds of the input u according to physical constraints.

For simplicity, we consider the case where $f(\cdot)$ is invertible. We consider this assumption because it is often satisfied in a large number of circuit examples.

As a result of this assumption, we have

$$x = f^{-1}(-Bu), \tag{5.2}$$

which defines a one-dimensional manifold (parameterized by u) consisting of all DC solutions. This constitutes the first dimension of the manifold, and we call it **DC manifold**.

Equivalently, the solution of (5.2) can be written as the solution to

$$\frac{\partial x}{\partial u} = -\frac{\partial f^{-1}}{\partial b}(-Bu)B, \quad x(u=0) = f^{-1}(0), \tag{5.3}$$

where $\frac{\partial f^{-1}}{\partial b}(\cdot)$ is the Jacobian of $f^{-1}(\cdot)$. (5.3) is obtained by taking the partial derivative of (5.2) with respect to u.

Similarly, we may directly take the partial derivative of (5.1) with respect to u, and obtain

$$G(x)\frac{\partial x}{\partial u} + B = 0, \tag{5.4}$$

and therefore we can define

$$v_1(x) = \frac{\partial x}{\partial u} = -G^{-1}(x)B, \tag{5.5}$$

which associate each point on the DC manifold with its tangent space.

The RHS of (5.5) is exactly the direction of the first Krylov vector of the linearized model around x. Therefore, as proved in [89], at each DC equilibrium, the linearized reduced model matches the first moment of the linearized full model.

5.1.2 AC manifold

The AC response is determined by the linearized models around DC solutions. To reduce these linearized models, we may use any LTI MOR method described in Section 2.3.

Specifically, for each linearized model, the first dimension of the reduced subspace is determined by the DC manifold, *i.e.*, the first Krylov vector, Krylov-subspace based methods seem to be appropriate, and can be used to compute the tangent spaces at each point on the manifold.

Using Arnoldi-based method, for example, we have that at each point x, the local linear subspace, or the tangent space of the manifold, is defined by the column span of a nonlinear

projection matrix $V_{N_r}(x)$ that satisfies

$$\text{span}(V_{N_r}(x)) = \text{span}(G^{-1}(x)B, G^{-2}(x)B, \cdots, G^{-N_r}(x)B). \tag{5.6}$$

Because the tangent space around a point x gives a good local approximation of the manifold, the points close to x is, loosely speaking, also on the manifold. Therefore, one may "grow" the manifold by integrating along the Krylov vectors to obtain the manifold. That is, solving

$$\frac{\partial x}{\partial x_r} = V_{N_r}(x) \tag{5.7}$$

to obtain the projection $x = v(x_r)$.

However, according to the discussion in Section 4.4, in general there may not exist a solution to (5.7). Nevertheless, we can always ensure that at each point on the DC manifold, (5.7) holds (one example is given in Section 5.1.3. We will call any manifold that satisfies this condition an **AC manifold**, because such manifolds lead to reduced order models that match up to the N_r-th moment of the linearized full models, and therefore the AC responses are well approximated.

5.1.3 Manifold used in ManiMOR

While we do not have a manifold whose tangent spaces match local Krylov subspaces, we can construct manifolds that cover DC and AC manifolds. In the following, we provide one such solution by constructing parametric equations of the manifold in terms of N_r variables $x_r = (x_{r,1}, \cdots, x_{r,N_r})$.

Let $V_{N_r}(x) = [v_1(x), \cdots, v_{N_r}(x)]$ be the Krylov basis of the linearized system around x, and let $x_0 = f^{-1}(0)$ be the DC solution for $u = 0$. We "grow" the manifold $\mathcal{V}$ starting from x_0:

1. Solve for $x = v(x_{r,1}, 0, \cdots, 0)$ by integrating

$$\frac{\partial x}{\partial x_{r,1}} = v_1(x), \quad x(0) = x_0. \tag{5.8}$$

2. Solve for $x = v(x_{r,1}, x_{r,2}, 0, \cdots, 0)$ by integrating

$$\frac{\partial x}{\partial x_{r,2}} = v_2(x), \quad x(0) = x(x_{r,1}, 0, \cdots, 0). \tag{5.9}$$

3. $\cdots$

4. Solve for $x = v(x_{r,1}, x_{r,2}, \cdots, x_{r,N_r})$ by integrating

$$\frac{\partial x}{\partial x_{r,N_r}} = v_{N_r}(x), \quad x(0) = x(x_{r,1}, \cdots, x_{r,N_r-1}, 0). \tag{5.10}$$

In short, the resulting projection function can then be written as

$$x = v(x_r) = \exp(V_{N_r}(x)x_r) = \exp(v_{N_r}(x)x_{r,N_r}) \circ \cdots \circ \exp(v_1(x)x_{r,1}). \tag{5.11}$$

In another word, the manifold is composed of points that are solutions of (5.11).

The rationale behind the construction of such a manifold is that at a point x in the state space, the state variable can move in N independent directions, corresponding to N modes which are approximated by the N Krylov basis vectors. Ignoring $N - N_r$ "fast" modes, the state variable will first move along the N_r-th direction (the fastest remaining mode), and then the $(N_r - 1)$-th direction (the second fastest remaining mode), and so on. In the end, it moves along the first direction (the slowest mode). Therefore, at each point on the N_r-dimensional manifold, we only guarantee that $v_{N_r}(x)$ is covered by the local tangent space.

This leads to the conclusion that at each point on the k-dimensional manifold, the local tangent space covers $\text{span}(v_k(x), \cdots, v_{N_r}(x))$. In particular, at each DC equilibrium, the local tangent space covers the Krylov subspace of the linearized model. Therefore, the resulting nonlinear model covers the DC and AC manifolds.

5.2 Parameterization

(5.11) gives a straightforward parameterization of the manifold using x_r.

While the order of the integration from (5.8) to (5.10) is extremely important, we may change the magnitude of each Krylov vector without changing the manifold. Hence, we may obtain many parameterizations of the same manifold. This can be seen from the following theorem.

Theorem 5.2.1 *Suppose $x(t)$ and $\hat{x}(\tau)$ are the solutions to*

$$\frac{d}{dt}x(t) = g(x(t)) \tag{5.12}$$

and

$$\frac{d}{d\tau}\hat{x}(\tau) = \sigma'(\tau)g(\hat{x}(\tau)), \tag{5.13}$$

respectively, where $t = \sigma(\tau)$ is a function of τ, and $g(\cdot)$ is Lipschitz. Then $x(t)$ and $\hat{x}(\tau)$ define the same points in the state space, i.e.$\forall t, \exists \hat{t}$, such that $\hat{x}(\hat{t}) = x(t)$.

Proof Since $t = \sigma(\tau)$, we have

$$dt = \sigma'(\tau)d\tau \tag{5.14}$$

Define

$$\hat{x}(\tau) \equiv x(t) = x(\sigma(\tau)). \tag{5.15}$$

Therefore,

$$\frac{d}{d\tau}\hat{x}(\tau) = \frac{d\hat{x}(\tau)}{dt}\frac{dt}{d\tau} = g(x(t))\sigma'(\tau) = g(\hat{x}(\tau))\sigma'(\tau), \tag{5.16}$$

i.e.,

$$\frac{d}{d\tau}\hat{x}(\tau) = \sigma'(\tau)g(\hat{x}(\tau)). \tag{5.17}$$

According to the existence and uniqueness theorem for ordinary differential equations, $x(t)$ and $\hat{x}(\tau)$ are the unique solutions to (5.12) and (5.17), respectively.

Therefore, $x(t)$ and $\hat{x}(t)$ define the same points in the state space. ■

Based on theorem 5.2.1, the following corollary follows.

Corollary 5.2.2 *The solutions to the normalized integral curve equation*

$$\frac{\partial x}{\partial x_{r,i}} = \frac{v_i(x)}{||v_i(x)||} \tag{5.18}$$

define the same trajectory as the solutions to the regular integral curve equation

$$\frac{\partial x}{\partial x_{r,i}} = v_i(x). \tag{5.19}$$

Proof Suppose the solutions to (5.19) and (5.18) are $x(t)$ and $\hat{x}(\hat{t})$, respectively. Let

$$t = \sigma(\hat{t}) = \int_0^{\hat{t}} \frac{1}{||v_i(\hat{x}(\mu))||} d\mu. \tag{5.20}$$

By theorem 5.2.1, $x(t)$ and $\hat{x}(\hat{t})$ span the same state space. ■

From the above discussion, we conclude that different choices of $\sigma(t)$ leads to different parameterizations of the manifold. Nevertheless, they all define the same manifold. Therefore, we may define $\sigma(t)$ to obtain better numerical accuracy in computing the manifold and the projection function.

5.3 Projection Functions and Matrices

To connect ManiMOR to the nonlinear projection framework, we examine the projection functions $x = v(x_r)$, $x_r = w(x)$, and the projection matrices $V(x_r)$ and $W^T(x)$ in ManiMOR.

Essentially, we have used method 3 from Section 4.6.4 in ManiMOR to derive the projections. In Section 4.6.4, $v(x_r)$ is given by (4.68), from which we may derive $V(x_r)$ by (4.70). With $V(x_r)$, we may define $W^T(x_r)$ such that $W^T(x_r)V(x_r) = I$, *e.g.*, by $W^T(x_r) = (V^T(x_r)V(x_r))^{-1}V^T(x_r)$.

5.3.1 Numerical Computation

It can be very expensive to use (4.68), or (5.11), to compute the projection functions and the reduced nonlinear models, because they themselves involve solving a series of differential equations.

For example, to compute $V(x) = \frac{\partial x}{\partial x_r}$, we have to solve

$$\frac{\partial x}{\partial x_{r,N_r}} = v_{N_r}(x), \quad \frac{dx}{dx_{r,N_r-1}} = \frac{\partial x}{\partial x_{0,x_{r,N_r-1}}} v_{N_r-1}(x), \quad \frac{dx}{dx_{r,N_r-2}} = \frac{\partial x}{\partial x_{0,x_{r,N_r-2}}} v_{N_r-2}(x), \cdots, \tag{5.21}$$

where $x_{0,x_{r,i}} = x(x_{r,1}, \cdots, x_{r,i}, 0, \cdots, 0)$, and the computation of RHS in (5.21) involves expensive transient sensitivity analysis.

To alleviate the computational cost of the reduced model, we employ the piece-wise linear approximation method described in Section 4.7.2 that makes the time and space complexity $O(N_s N_r^2)$.

5.4 Summary and Algorithm

As a summary, we present the complete ManiMOR algorithm in Algorithm 5.

Algorithm 5 ManiMOR

Inputs: The full model $\frac{dx}{dt} = f(x) + Bu$. The reduced order model size N_r.
Outputs: $\frac{dx_r}{dt} = f_r(x_r) + B_r u$. A look-up table storing discrete points of x, x_r, f_r, B_r, *etc.*.

1: Solve (4.68) for discrete points on the manifold and its parameterization $(x_i, x_{r,i})$.
2: For each x_i, compute $V(x_r)$ by (4.70), and $W^T(x_r)$ by $(V^T(x_r)V(x_r))^{-1}V^T(x_r)$.
3: Compute $f_{r,i}, q_{r,i}, G_{r,i}, C_{r,i}, B_r$ as described in Section 4.7.2.
4: Store $f_{r,i}, q_{r,i}, G_{r,i}, C_{r,i}, B_r$ in a look-up table.

5.4.1 Connections to Previous Methods

Connection to Trajectory-Based Methods

In trajectory-based methods, the trajectories are sampled to identify the region in the state space where main dynamics of the system evolve. However, trajectory-based methods embeds these trajectories in a linear subspace, instead of a manifold as is done in ManiMOR. This is the key difference between ManiMOR and trajectory-based methods.

Because of this key difference, trajectory-based methods usually generate a model of much larger size than that of ManiMOR. The simplest example is a nonlinear system which has different AC modes (corresponding to local subspaces) when operating under different DC equilibria. Suppose that the dynamics of each linearized system along the equilibria could be approximately embedded in a q-dimensional linear subspace, then for the entire system, a q-dimensional nonlinear manifold is adequate to embed all the q-dimensional linear subspaces, while a much larger linear subspace is needed to include all the q-dimensional linear subspaces.

Connection to TP-PPV

For oscillator circuits, the phase macromodel is more useful for understanding the oscillator behavior and performing phase noise analysis. Among many available phase macromodels, the PPV macromodel [29] stands out to capture many interesting behaviors of oscillators when there is small perturbation to the oscillator. However, it is not directly applicable to large injection signal, and to address that problem, TP-PPV [30] has been proposed which combines several PPV macromodel at several different DC inputs.

From a projection perspective, it is in fact a simplified version of ManiMOR for oscillators. As we have mentioned, the attractor for oscillators are the limit cycles. TP-PPV in fact computes the limit cycles for the oscillator circuit at different DC bias, and then stitching them together to obtain a cylinder-like manifold in the state space. In TP-PPV, the variable α and the DC input u parameterize this manifold, and the time-domain waveforms are restored by interpolation on this manifold given the $\alpha(t)$ and $u(t)$ waveforms. Therefore, it fits well into the nonlinear projection framework, except that the projection is implicitly done by formulating the PPV equation.

5.5 Examples and Experimental Results

In this section, we apply ManiMOR to several examples, including artificial simple nonlinear systems, practical circuits such as ring mixer, nonlinear transmission line and I/O buffer circuit, as well as a bio-chemical signaling pathway example. We validate the ManiMOR model by comparing the simulation results using ManiMOR model and the full model.

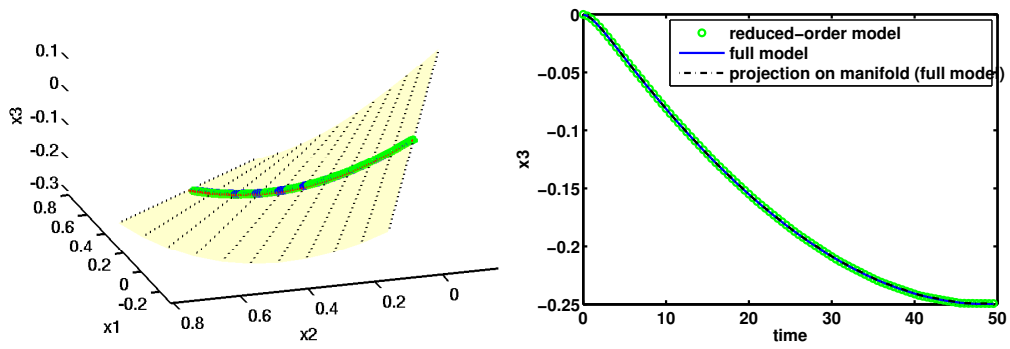
We show comparisons of ManiMOR to existing methods, mainly against the TPWL method, to examine the accuracy and efficiency of ManiMOR.

5.5.1 An Illustrative Nonlinear System

In the first example, we apply ManiMOR to the simple nonlinear system (4.2) mentioned in Section 4.1, in order to illustrate how ManiMOR works.

We apply the ManiMOR algorithm to generate a 2-dimensional reduced order model for this 3-dimensional system. We then perform transient analysis of both models and compare the results.

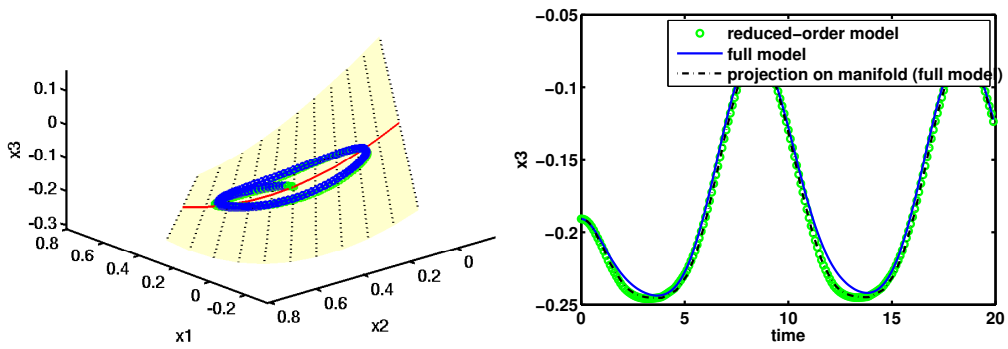
The simulation results obtained by applying a slowly changing input $u(t) = t$ are shown in Fig. 5.1. We observe that the trajectories of the reduced order model follow the DC manifold in the state space. The reduced model covers the DC manifold and therefore reproduces the slowest dynamics.



(a) state space (b) $x_3(t)$

Figure 5.1: Simulation of the ManiMOR model. ($u(t) = t$) Fig. 5.1(a) shows the manifold ManiMOR identifies (yellow), and the trajectories of the full model (blue) and ManiMOR model (green). Fig. 5.1(b) shows the waveform of $x_3(t)$ of the ManiMOR model (green), the full model (blue), and the projection of the full model solution onto the manifold (black).

The simulation results obtained by applying a sinusoidal signal $u(t) = 2.5 + 2sin(0.2\pi t)$ are shown in Fig. 5.2. Because the frequency of the input signal is low, the trajectory of the full model is nearly on the manifold. In this case, the trajectory stays close to the DC solution, but nonlinear distortion is exhibited in steady state solution. Since the manifold in ManiMOR is good in capturing behaviors around DC solutions, this type of distortion is well approximated by the ManiMOR model, as shown in Fig. 5.2. The maximum absolute and relative mean squared error of ManiMOR, compared to the full model, are 0.0136 and 0.0473, respectively.



(a) state space (b) $x_3(t)$

Figure 5.2: Simulation of the ManiMOR model. (sinusoidal input) Fig. 5.2(a) shows the DC curve (red) and the manifold ManiMOR identifies (yellow), the trajectories of the full model (blue) and ManiMOR model (green). Fig. 5.2(b) shows the waveform of $x_3(t)$ of the ManiMOR model (green), the full model (blue), and the projection of the full model solution onto the manifold (black).

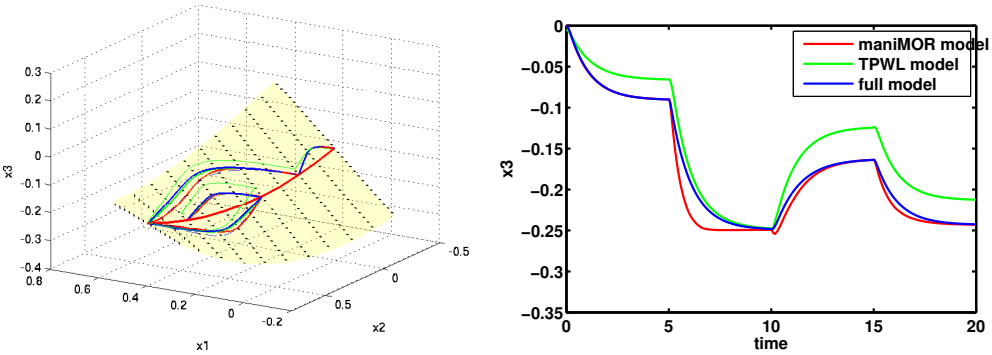
Fig. 5.3 and Fig. 5.4 show simulation results to compare the ManiMOR model and the TPWL model. Fig. 5.3 shows the simulation results obtained by applying a multiple-step function

$$u(t) = \begin{cases} 1, & t < 5, \\ 5, & 5 \le t < 10, \\ 2, & 10 \le t < 15, \\ 4, & 15 \le t \le 20. \end{cases} \tag{5.22}$$

We observe that the ManiMOR model tracks the trajectory of the full model "better" than TPWL. "Better" means that 1) the distance (error) between the trajectories of ManiMOR model and the full model is smaller than that of the TPWL model; 2) the ManiMOR model converges to the right DC solution, while the TPWL model fails to get back to the right operating point after transient response dies out, which is better observed in Fig. 5.3(b).

In this example, the maximum absolute mean square error of the ManiMOR model and TPWL model, compared to full model, are 0.0534 and 0.0627, respectively. Although the TPWL model only leads to an error that is slightly larger than the ManiMOR model, the trajectory of TPWL model fails to converge to the right DC operating point – this can cause serious problems when using reduced models because the attractor is not preserved. Such models may not be acceptable in practice.

Similar results are observed when a two-tone sinusoidal input $u(t) = 2.5 + sin(0.1\pi t) + cos(0.4\pi t)$ is applied, as shown in Fig. 5.4. Although the waveforms obtained using the TPWL model looks similar to those of the full model, there is an observable offset. In



(a) state space (b) $x_3(t)$

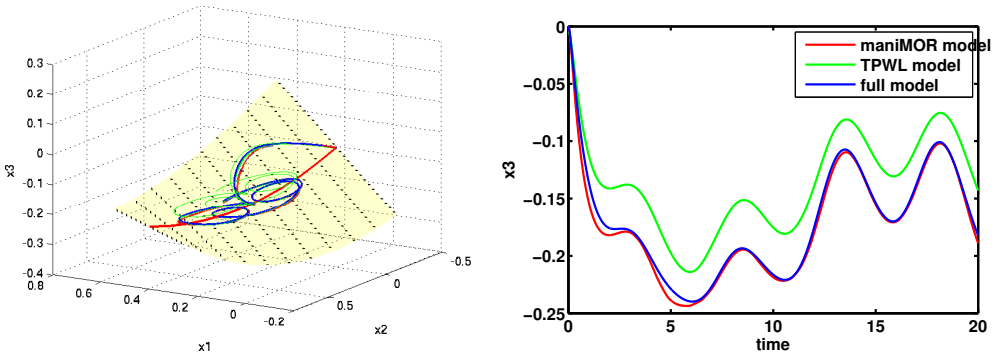
Figure 5.3: Comparison of ManiMOR and TPWL model. (multiple-step input) Red, green and blue trajectories represent simulation results of ManiMOR, TPWL and full model, respectively.

contrast, the ManiMOR model tracks the nonlinear dynamics correctly.

5.5.2 A CMOS Ring Mixer

The second example, a CMOS ring mixer [1] (shown in Fig. 5.5), is highly nonlinear. When the input changes, the DC operating point moves from one state to another along a highly nonlinear curve, as shown by the red curve in Fig. 5.6(a). The 2-D manifold identified by ManiMOR is also depicted in Fig. 5.6(a). Clearly the highly nonlinear DC curve is not on a linear subspace, and the 2-D manifold is extremely twisted. In fact, regions that are near two ends of the DC curve are relatively linear (the two linear spaces are almost orthogonal to each other), while the region in the middle shows strong nonlinearity, and smoothly connects two nearly-orthogonal linear subspaces. This again suggests that TPWL, which asserts that the solutions will be close to a linear subspace, will be inefficient to reduce the full model.

To examine how well ManiMOR captures dynamics and nonlinearities in this example, we apply a step input, and simulate the circuit using the full model and the ManiMOR model. The simulation results are shown in Fig. 5.6, where the trajectory starts from one DC operating point, goes out of the manifold, and finally converges to another DC operating point. Again, the ManiMOR model shows close match to the full model and qualitatively correct behaviors. The maximum absolute and relative mean square error in this case is 0.1293 and 0.0492, respectively.



(a) state space (b) $x_3(t)$

Figure 5.4: Comparison of ManiMOR and TPWL model. (two-tone sinusoidal input) Red, green and blue trajectories represent simulation results of ManiMOR, TPWL and full model, respectively.

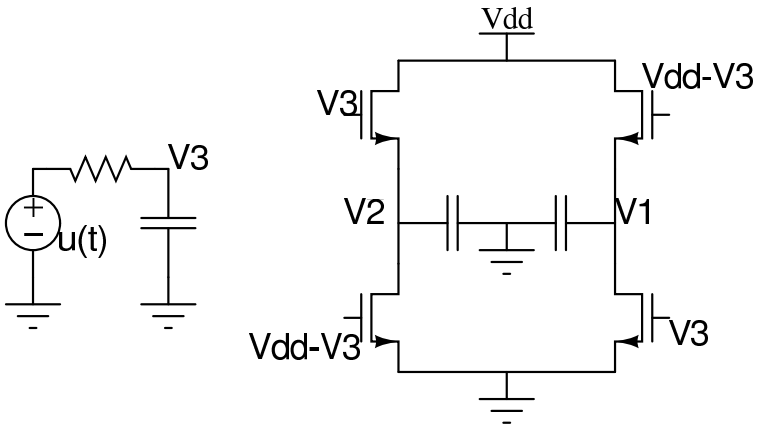
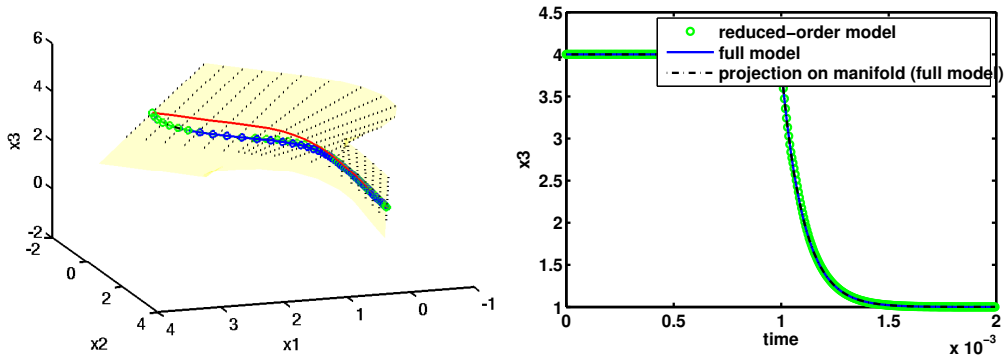


Figure 5.5: Circuit diagram of a CMOS ring mixer [1].



(a) state space (b) $x_3(t)$

Figure 5.6: Simulation of ManiMOR model and full model for the CMOS ring mixer. (step input) Fig. 5.6(a) shows the manifold ManiMOR identifies (yellow), and the trajectories of the full model (blue) and ManiMOR model (green). Fig. 5.6(b) shows the waveform of $x_3(t)$ of the ManiMOR model (green), the full model (blue), and the projection of the full model result onto the constructed manifold (black).

5.5.3 A Nonlinear Transmission Line Circuit

A test example that is examined in most papers about trajectory-based methods [2, 114] is a nonlinear transmission line circuit. To verify ManiMOR, we apply both ManiMOR and TPWL on the nonlinear transmission line circuit with 100 nodes.

Using the same training set and model size in [2] (*i.e.*, TPWL model is of order 10; training input is the step input $i(t) = u(t-3)$), we apply a test input of $i(t) = 0.8u(t-3)$ to the reduced models and the full model. As shown in Fig. 5.7, since the DC solution corresponding to $i(t) = 0.8$ is not trained, an observable error, compared to the full model, presents in the result of the TPWL model.

Accordingly, a ManiMOR model of order 4 is generated, and the results are shown in Fig. 5.7. We observe that (1) the error is less than that of TPWL model, even if the model size is almost halved; (2) despite the initial transient error, the trajectory converges to the exact steady state solution.

5.5.4 A CML Buffer Circuit

Another test circuit is a current-mode logic (CML) buffer chain [115], which has also been studied in previous MOR literatures [116, 117]. The circuit diagram of the CML buffer is shown in Fig. 5.8. We use the BSIM3 [118] model for MOSFETs in our simulations, and the size of differential algebraic equations for the full circuit is 52.

Applying ManiMOR, a size 5 model is generated for this circuit. The nonlinear manifold

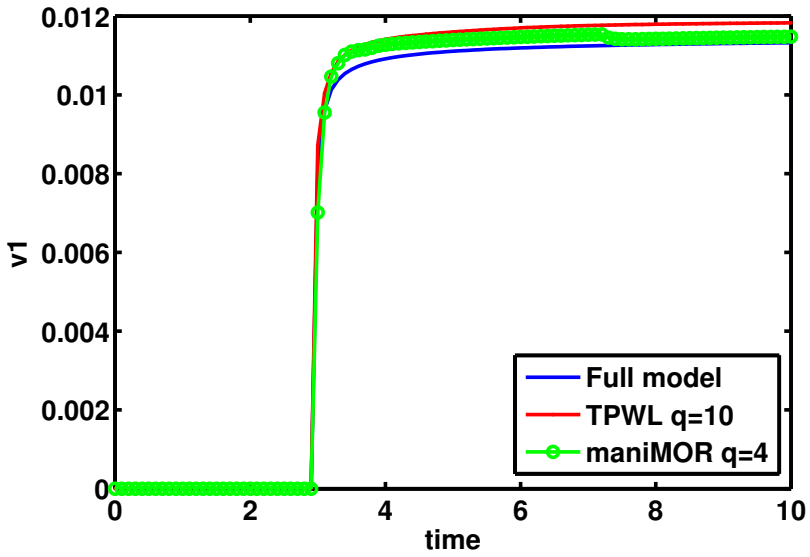


Figure 5.7: Comparison of of the circuit response of ManiMOR model and TPWL model. (Nonlinear transmission line circuit [2])

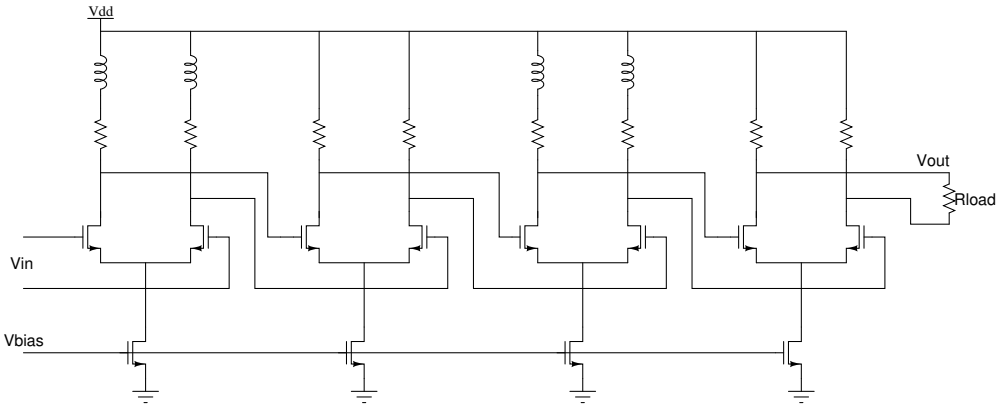


Figure 5.8: Circuit diagram of a current-model logic (CML) buffer.

constructed by ManiMOR is visualized in Fig. 5.9 (only first two dimensions are plotted) by picking three node voltages as axes. Clearly, the manifold is nonlinear, and blue circled points, which are sampled on a transient trajectory, do stay close to this nonlinear manifold. This confirms that the manifold attracts system responses.

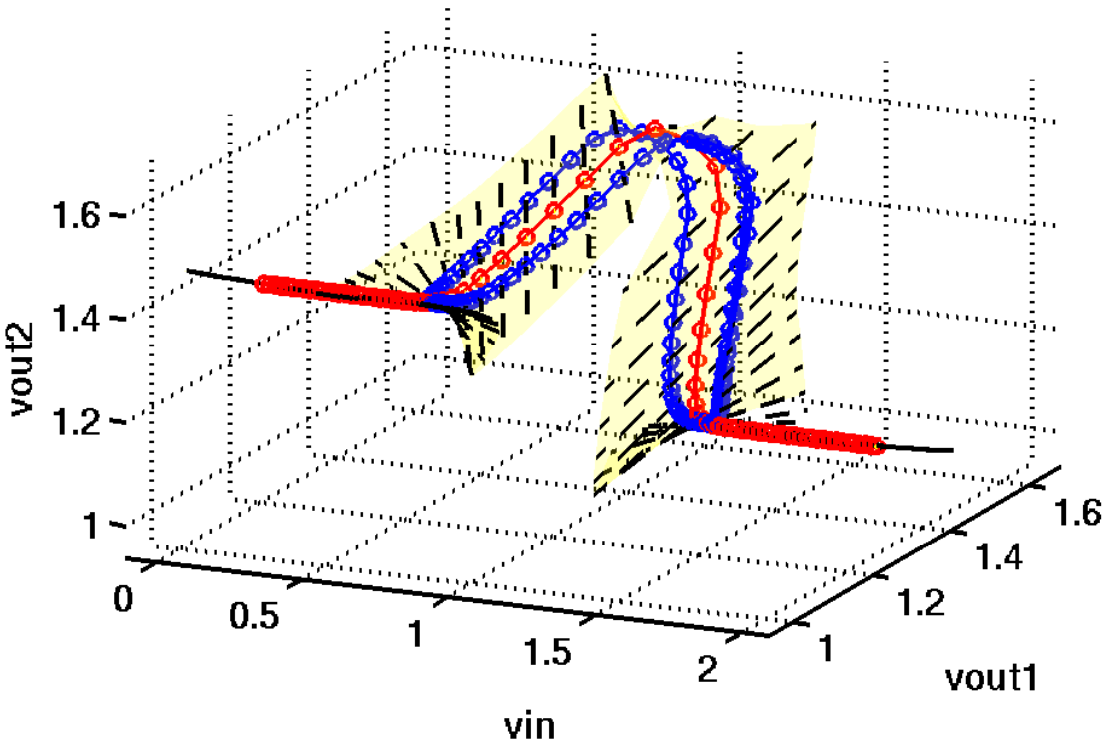


Figure 5.9: Visualization of the low-order manifold subspace generated by ManiMOR. The axes represent three node voltages in the circuit. The red curve consists of the DC operating points; the black dashed lines are the second local subspace bases; the yellow manifold is generated by stitching together all the local linear subspaces. The blue circled points are on a transient trajectory simulated using the full model.

In contrast, the TPWL model aggregates all the samples and all projection matrices into the final projection matrix. We used 20 DC operating points for training – by aggregating only the state variables, we obtain a matrix of 20 columns. The singular values of this matrix are shown in Fig. 5.10. Therefore, to get a set of bases which approximate all the DC operating points with an overall accuracy of 0.01, first 8 bases must be used. This implies that the size of the TPWL model must be at least 8, only to capture the DC response.

When generating the TPWL model, we use the same method (*i.e.*, Arnoldi algorithm),

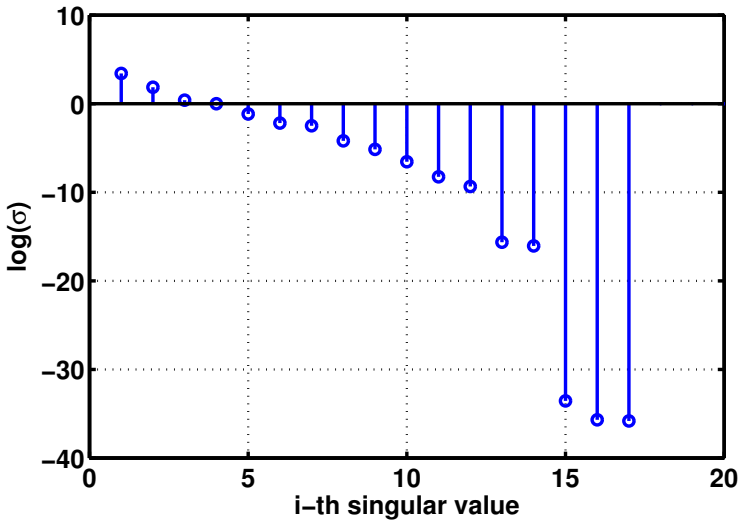


Figure 5.10: Singular values of the matrix containing 20 equilibrium points, sorted.

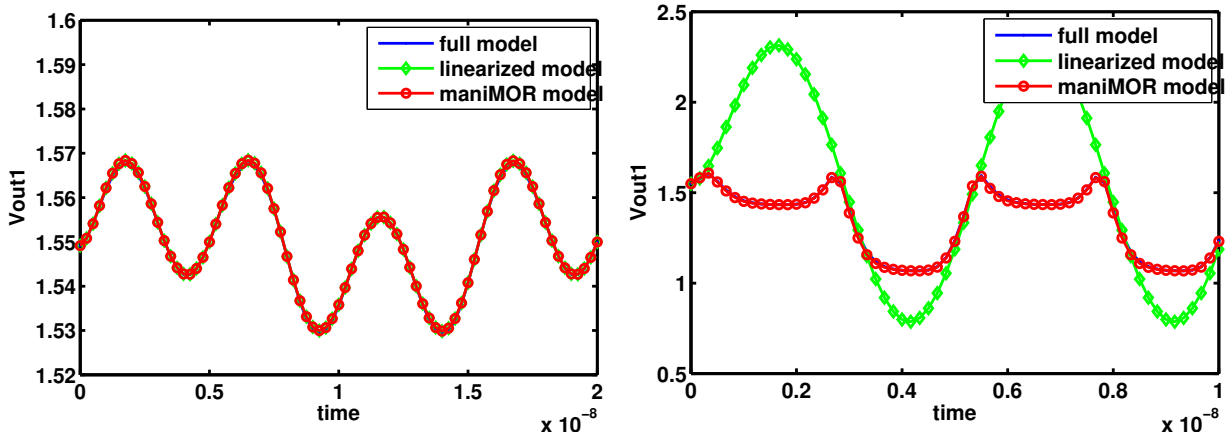
the same linear subspace size (*i.e.*, 5), to calculate the local linear projection matrices as in ManiMOR. As a result, a size 12 TPWL model is generated in order to match all the DC operating points, and the first 5 moments of the transfer functions for all the linearized systems.

In this example, we also generate a small-signal linearized model of size 5 – we linearize the nonlinear model around its DC solution, and generate a linear reduced model from that linear system.

We first compare the ManiMOR model to the small-signal linearized model, by applying a small signal input superimposed on a DC bias, $u(t) = 0.005\sin(\pi\times10^9 t)+0.0025\sin(1/3\pi\times 10^9 t)$. Not surprisingly, since the amplitude of the input is small, the transient trajectory is close to the DC operating point, and the linear small-signal reduced order model almost tracks the full model, as shown in Fig. 5.11(a). For the same reason, the result of the ManiMOR model also matches that of the full model.

However, when the input amplitude is large, the trajectory will not be confined close to a DC operating point, but travel among different regions. In that case, the small signal model fails to match the waveforms of the full model. To see that, we apply a sinusoidal input with larger amplitude $u(t) = 0.25\sin(\pi \times 10^9 t)$. As shown in Fig. 5.11(b), the output waveform is highly distorted. However, the waveform of the ManiMOR model is almost overlapped with that of the full model – this is because the ManiMOR model uses different local models when the state variable travels to different regions in the state space. On the contrary, the small signal model, which makes the "linear" and "small-signal" assumption, still generates a sinusoidal output waveform, and therefore fails to capture the distortion effect.

We then compare the ManiMOR model to the TPWL model by applying a step input



(a) Small signal input.

(b) Large signal input.

Figure 5.11: Comparison of ManiMOR model with linearized reduced-order model. The waveforms of one output using different models are plotted.

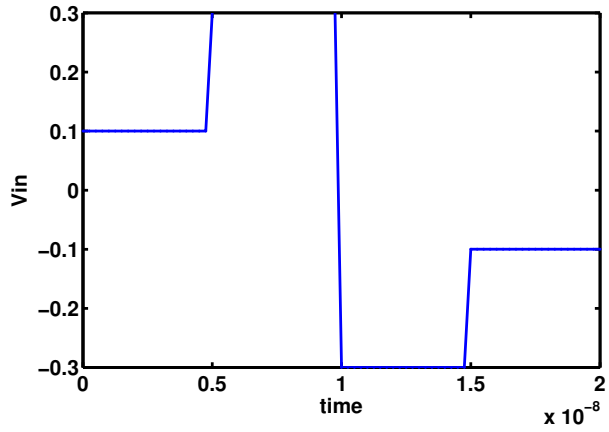
as shown in Fig. 5.12(a), which is chosen to traverse four DC operating points. As shown in Fig. 5.12(b), without the heuristic to handle the DC solutions, the TPWL model with model size 10, fails to converge back to the correct DC operating points. After including the DC operating points into the projection matrix, the TPWL model of size 10 still leads to some error. During experiments, we find that the model size must be at least 12 in order to match the original trajectory. This result is obtained by using a training input to be the same as the test input – potentially an even larger model will be needed.

In comparison, ManiMOR model with size 5 renders an output waveform almost indistinguishable from that of the full model – it is less than half of the size of the TPWL model with the same accuracy. We conclude that the redundancy in TPWL is because of the "aggregation" and the SVD step which embed all the DC solutions and all the local Krylov subspaces into a linear subspace. Under the linear subspace constraint, it is highly possible for TPWL to obtain less reduction than ManiMOR.

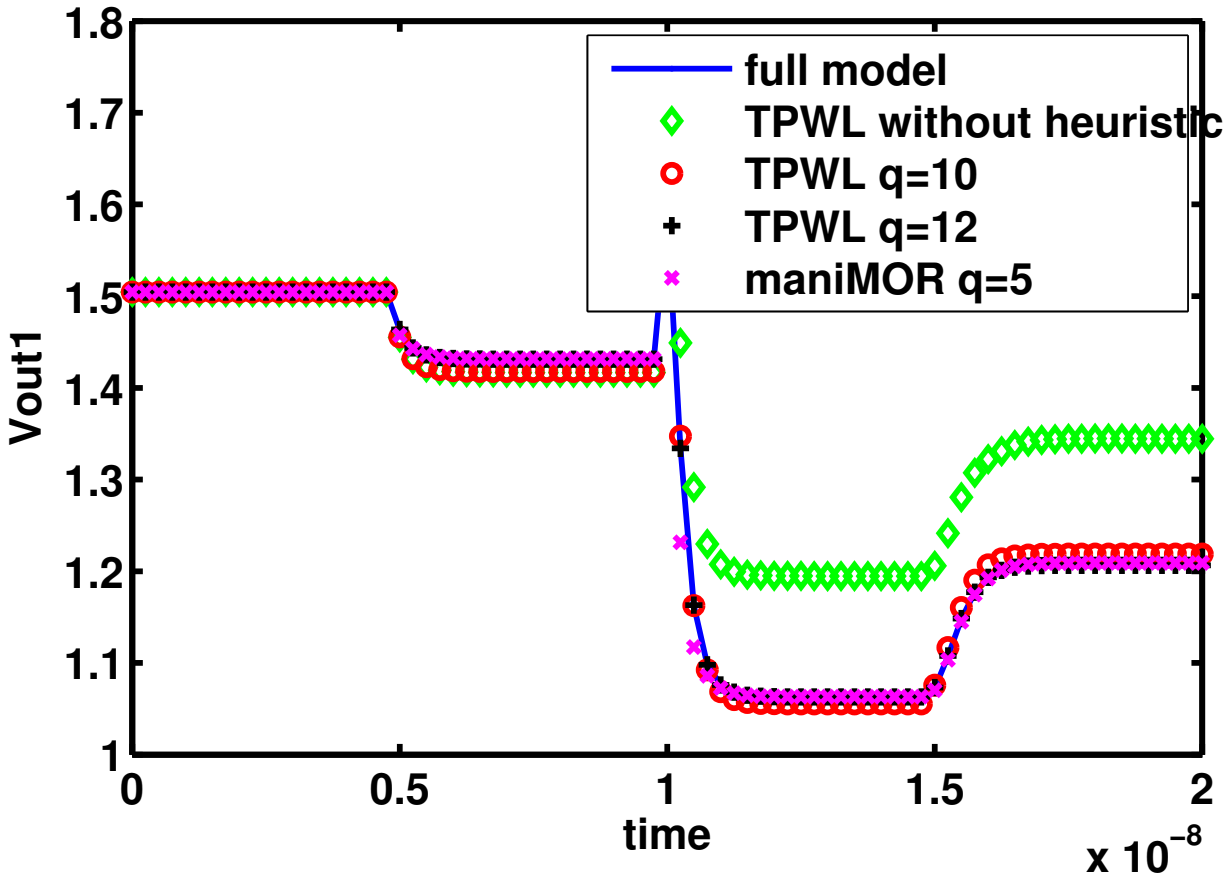
5.5.5 Yeast Pheromone Signaling Pathway

In this example, we show the application of ManiMOR macromodeling method on a Biochemical system. We have found that ManiMOR model reduces the system size by a large amount for this example.

In a chemical reaction, the number of reactants is usually large. For example, yeast pheromone signaling pathway [3] contains a chain of chemical reactions, as shown in Fig. 5.13. For each reaction, several reactants participate. Shown in [119, 120], 304 reactants in this pathway are modeled, rendering a dynamical system of size 304. However, questions, such



(a) Input signal.



(b) Simulation of TPWL model and maniMORmodel.

Figure 5.12: Comparison of ManiMOR model with TPWL model. The waveforms of one output voltage using different models are plotted.

as how inputs affect specified outputs of the system, are of more interest to bio-chemical scientists, and this is where MOR can play an important role. Reduced-order models also makes it possible to simulate "large scale" bio-chemical systems.

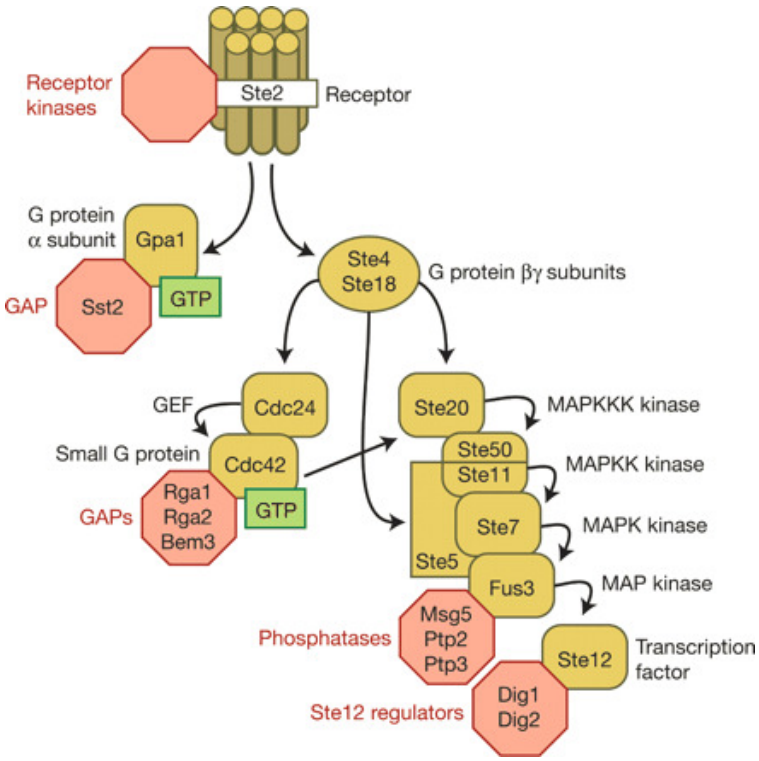
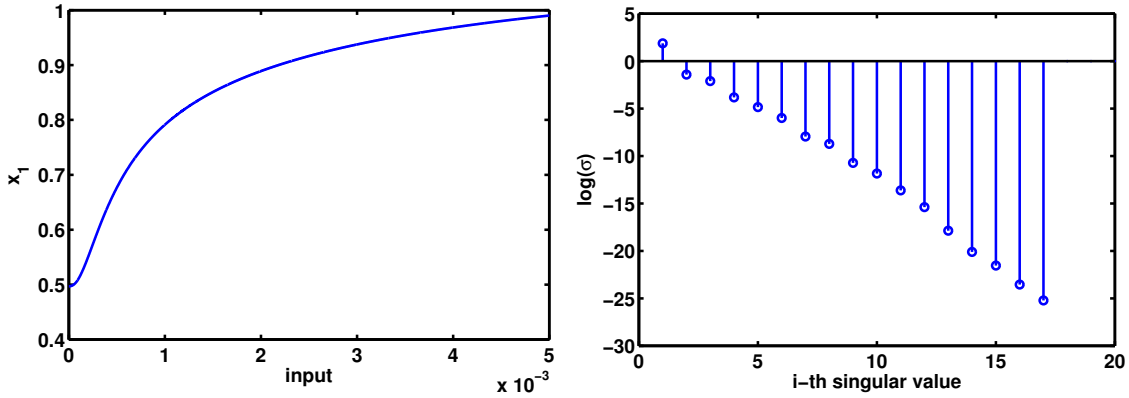


Figure 5.13: Components of the pheromone response pathway in yeast. [3] Each arrow represents a type of reaction.

In the procedure of manifold construction in ManiMOR, we carefully simulate the DC operating points due to a wide range of input values, since it is found that DC analysis for this specific system is hard to converge unless continuation/limiting methods are used. Moreover, it is observed that several equilibriums exist for some inputs, although some of them are unstable and cannot occur in reality. (The equilibrium solutions of one output are plotted in Fig. 5.14(a). In the process of building the ManiMOR model, we also implicitly eliminates all the useless equilibriums, which makes the DC convergence of the reduced model much better than the full model.

In contrast, TPWL model has to generate a matrix containing all the equilibrium points. We picked up 20 equilibrium points into a matrix. The singular values of this matrix is again plotted in Fig. 5.14(b). Setting the threshold of the singular value to be 10^{-10}, we have to include 8 basis vectors into the projection matrix, which renders at least a size 8 model.



(a) Concentrations of x_1. (b) Singular values.

Figure 5.14: Concentrations of x_1 at equilibrium corresponding to different inputs, and the singular values for the DC matrix.

The ManiMOR model is tested using a large signal input $u(t) = 0.003+0.002\sin(2\pi\frac{1}{72000}t)$. As plotted in Fig. 5.15, we observe a relatively good match between the ManiMOR model (size 30) and the full model. Even with a size 3 model, the error is quite small. In contrast, TPWL method is extremely inefficient in this case. A plain implementation of TPWL (including DC operating points, using local subspaces of size 6, setting the training input to be the test input) renders a model of size 300 (almost the same as the full model). Even though, it cannot match correct system dynamics. Our explanation is that it results from the problems brought by aggregation of projection matrices – trivial basis vectors kill the non-trivial ones. By employing a heuristic,[1] TPWL method works much better for this example, with order reduction from 304 to 144. But even with model size 144, the simulation results of TPWL model, as shown in Fig. 5.15, is not better than ManiMOR model with $q = 3$.

In this example, the ManiMOR model greatly out-performs the TPWL model. We also see that the drawbacks in the TPWL method can sometimes lead to significant error in practice, and these problems are very hard to be tackled if the common aggregated linear subspace is used in the reduction procedure.

[1]Since the first few basis vectors in the local projection matrix V_i are more important than others, we construct $\hat{V}_k$, whose columns are the aggregation of the k-th basis vector from all V_i at different samples. Then SVD is performed to each $\hat{V}_k$, and the basis vectors corresponding to singular values larger than a threshold ϵ are chosen. For different k, the threshold changes due to their importance. Then, we simply aggregate all $\hat{V}_k$ into $\hat{V}_{agg}$, and do an SVD on $\hat{V}_{agg} = U\Lambda V^T$ just for ortho-normalization. The projection matrix is then set to U. Using this heuristic, the important basis vectors are typically not filtered in the aggregation and SVD step.

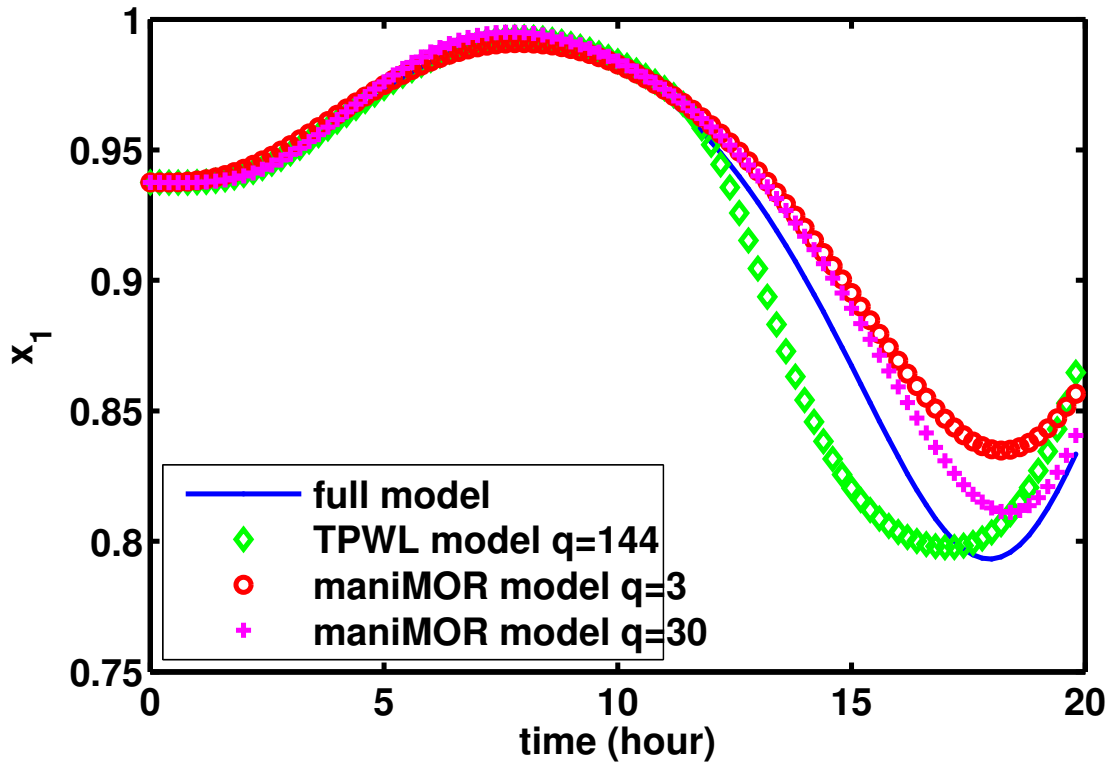


Figure 5.15: Transient simulation of the yeast pheromone pathway. The concentration of one reactant is plotted.

Chapter 6

QLMOR: MOR using Equivalent Quadratic-Linear Systems

In this chapter, we present another nonlinear projection-based MOR technique, based on an "equivalent" quadratic-linear system representation of the original nonlinear system. This representation is exact in the sense that the solution of the equivalent system is exactly the same as the original nonlinear system. Therefore, the quadratic-linear model differentiates itself from Taylor-expansion-based quadratic/polynomial models. The reduction is then performed on the quadratic-linear representation to obtain a quadratic-linear reduced model.

We first summarize previous work on polynomial reduced order modeling techniques, and the major problems therein. We then present algorithms that transform almost all nonlinear differential equations to their quadratic-linear representation. With the resulting quadratic-linear representation, we perform Volterra analysis, and derive a reduced-order modeling technique. We show experimental results and comparisons to other techniques.

6.1 Volterra-Series Based Reduction of Polynomial Systems

Previous Volterra-series based MOR methods [106, 107, 121] are derived from the result that the response of a nonlinear system can be decomposed into responses of a series of homogeneous nonlinear systems, *i.e.*, the system response $x(t)$ can be written as the summation of responses of all n-th order homogeneous nonlinear systems $x_n(t)$:

$$x(t) = \sum_{n=1}^{\infty} x_n(t), \tag{6.1}$$

where

$$x_n(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} h_n(\sigma_1, \cdots, \sigma_n) \\ u(t-\sigma_1) \cdots u(t-\sigma_n)\, d\sigma_1 \cdots d\sigma_n. \tag{6.2}$$

In (6.2), $h_n(\sigma_1, \cdots, \sigma_n)$ is the *Volterra kernel of order n*, and we can define $H_n(s_1, \cdots, s_n)$, the *transfer function of order n*, to be

$$H_n(s_1, \cdots, s_n) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} h_n(\sigma_1, \cdots, \sigma_n) \\ e^{-(s_1\sigma_1 + \cdots + s_n\sigma_n)}\, d\sigma_1 \cdots d\sigma_n, \tag{6.3}$$

which is the multi-variable Laplace transform of $h_n(\sigma_1, \cdots, \sigma_n)$ [105].

Previous Volterra-series based methods can be summarized in four steps:

1. Perform a Taylor expansion of the nonlinear function $f(x)$, *i.e.*, expand $f(x)$ in a series of multidimensional polynomials, and truncate the expansion to a predefined order (*e.g.*, 3, as shown in (6.4)[1]).

$$C\frac{d}{dt}x = G_1 x + G_2 x \otimes x + G_3 x \otimes x \otimes x + Bu, \tag{6.4}$$

   where C, G_1, G_2, G_3 and B are of appropriate dimensions given $x \in \mathbb{R}^N$ and $u \in \mathbb{R}$.

2. For methods based on bilinear forms, perform a Carleman bilinearization [73] on (6.4) to obtain a bilinear approximation

$$C\frac{d}{dt}x = Gx + Dxu + Bu, \tag{6.5}$$

   where C, G, D and B are of appropriate dimensions.

3. Construct a projection matrix $V \in \mathbb{R}^{N \times N_r}$ to match the moments of H_n, $n = 1, 2, 3, \cdots$ up to a given order. *E.g.*, to match up to the second order moments of $H_2(s_1, s_2)$ means to match the coefficients of terms $1, s_1, s_2, s_1^2, s_1 s_2, s_2^2$ in the Taylor expansion of $H_2(s_1, s_2)$ [106].

4. Construct the projected reduced system. *E.g.*,

$$C_r\frac{d}{dt}x_r = G_{r,1}x_r + G_{r,2}x_r \otimes x_r + G_{r,3}x_r \otimes x_r \otimes x_r + B_r u \tag{6.6}$$

[1]For notational simplicity, we will sometimes use $x^{②}$ to denote $\vec{x} \otimes \vec{x}$, $x^{③}$ to denote $\vec{x} \otimes \vec{x} \otimes \vec{x}$, *etc.*.

by defining

$$\begin{aligned} x &= Vx_r, \\ C_r &= V^T C V, \\ B_r &= V^T B, \\ G_{r,1} &= V^T G_1 V, \\ G_{r,2} &= V^T G_2 (V \otimes V), \\ G_{r,3} &= V^T G_3 (V \otimes V \otimes V). \end{aligned} \tag{6.7}$$

The common step in [106, 107, 121] is Taylor approximation. This results in three major drawbacks that limit the wide applicability of these methods. Firstly, the convergence of the Volterra system representation is only valid when $|u| \leq \epsilon$, for some small $\epsilon > 0$, *i.e.*, for "small" inputs, or the "weakly nonlinear" case. This constraint stems partly from the fact that the Taylor series only converges in a small region around the expansion point. Incorrect behavior of the reduced model due to large inputs was reported in [107], and similar results are reproduced in Section 6.5.2.

Secondly, Taylor expansion typically destroys important global properties of the system, such as passivity and stability. For example [107], the quadratic approximation for e^x, which is part of the I-V curve of a diode, around $x = 0$ leads to an artificial diode that can generate energy when the voltage applied is negative. As a result, the reduced model, which is created using the Taylor approximated system as a starting point, will typically not be passive/stable.

Thirdly, the computational cost for higher-order terms such as $G_{r,3}\, x_r \otimes x_r \otimes x_r$ dominates the computational cost of the reduced model. Since $x_r \in \mathbb{R}^{N_r}$, $G_{r,3} \in \mathbb{R}^{N_r \times N_r^3}$, and with $G_{r,3}$ usually dense, the computational cost is $O(N_r^4)$. (It is possible to make $G_{r,3}$ of a smaller size by combining terms in $x_r \otimes x_r \otimes x_r$, but the computational complexity does not change.) Although it is N-independent, it can potentially be too expensive to use in practice. Moreover, it is generally difficult to accurately extract the high-order matrices $G_2, G_3, \cdots$, which also constrains applications of Taylor-expansion based methods.

Remark MOR methods based on bilinear forms and Volterra series [107] actually define a nonlinear projection – *e.g.*, by explicitly writing out the differential equations for $\vec{x} \otimes \vec{x}, \vec{x} \otimes \vec{x} \otimes \vec{x}$, the final projection is defined by $x_r = V^T[x^T, x^T \otimes x^T, x^T \otimes x^T \otimes x^T]^T$, which is a polynomial projection function. In this sense, QLMOR indeed allows a richer set of projections than those in previous methods [107].

Briefly summarizing this section, we list in Table 6.1 the main drawbacks and difficulties that have not been solved in existing MOR methods, and how well QLMOR addresses these drawbacks.

Table 6.1: Drawbacks addressed by QLMOR

Existing methods	Drawbacks	QLMOR
Volterra series based	Valid only for small-signal response	Solved
	Stability, passivity loss	Partially solved
	Expensive computation with higher-order terms	Improved
Trajectory based	Expensive $V^T f(Vx_r)$ computation	Improved

6.2 QLMOR Overview

In this section, we outline the key ideas of QLMOR. Details are provided in Section 6.3 and Section 6.4.

From the discussion in Section 6.1, the drawbacks of previous Volterra-series based methods are mainly caused by Taylor expansion step, in which "information" about the system is lost.

In contrast, QLMOR derives a polynomial system that is equivalent to the orignal system without losing any information. By doing so, QLMOR avoids the "small" input assumption, and preserves properties of the original system in the first step.

We show that any polynomial ODEs can be converted into *quadratic-linear differential algebraic equations* (QLDAEs)

$$C\frac{d}{dt}x = G_1 x + G_2 x \otimes x + D_1 xu + D_2(x \otimes x)u + Bu, \tag{6.8}$$

where $C \in \mathbb{R}^{N\times N}$, $G_1 \in \mathbb{R}^{N\times N}$, $G_2 \in \mathbb{R}^{N\times N^2}$, $D_1 \in \mathbb{R}^{N\times N}$, $D_2 \in \mathbb{R}^{N\times N^2}$, $B \in \mathbb{R}^{N\times 1}$. That is, the differential equations are quadratic in state variables x and linear in the input u.

Similarly, any generalized polynomial system (to be defined in Section 6.3.8) can be converted into generalized QLDAEs

$$\frac{d}{dt}\left[C_1 x + C_2 x \otimes x\right] = G_1 x + G_2 x \otimes x + (D_1 x + D_2 x \otimes x)u + Bu, \tag{6.9}$$

where $C \in \mathbb{R}^{N\times N}$, $C_2 \in \mathbb{R}^{N\times N^2}$, $G_1 \in \mathbb{R}^{N\times N}$, $G_2 \in \mathbb{R}^{N\times N^2}$, $D_1 \in \mathbb{R}^{N\times N}$, $D_2 \in \mathbb{R}^{N\times N^2}$, $B \in \mathbb{R}^{N\times 1}$.

The QLDAE system is an equivalent representation of the original system, meaning that the solution of the QLDAE system is exactly equal to that of the original one if the initial conditions are consistent (to be defined in Section 6.3). Therefore, no approximation is involved in this step.

QLMOR reduces this QLDAE system to a smaller model that is also in the QLDAE form, as opposed to the polynomial form in prior methods. The important advantage here is that higher-order terms beyond quadratic terms are absent in QLMOR-reduced model. Therefore, it avoids the expensive computational cost (*e.g.*, $x_r \otimes x_r \otimes x_r$) in a high-order

polynomial reduced model, and quadratic terms dominates the computational complexity.

The reduced order model is obtained by projection of the QLDAE system. While it looks like a linear projection (of the QLDAE system), it indeed defines a nonlinear projection (of the original system). To obtain the *minimum* subspace in the sense of moment matching, we analyze the transfer functions of systems in the QLDAE form, and results related to prior methods are derived. For example, it turns out that the moments of $H_2(s_1, s_2)$ of a QLDAE system are a summation of the moments of $H_2(s_1, s_2)$ of a quadratic system and a bilinear system – direct application of existing codes are possible. In our algorithm, we also avoid explicit moment evaluations, and therefore the algorithm has better numerical stability properties.

Another point to note is that QLMOR serves as a new core method that can be combined with any trajectory-based methods. For example, in TPWL method, to generate reduced models along a trajectory, previous Krylov-subspace methods or TBR methods can be replaced with QLMOR straightforwardly, potentially leading to a more accurate model and less number of regions along the trajectory. For another example, POD method can be directly applied to the transformed QLDAE system, which results in a reduced model whose computational cost is lower than the usual POD reduced model if the reduced model size remains the same.

To summarize, QLMOR is composed of three steps:

1. construct an equivalent polynomial system from the orignal system;

2. construct an equivalent quadratic-linear system from the polynomial system;

3. reduced the quadratic-linear system to a smaller quadratic-linear system.

We chart in Fig. 6.1 the basic flow of QLMOR, as well as the flow of MOR based on polynomial systems and bilinear systems for comparison. In what follows, Section 6.3 presents the polynomialization and the quadratic-linearization procedure, and Section 6.4 discusses the reduction step.

6.3 Quadratic-Linearization Procedure

In this section, we develop an algorithm to convert systems of ordinary differential equations to an equivalent system which has functions that are quadratic in state variables and linear in inputs. We call it **quadratic-linearization** procedure, and we call the resulting system QLDAEs (quadratic-linear differential-algebraic equations). We also discuss extensions to quadratic-linearize DAEs in the end of this section.

Considering nonlinear ODEs, the quadratic-linearization procedure is composed of two steps:

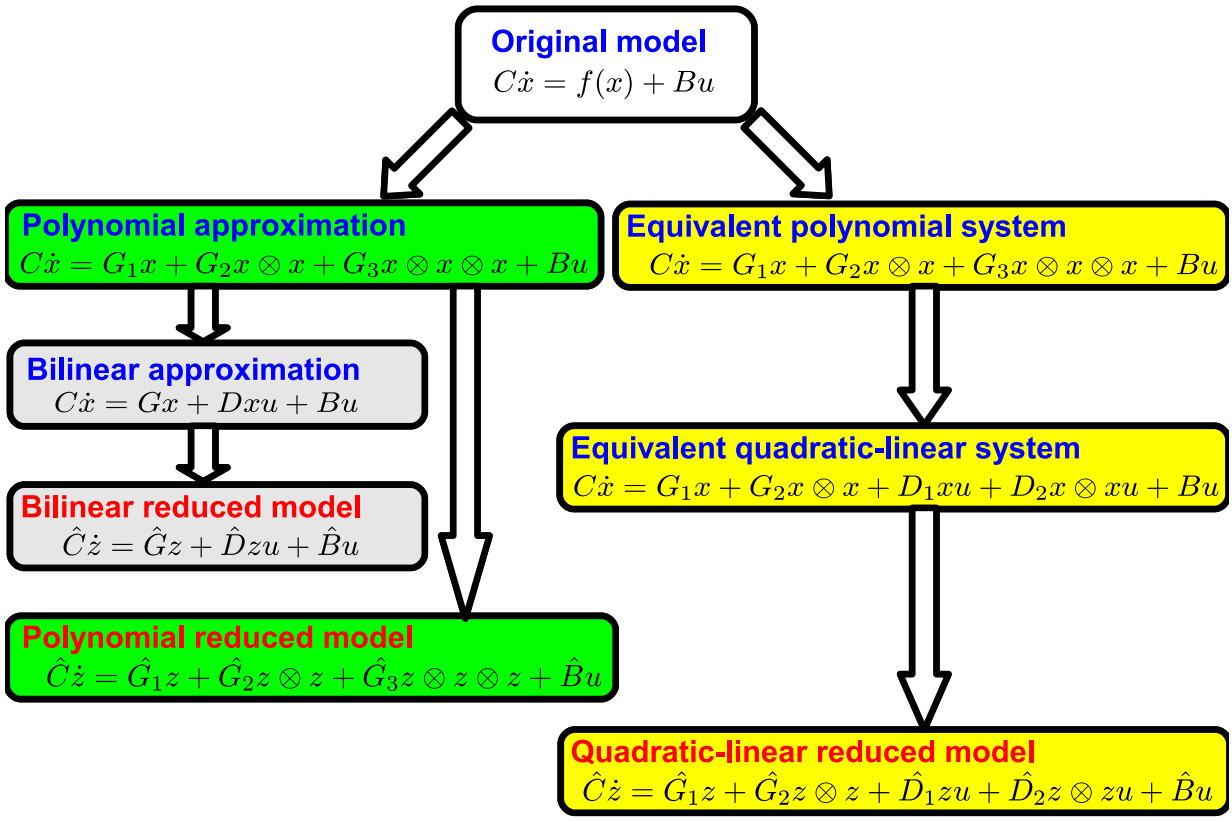


Figure 6.1: QLMOR flow (yellow), with comparison to MOR based on polynomial form (green) and bilinear form (gray).

1. convert the original nonlinear ODEs

$$\frac{d}{dt}x = f(x) + Bu \tag{6.10}$$

   into a polynomial system;

2. convert the polynomial system into QLDAEs.

We call the first procedure polynomialization, the second procedure quadratic-linearization of a polynomial system.

6.3.1 Nonlinear ODEs and Polynomial Systems

To be more specific, we consider "semi-general" nonlinear ODEs (6.10) where the nonlinear functions $f(x) = [f_1(x), \cdots, f_N(x)]$ can be written as a linear combination of a set of elementary functions $\{g_i(x)\}, i = 1, \cdots, k$, *i.e.*, the differential equation for x_i is

$$\frac{d}{dt}x_i = p_i^T x + a_{i,1}g_1(x) + a_{i,2}g_2(x) + \cdots + a_{i,k}g_k(x) + b_i u, \forall i = 1, \cdots, N, \tag{6.11}$$

where $p_i \in \mathbb{R}^{N\times 1}$, $a_{i,j} \in \mathbb{R}$, $b_i \in \mathbb{R}$. In matrix form, (6.11) can be written as

$$\frac{d}{dt}x = Px + Ag(x) + Bu, \tag{6.12}$$

where $P^T = [p_1, \cdots, p_N]$, $A = [a_{i,j}]$, $g(x) = [g_1(x); \cdots, g_k(x)]$ and $B = [b_1; \cdots; b_N]$. Note that A is extremely sparse due to the fact that each differential equation typically contains just a few nonlinear functions.

As mentioned in Section 2.7, this form of differential equations is prevalent in circuit simulation, as well as other engineering problems. For example, in circuit MNA equations [108, 109], each $g_i(x)$ represents the current flowing into a node from a specific device; in chemical rate equations, each $g_i(x)$ represents the reaction rate according to a specific reaction; in mechanical applications, $g_i(x)$ is the force to the system.

The elementary functions $g_i(x)$ include, but not limited to, the ones listed in Table 6.2, and compositions of these elementary functions. Because of the function composition, the nonlinear functions in Table 6.2 in fact cover a very rich set of nonlinear functions that are almost adequate in engineering problems.

Specifically, in circuit simulation, two important classes of models use functions that are a subset of composition of these elementary functions.

The first class is the most commonly used transistor models, such as BSIM [118] or PSP models [123]. These models are written in C/C++, and contain "if-else" statements to implement piecewise-linear functions. For example, the C code

Table 6.2: Commonly used nonlinear functions

Function	Applications
$x_1^{\beta_1} \cdots x_N^{\beta_N}$	chemical rate equations MOSFET in saturation mode (x^2, x^α [122])
$\frac{x}{k+x}$	chemical rate equations, smoothing functions
e^x	diodes, BJTs, ion-channel models, smoothing functions
$\sin(x), \cos(x)$	control systems (*e.g.*, x is the angle to be steered)

```
if (y<1)
  return f1(x,y);
else
  return f2(x,y);
```

implements the function

$$f(x,y) = f_1(x,y)(1 - s(y-1)) + f_2(x,y)s(y-1), \tag{6.13}$$

where $s(\cdot)$ is the step function, Because that the step function can be well-approximated and smoothed by

$$s(t) \simeq 0.5 \tanh(Kt) + 0.5, \tag{6.14}$$

where K is a large constant number, and that $\tanh(t)$ is a composition of elementary functions, we can represent all piecewise-linear functions as compositions of elementary functions in Table 6.2.

The second class is models composed of look-up tables (LUT). The LUT model can be viewed to implement a function whose outputs of given input values are stored in the LUT. Therefore, one way to interpret the function that LUT implements is through the kernel method (interpolation), *i.e.*,

$$f(x) = \sum_i f(x_i) K(x - x_i), \tag{6.15}$$

where $K(\cdot)$ is a kernel function [124]. By choosing $K(\cdot)$ appropriately, we can make $f(x)$ a piecewise-linear function, a polynomial function, or more generally the composition of elementary functions.

Given these nonlinear ODEs, in the first step, we convert them into a polynomial system which is defined as follows:

Definition 6.3.1 (Polynomial system of order M) *A polynomial system (6.16) is said to be of order M, if the highest order of monomial of x is M, i.e.,*

$M = \max_{k,i} \left(\sum_{l=1}^{N} \beta_{i,k,l}, \sum_{l=1}^{N} \gamma_{i,k,l} \right)$.

$$\sum_k c_{i,k} \dot{x_i} = \sum_k \alpha_{i,k} x_1^{\beta_{i,k,1}} \cdots x_N^{\beta_{i,k,N}} + \left(\sum_k \eta_{i,k} x_1^{\gamma_{i,k,1}} \cdots x_N^{\gamma_{i,k,N}} \right) u, \quad i = 1, \cdots, N. \tag{6.16}$$

In (6.16),

$$c_{i,k} \in \mathbb{R}, \alpha_{i,k} \in \mathbb{R}, \eta \in \mathbb{R}^N, \beta_{i,k,j} \geq 0, \gamma_{i,k,j} \geq 0, \beta_{i,k,j} \in \mathbb{Z}, \gamma_{i,k,j} \in \mathbb{Z}, \forall k, j. \tag{6.17}$$

Note that the nonlinear function on the RHS of (6.16) is linear in terms of the input u which is an important property we will preserve in the process of polynomialization and quadratic-linearization.

6.3.2 Polynomialization by Adding Polynomial Algebraic Equations

The first approach to polynomialization is only applicable to certain nonlinear functions, such as $g_i(x) = \frac{1}{k+x}$. The procedure is quite straightforward and is listed as follows:

1. Introduce a new variable $y_i = g_i(x)$;
2. Replace $g_i(x)$ by y_i in original equations;
3. Add $y_i - g_i(x) = 0$ as a new polynomial equation.

Example 6.3.1 *Consider an ODE*

$$\dot{x} = x + x^2 + \frac{1}{k+x} + u. \tag{6.18}$$

To polynomialize (6.18), we introduce a new variable $y = \frac{1}{k+x}$, and rewrite $y = \frac{1}{k+x}$ as a polynomial equation, and obtain

$$\begin{aligned} \dot{x} &= x + x^2 + y + u \\ 0 &= -ky - xy + 1. \end{aligned} \tag{6.19}$$

This approach preserves the linearity to the input u which is shown by the following lemma:

Lemma 6.3.2 *By adding polynomial algebraic equations through variable change, the resulting polynomial system is linear in terms of the input u.*

Proof The new variable introduced is a function of x, *i.e.*, $y_i = g_i(x)$, and therefore, the new polynomial equation introduced is independent of the input u.

In original differential equations, we only replace $g_i(x)$ with y_i, and therefore, the ODEs are still linear in terms of the input u. ■

The assumption for this approach to work is that the variable change can lead to a polynomial algebraic equation. For example, any variable change to a rational function leads to a polynomial algebraic equation. Constrained by this assumption, however, this approach cannot deal with other nonlinear functions such as $y = e^x$.

6.3.3 Polynomialization by Taking Lie Derivatives

To polynomialize a larger set of nonlinear functions, we may add differential equations, instead of algebraic equations. The procedure is similar to the previous one except that we add a new differential equation of the new variable:

1. Introduce a new variable $y_i = g_i(x)$;
2. Replace $g_i(x)$ by y_i in original equations;
3. Add $\dot{y_i} = \frac{dg_i}{dx} f(x)$ as a new polynomial differential equation, where $\frac{dg_i}{dx}$ consists of polynomial functions of x and y_i.[2]

The resulting differential equations are then

$$\begin{aligned} \frac{d}{dt}x_i =& p_i^T x + a_{i,1}y_1 + \cdots + a_{i,k}y_k + Bu, \quad i = 1, \cdots, N \\ \frac{d}{dt}y_i =& \mathfrak{L}_{\dot{x}} g_i(\vec{x}) = g_i'(\vec{x})(p_i^T \vec{x} + a_{i,1}y_1 + \cdots + a_{i,k}y_k + Bu), \quad i = 1, \cdots, k \end{aligned} \tag{6.20}$$

where $g_i'(x) = \frac{dg_i(x)}{dx}$. Hence, if $g_i'(x)$ consists of polynomial functions in x and y_i, (6.20) is a polynomial system.

We now examine some functions $y_i = g_i(x)$ and show that in all cases we encounter, $g_i'(x)$ consists of polynomial functions in x and y_i.

Considering uni-variable function (*i.e.*, functions of only one variable), we have

$$\begin{aligned} y = e^x &\Rightarrow (e^x)' = e^x = y \\ y = 1/(k+x) &\Rightarrow (1/(k+x))' = -1/(x+k)^2 = -y^2 \\ y = x^\alpha &\Rightarrow (x^\alpha)' = \alpha x^{\alpha-1} = \alpha y x^{-1} \\ y = \ln(x) &\Rightarrow (\ln(x))' = x^{-1}. \end{aligned} \tag{6.21}$$

[2]The new equation is obtained by applying chain rule, and is the Lie derivative of y_i with respect to $\frac{d}{dt}x$.

Note that when $y = x^\alpha$, $y' = \alpha y x^{-1}$, where another new variable $z = x^{-1}$ can be further introduced to eliminate the negative powers of x. The same reasoning applies to $y = \ln(x)$. Therefore our claim that $g_i'(\vec{x})$ are polynomial functions in x and y_i is correct for these functions.

Some other uni-variable functions need to be handled by introducing two new variables. For example, if $g_i(x) = \sin(x)$, then we have

$$\begin{aligned} & y_1 = \sin(x), \quad y_2 = \cos(x) \\ \Rightarrow & (\sin(x))' = \cos(x) = y_2 \\ & (\cos(x))' = -\sin(x) = -y_1. \end{aligned} \tag{6.22}$$

Considering functions of two variables, we have

$$\begin{aligned} y = x_1 + x_2 \Rightarrow y'(x) = [1, 1] \\ y = x_1 x_2 \Rightarrow y'(x) = [x_2, x_1], \end{aligned} \tag{6.23}$$

which also shows that $g_i'(x)$ are polynomial functions in x and y_i is correct for these functions.

Furthermore, if $g(x)$ is a composition of several functions, *e.g.*, $g(x) = (g_2 \circ g_1)(x) = g_1(g_2(x))$, we can convert it into polynomial nonlinearities by a similar procedure:

1. Introduce new variables $y_2 = g_2(x)$ and $y_1 = g_1(y_2)$;
2. Replace $g_1(g_2(x))$ by y_1 in original equations;
3. Add $\dot{y_1} = \frac{\partial y_1}{\partial y_2}\dot{y_2}$ and $\dot{y_2} = \frac{\partial y_2}{\partial x}\dot{x}$ as new polynomial differential equations.

Since we can show that $\frac{\partial y_1}{\partial y_2}$ and $\frac{\partial y_2}{\partial \vec{x}}$ are polynomial functions of x, y_1, y_2, we conclude that we can polynomialize nonlinear ODEs which have nonlinear functions that are compositions of nonlinear kernels in Table 6.2.

Similar to the previous approach, we have the following lemma regarding the linearity to the input u:

Lemma 6.3.3 *By adding polynomial differential equations through variable change, the resulting polynomial system is linear in terms of the input u.*

Proof The new variable introduced is a function of x, *i.e.*, $y_i = g_i(x)$, and the new differential equation is $\frac{d}{dt}y_i = g_i'(x)\frac{d}{dt}x$.

Given that $\frac{d}{dt}\vec{x}$ is a function that is linear in terms of the input u, and $g_i'(x)$ is a function of x and y_i, we have $\frac{d}{dt}y_i$ is also linear in terms of the input u. ■

Combining the above derivations, Theorem 6.3.4 follows:

Theorem 6.3.4 *By iteratively applying polynomialization by adding polynomial algebraic equations and taking Lie derivatives, a nonlinear system with nonlinear functions being compositions of functions in Table 6.2 can be polynomialized into a polynomial system in the form of (6.16).*

Moreover, the size of the equivalent polynomial system is linear with respect to the number of elementary functions in original ODEs/DAEs.

Proof The proof for the first part is constructive – the procedure described in previous subsections (also summarized in Algorithm 6 in Section 6.3.4) converts ODEs/DAEs to an equivalent polynomial system.

Regarding the size of the equivalent polynomial system, we note that for an elementary function, we can always polynomialize it by adding constant number ($O(1)$) of new variables and equations (normally 1 or 2); For a nonlinear function that is a composition of m elementary functions, *i.e.*, $g(x) = g_1(g_2(\cdots g_m(x)\cdots))$, we can always polynomialize it by adding $O(m)$ new variables and equations. Therefore, the size of the equivalent polynomial system is linear with respect to the number of elementary functions in original ODEs/DAEs. ■

Remark In circuit applications, the number of elementary functions is linear with respect to the number of nonlinear devices in the circuit. Therefore, the polynomialization procedure scales well with the size of the circuit. Similar results can be obtained for other engineering problems.

6.3.4 Polynomialization Algorithm

Let the original nonlinear system be given in the form of (6.11), we summarize the polynomialization algorithm in Algorithm 6[3]. Because the size of the equivalent polynomial system is linear with respect to the number of elementary functions in original ODEs/DAEs, the computational complexity of Algorithm 6 is linear with respect to the number of elementary functions in original ODEs/DAEs.

Example 6.3.5 *Consider an ODE*

$$\dot{x} = \frac{1}{1+e^x}. \tag{6.24}$$

To polynomialize (6.24), we introduce two new variables

$$\begin{aligned} y_2 =& e^x \\ y_1 =& \frac{1}{1+y_2}. \end{aligned} \tag{6.25}$$

[3]In this algorithm, we perform polynomialization by only taking Lie derivatives, because it is more general.

Algorithm 6 Polynomialization of Nonlinear ODEs

Inputs: $E = [\dot{x_1}, \cdots, \dot{x_n}]$, the list of symbolic expressions of the ODEs.
Outputs: Y_{var}, the set of new variables;
Y_{expr}, the set of expressions of new variables;
E, the list of symbolic expressions of the polynomial ODEs.

1: Initialize $Y_{var} \leftarrow \{\}$, $Y_{expr} \leftarrow \{\}$;
2: **repeat**
3: From E, the list of symbolic expressions of the nonlinear ODEs, pick a nonlinear function $g(x, v)$ that is not a polynomial function of x and $v \in Y_{var}$.
4: Define a new state variable $y = g(x, v)$.
5: Add y into Y_{var}, and $g(x, v)$ into Y_{expr}.
6: Compute the symbolic expression of $\dot{y} = g'(x, v)[\dot{x}; \dot{v}]$.
7: Add the symbolic expression of $\dot{y}$ to E.
8: In E, replace occurrences of expressions in Y_{expr} by corresponding variables in Y_{var}.
9: **until** All expressions in E are polynomial functions of x and variables in Y_{var}.

By adding differential equations for y_1 and y_2, we obtain a polynomial system

$$\begin{aligned} \dot{x} &= y_1 \\ \dot{y_1} &= -y_1^2 \dot{y_2} = -y_2 y_1^3 \\ \dot{y_2} &= y_2 \dot{x} = y_2 y_1. \end{aligned} \tag{6.26}$$

Example 6.3.6 (A MOS Circuit) *Consider a MOSFET circuit shown in Fig. 6.2, where x_1 and x_2 are two node voltages, u_1 and u_2 are inputs, the resistance of resistors is 1 and the capacitance of capacitors is 1. We can write the differential equations in terms of node voltage x_1 and x_2 as follows:*

$$\begin{aligned} \dot{x_1} &= -x_1 + u_1 \\ \dot{x_2} &= -x_2 + u_2 - I_{DS}(x_1, x_2), \end{aligned} \tag{6.27}$$

where I_{DS} is defined in the CMOS device model.

For illustration, we use a smoothed version of the Schichman-Hodges model [125]. The Schichman-Hodges equations (also known as the SPICE level 1 model) for the MOSFET are

$$\begin{aligned} I_{DS} &= \begin{cases} \beta \left[(V_{GS} - V_T) - \frac{V_{DS}}{2} \right] V_{DS}, & \text{if } V_{DS} < V_{GS} - V_T \\ \frac{\beta}{2} (V_{GS} - V_T)^2 & \text{if } V_{DS} \geq V_{GS} - V_T \end{cases} \\ I_{GS} &= 0, \end{aligned} \tag{6.28}$$

where β and V_T are constants. The smoothed version of the Schichman-Hodges equations

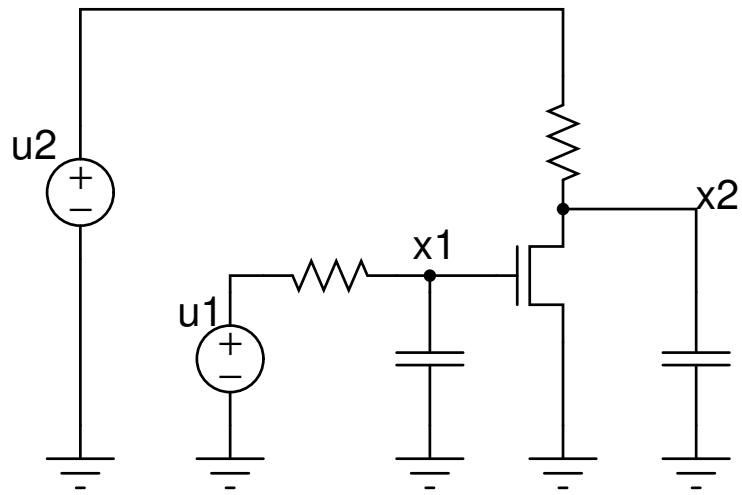


Figure 6.2: A simple CMOS circuit.

(for I_{DS}) is

$$\begin{aligned} I_{DS}(V_{GS}, V_{DS}) =& \beta \left[(V_{GS} - V_T) - \frac{V_{DS}}{2} \right] V_{DS} s(V_{GS} - V_T - V_{GS}) \\ &+ \frac{\beta}{2}(V_{GS} - V_T)^2 (1 - s(V_{GS} - V_T - V_{GS})), \end{aligned} \tag{6.29}$$

where $s(x)$ is a smoothing function defined using a hyperbolic tangent function

$$s(x) = \frac{1}{2}(\tanh(Kx) + 1) = \frac{e^{2Kx}}{e^{2Kx} + 1}, \tag{6.30}$$

where K is a constant.

Therefore, the circuit equations (6.27) are

$$\begin{aligned} \dot{x_1} =& - x_1 + u_1 \\ \dot{x_2} =& - x_2 + u_2 - \left[\beta \left[(x_1 - V_T) - \frac{x_2}{2} \right] x_2 \frac{e^{2K(x_1 - V_T - x_2)}}{e^{2K(x_1 - V_T - x_2)} + 1} \right. \\ &\left. + \frac{\beta}{2}(x_1 - V_T)^2 \frac{1}{e^{2K(x_1 - V_T - x_2)} + 1} \right]. \end{aligned} \tag{6.31}$$

To polynomialize (6.31), we introduce two new variables

$$\begin{aligned} y_1 &= e^{2K(x_1 - V_T - x_2)} \\ y_2 &= \frac{1}{y_1 + 1}, \end{aligned} \tag{6.32}$$

and add differential equations for y_1 and y_2. We then obtain a polynomial system

$$\begin{aligned} \dot{x_1} =& - x_1 + u_1 \\ \dot{x_2} =& - x_2 + u_2 - \left[\beta \left[(x_1 - V_T) - \frac{x_2}{2}\right] x_2 y_1 y_2 + \frac{\beta}{2}(x_1 - V_T)^2 y_2\right]. \\ \dot{y_1} =& 2K e^{2K(x_1 - V_T - x_2)}(\dot{x_1} - \dot{x_2}) = 2K y_1 (\dot{x_1} - \dot{x_2}) \\ \dot{y_2} =& - \frac{1}{(1 + y_1)^2}\dot{y_1} = -y_2^2 \dot{y_1}. \end{aligned} \tag{6.33}$$

Remark The polynomialization of a nonlinear system is **not unique**. In fact, there should exist a **minimum-size** polynomial system that corresponds to the original system. The algorithm to find this *minimum-size* polynomial system could be devised by symbolic computational tools. Methods in logic synthesis might also be applicable.

Despite the non-uniqueness, however, as long as the existence and uniqueness conditions of solutions to differential equations are satisfied, the polynomial system is **equivalent** to the original system since they have the same solution.

Formally, suppose the original DAEs have state variables x. We define new variables $y = g(x)$, and expand the DAEs by adding differential equations for y. Let $x(0)$ be the initial condition of the original DAEs, and $(x(0), y(0))$ be the initial condition of the expanded DAEs. If $y(0) = g(x(0))$ and $x(t)$ is the solution to the original DAEs, then $(x(t), g(x(t)))$ is the solution to the expanded DAEs. Moreover, if the expanded DAEs satisfy the existence and uniqueness condition of solutions, $(x(t), g(x(t)))$ is also the unique solution.

After the polynomialization, we obtain (6.16), a polynomial system of order M. Now we present two approaches to convert a polynomial system (6.16) to a QLDAE system (6.8).

6.3.5 Quadratic-Linearization by Adding Quadratic Algebraic Equations

Similar to the polynomialization procedure, we iteratively define new variables y, such that

$$y = x_1^{\beta_1} \cdots x_N^{\beta_N} / \hat{x} \tag{6.34}$$

is satisfied. In (6.34), $\hat{x}$ is a variable chosen from the state variables of the expanded system at each iteration – they include all the original state variables, as well as the newly added

state variables up to that iteration. We then replace $x_1^{\beta_1} \cdots x_N^{\beta_N}$ in original equations by $y\hat{x}$, and we add (6.34) into the original system. The equations in the form of (6.34) are then re-written as quadratic algebraic equations, if possible, and are added to the original system.

Example 6.3.7 *Consider an ODE*

$$\dot{x} = x^3 + u(t). \tag{6.35}$$

To quadratic-linearize (6.35), we introduce a new variable $y = x^2$ (i.e., $\hat{x} = x$ in (6.34)), and add it as a quadratic equation. The resulting quadratic-linear equations are

$$\begin{aligned} \dot{x} &= xy + u \\ 0 &= y - x^2. \end{aligned} \tag{6.36}$$

Similar to the polynomialization procedure, quadratic-linearization procedure also preserves the linearity to the inputs u. Besides, we prove in the following theorem that a polynomial system can always be quadratic-linearized by adding quadratic algebraic equations. As a byproduct, the theorem also provides an upper bound on the size of the QLDAE system derived from a polynomial system.

Theorem 6.3.8 *For a polynomial system of order M, the maximum number of state variables in the resulting QLDAE system does not exceed the number of state variables of $[x^T, (x^T)^{\textcircled{2}}, (x^T)^{\textcircled{3}}, \cdots, (x^T)^{(\lceil \frac{M}{2} \rceil)}]^T$, by adding quadratic algebraic equations,*

Proof Let $y = [x^T, (x^T)^{\textcircled{2}}, \cdots, (x^T)^{\textcircled{p}}]^T, p \in \mathbb{Z}^+$, then QLDAEs in y can generate any nonlinear functions of the form $G_1 y + G_2 y \otimes y$, which is essentially all the polynomial functions in x of order not greater than $2p$.

So, $M \leq 2p$, *i.e.*, $p = \lceil \frac{M}{2} \rceil$. ■

6.3.6 Quadratic-Linearization by Taking Lie Derivatives

Similarly, taking Lie derivatives of polynomial functions preserves the linearity to the input u, and it can also lead to a QLDAE system, as proved in Theorem 6.3.9. However, the theoretical upper bound on the size of the resulting QLDAE system is more than that by adding quadratic algebraic equations.

Theorem 6.3.9 *For a polynomial system of order M, the maximum number of state variables in the resulting QLDAE system does not exceed the number of state variables of $[x^T, (x^T)^{\textcircled{2}}, (x^T)^{\textcircled{3}}, \cdots, (x^T)^{\textcircled{M}}]^T$, by adding differential equations that are derived by taking Lie derivatives of changed variables.*

Proof Let $y = x_1^{i_1} \cdots x_N^{i_N}$, where $\sum_{k=1}^{N} i_k \leq M$. Then, $y \in [x^T, (x^T)^{②}, (x^T)^{③}, \cdots, (x^T)^{Ⓜ}]^T$. The differential equation of y is derived by taking its Lie derivative

$$\begin{aligned} \frac{d}{dt}(x_1^{i_1} \cdots x_N^{i_N}) &= \mathfrak{L}_{\dot{x}}(x_1^{i_1} \cdots x_N^{i_N}) \\ &= \sum_{k=1}^{N} x_1^{i_1} \cdots i_k x_i^{i_k - 1} \left(\frac{d}{dt} x_i\right) \cdots x_N^{i_k}. \end{aligned} \tag{6.37}$$

Since $\frac{d}{dt}x_i$ is a polynomial function of maximum order M in x and linear in u, therefore from (6.37), $\mathfrak{L}_{\dot{x}}(x_1^{i_1} \cdots x_N^{i_N})$ is a polynomial function of maximum order $2M-1$ in x and linear in u, *i.e.*, this is a QLDAE system in $[x^T, (x^T)^{②}, \cdots, (x^T)^{Ⓜ}]^T$. ■

6.3.7 Quadratic-Linearization Algorithm

As a summary, we present the quadratic-linearization algorithm for polynomial ODEs in Algorithm 7, which quadratic-linearizes a polynomial system of order M in the form of (6.16) where $c_{i,i} = 1$ and $c_{i,j} = 0$, $\forall i \neq j$.

The computational complexity of Algorithm 7 is polynomial with respect to the order of the polynomial system, because the size of the equivalent quadratic-linear system is a polynomial function of the order of the polynomial system.

6.3.8 Polynomialization and Quadratic-Linearization of DAEs

More generally, systems may be modeled by differential algebraic equations in the form of

$$\frac{d}{dt}(q(x)) = f(x) + Bu. \tag{6.38}$$

Similarly, we deal with equations where each element in $f(x)$, denoted by $f_i(x)$, and each element in $q(x)$, denoted by $q_i(x)$, are summations of elementary functions, *i.e.*,

$$\begin{aligned} f_i(x) =& g_{i,1}(x) + \cdots + g_{i,k_i}(x), \\ q_i(x) =& h_{i,1}(x) + \cdots + h_{i,l_i}(x), \end{aligned} \tag{6.39}$$

where k_i, l_i are constant integers denoting the number of nonlinear functions in f_i and q_i, respectively.

Algorithm 7 Quadratic-Linearization of Polynomial ODEs

Inputs: $E = [\dot{x_1}, \cdots, \dot{x_n}]$, the list of symbolic expressions of the polynomial ODEs.
Outputs: Y_{var}, the set of new variables;
Y_{expr}, the set of expressions of new variables;
E, the list of symbolic expressions of the QLDAEs.

Initialize $Y_{var} \leftarrow \{\}$, $Y_{expr} \leftarrow \{\}$;
repeat
  From E, the list of symbolic expressions of the ODEs, pick a monomial $m(x)$ that has degree > 2 (in terms of x and variables in Y_{var}).
  Find a decomposition of $m(x)$, *i.e.*, find $g(x)$ and $h(x)$ that satisfy $m(x) = g(x) \times h(x)$.
  Define a new state variable $y = g(x)$.
  Add y into Y_{var}, and $g(x)$ into Y_{expr}.
  Compute the symbolic expression of $\dot{y} = g'(x)\dot{x}$.
  Add the symbolic expression of $\dot{y}$ to E.
  for all monomials $m(x)$ in E **do**
    if $m(x)$ is linear or quadratic in terms of x and variables in Y_{var} **then**
      Replace $m(x)$ as a linear or quadratic term.
    end if
  end for
until All expressions in E are linear or quadratic functions of x and variables in Y_{var}.

Polynomialization

The polynomialization procedure is similar to the ODE case. We first introduce new variables

$$\begin{aligned} y_{i,1} = g_{i,1}(x), \cdots, y_{i,k_i} = g_{i,k_i}(x), \\ z_{i,1} = h_{i,1}(x), \cdots, z_{i,l_i} = h_{i,l_i}(x). \end{aligned} \tag{6.40}$$

Therefore, (6.38) is converted to

$$\frac{d}{dt}(z_{i,1} + \cdots + z_{i,l_i}) = y_{i,1} + \cdots + y_{i,k_i} + b_i u, \quad i = 1, \cdots, N. \tag{6.41}$$

If equations in (6.40) can be written in the polynomial form, we can simply add them as polynomial equations.

If equations in (6.40) cannot be written in the polynomial form, we add the differential

equations

$$\begin{aligned}\frac{d}{dt}y_{i,1} &= g'_{i,1}(x)\frac{d}{dt}x, \cdots, \frac{d}{dt}y_{i,k_i} = g'_{i,k_i}(x)\frac{d}{dt}x,\\ \frac{d}{dt}z_{i,1} &= h'_{i,1}(x)\frac{d}{dt}x, \cdots, \frac{d}{dt}z_{i,l_i} = h'_{i,l_i}(x)\frac{d}{dt}x.\end{aligned} \tag{6.42}$$

However, we may not have explicit expressions for $\frac{d}{dt}x$, and we do not reach a polynomial system in the form of (6.16). We therefore extend the definition of a polynomial system to be the following:

Definition 6.3.2 (Generalized polynomial system) *The generalized polynomial system is defined to be of the form (6.43), and it is said to be of order M, if the highest order of monomial of x is M, i.e., $M = \max_{k,i}\left(\sum_{j=1}^{N}\zeta_{i,k,l,j}, \sum_{l=1}^{N}\beta_{i,k,l}, \sum_{l=1}^{N}\gamma_{i,k,l}\right)$.*

$$\begin{aligned}&\sum_l\left[\sum_k \xi_{i,k,l}x_1^{\zeta_{i,k,l,1}}\cdots x_N^{\zeta_{i,k,l,N}}\right]\dot{x}_l\\ =&\sum_k \alpha_{i,k}x_1^{\beta_{i,k,1}}\cdots x_N^{\beta_{i,k,N}} + \left(\sum_k \eta_{i,k}x_1^{\gamma_{i,k,1}}\cdots x_N^{\gamma_{i,k,N}}\right)u\\ &i=1,\cdots,N, \zeta_{i,k,l,j}\geq 0, \beta_{i,k,j}\geq 0, \gamma_{i,k,j}\geq 0,\\ &\zeta_{i,k,l,j}\in\mathbb{Z}, \beta_{i,k,j}\in\mathbb{Z}, \gamma_{i,k,j}\in\mathbb{Z}, \forall k,j,l.\end{aligned} \tag{6.43}$$

With this new definition, and following the proof of 6.3.4, it is straightforward that DAEs can be converted to a generalized polynomial system.

Quadratic-Linearization

Similarly, we extend the definition of QLDAEs to a more general form

$$\frac{d}{dt}\left[C_1x + C_2x\otimes x\right] = G_1x + G_2x\otimes x + (D_1x + D_2x\otimes x)u + Bu. \tag{6.44}$$

Similar to the proof of Theorems 6.3.8 and 6.3.9, it is easy to show that by adding quadratic algebraic equations and differential equations, we can convert a generalized polynomial system (6.43) to a generalized QLDAE system (6.44).

6.4 QLMOR

In this section, we focus on the reduction step for QLDAEs (6.8). Similar, but more complicated derivations can be performed for generalized QLDAEs (6.44).

We start by performing a variational analysis to show the validity of applying Krylov-subspace methods. We then derive the moments of transfer functions of (6.8), which provides a guideline of how to choose the proper Krylov subspace for generating a smallest reduced model in the sense of moment-matching. The derivations in this section unifies previous works on polynomial systems and bilinear systems [106, 107].

6.4.1 Variational Analysis

The variational analysis [105] starts by assuming the QLDAE system is a combination of a series of homogeneous nonlinear sub-systems, whose responses to $u(t)$ are $x_1(t)$, $x_2(t)$, *etc.*. That is, when input is $\alpha u(t)$, the response $x(t)$ is

$$x(t) = \alpha x_1(t) + \alpha^2 x_2(t) + \alpha^3 x_3(t) + \alpha^4 x_4(t) + \cdots . \tag{6.45}$$

Therefore, by inserting (6.45) into (6.8), and equating the terms corresponding to $\alpha^i, i = 1, 2, \cdots$, we obtain

$$\begin{aligned}
C\frac{d}{dt}x_1 =& G_1 x_1 + Bu, \\
C\frac{d}{dt}x_2 =& G_1 x_2 + G_2 x_1 \otimes x_1 + D_1 x_1 u, \\
C\frac{d}{dt}x_3 =& G_1 x_3 + G_2(x_1 \otimes x_2 + x_2 \otimes x_1) + D_1 x_2 u + D_2(x_1 \otimes x_1)u, \\
C\frac{d}{dt}x_4 =& G_1 x_4 + G_2(x_1 \otimes x_3 + x_2 \otimes x_2 + x_3 \otimes x_1) \\
& + D_1 x_3 u + D_2(x_1 \otimes x_2 + x_2 \otimes x_1)u.
\end{aligned} \tag{6.46}$$

From (6.46), if in the i-th set of differential equations (*i.e.*, equations where LHS is $\frac{d}{dt}x_i$), we lump the terms without x_i to be pseudo inputs, then the original system can be viewed as a series of linear systems. Therefore, the Krylov subspace that guarantees that the projected system matches the moments of these linear systems is $\mathcal{K}(G_1^{-1}C, G_1^{-1}([B, G_2, D_1, D_2]))$, if G_1 is non-singular. However, the size of this subspace can easily exceed the size of the original system. Therefore, care must be taken to choose a useful subspace.

6.4.2 Matrix Transfer Functions

Following [105], we can show that transfer functions for (6.8) can be written as follows. (Only first three transfer functions are listed.)

$$H_1(s) = (sC - G_1)^{-1}B, \tag{6.47}$$

$$
\begin{aligned}
H_2(s_1,s_2) =&((s_1+s_2)C-G_1)^{-1}\\
&\frac{1}{2}\Big[G_2(H_1(s_1)\otimes H_1(s_2)+H_1(s_2)\otimes H_1(s_1))\\
&+D_1(H_1(s_1)+H_2(s_2))\Big],
\end{aligned}
\tag{6.48}
$$

$$
\begin{aligned}
&H_3(s_1,s_2,s_3)\\
=&((s_1+s_2+s_3)C-G_1)^{-1}\\
&\frac{1}{6}\Big\{G_2\Big[H_1(s_1)\otimes H_2(s_2,s_3)+H_1(s_2)\otimes H_2(s_1,s_3)\\
&+H_1(s_3)\otimes H_2(s_1,s_2)+H_2(s_2,s_3)\otimes H_1(s_1)\\
&+H_2(s_1,s_3)\otimes H_1(s_2)+H_2(s_1,s_2)\otimes H_1(s_3)\Big]\\
&+D_1\Big[H_2(s_1,s_2)+H_2(s_2,s_3)+H_2(s_1,s_3)\Big]\\
&+D_2\Big[H_1(s_1)\otimes H_1(s_2)+H_1(s_1)\otimes H_1(s_3)+H_1(s_2)\otimes H_1(s_1)\\
&+H_1(s_2)\otimes H_1(s_3)+H_1(s_3)\otimes H_1(s_1)+H_1(s_3)\otimes H_1(s_2)\Big]\Big\}.
\end{aligned}
\tag{6.49}
$$

Based on (6.47), (6.48) and (6.49), the Taylor expansion of these transfer functions can be derived. Correspondingly, we obtain the moments of the transfer functions, *i.e.*, coefficients of s^k for $H_1(s)$, $s_1^k s_2^l$ for $H_2(s_1,s_2)$, *etc.*. For example,

$$
\begin{aligned}
H_1(s) =&\sum_{k=0}^{\infty} M_{1,k}s^k=\sum_{k=0}^{\infty}A^kR_1s^k\\
H_2(s) =&\sum_{k=0}^{\infty}\sum_{l=0}^{\infty}M_{2,k,l}s_1^ks_2^l=\sum_{k=0}^{\infty}A^kR_1(s_1+s_2)^kG_1^{-1}\\
&\frac{1}{2}\Bigg\{G_2\Bigg[\left(\sum_{k=0}^{\infty}A^kR_1s_1^k\right)\otimes\left(\sum_{k=0}^{\infty}A^kR_1s_2^k\right)\\
&+\left(\sum_{k=0}^{\infty}A^kR_1s_2^k\right)\otimes\left(\sum_{k=0}^{\infty}A^kR_1s_1^k\right)\Bigg]\\
&+D_1\left[\sum_{k=0}^{\infty}A^kR_1s_2^k+\sum_{k=0}^{\infty}A^kR_1s_2^k\right]\Bigg\},
\end{aligned}
\tag{6.50}
$$

where $A = G_1^{-1}C$, $R_1 = -G_1^{-1}B$, and $M_{1,k}$, $M_{2,k,l}$ are the moments for $H_1(s)$ and $H_2(s_1, s_2)$, respectively. Similar derivations can be performed for higher order moments.

An interesting point to note is that $H_2(s_1, s_2)$ is the same as the summation of the $H_2(s_1, s_2)$ of a bilinear system $C\frac{d}{dt}x = G_1x + D_1xu + Bu$ and the $H_2(s_1, s_2)$ of a quadratic system $C\frac{d}{dt}x = G_1x + G_2x \otimes x + Bu$.

6.4.3 Subspace Basis Generation for Moment Matching

To illustrate how QLMOR generates the projection matrix, we derive the theorem and algorithm for $H_2(s_1, s_2)$ in the following. These derivations can be easily extended for higher-order moment-matching.

From (6.50), we can derive the vectors that are the coefficients for an n-th order moment. (*E.g.*, for $H_2(s_1, s_2)$, the n-th order moments refers to the set of coefficients for $s_1^k s_2^l, k+l = n$.) As an example, those vectors for $H_2(s_1, s_2)$ are listed in Table 6.3. We then have the following theorem:

Theorem 6.4.1 *Given a QLDAE system (6.8), suppose*

$$\begin{aligned} R(V) =& \{A^i R_1, i \le q\} \bigcup \{A^i G_1^{-1} D_1 A^j R_1, i + j \le q\} \bigcup \\ & \{A^i G_1^{-1} G_2 (A^j R_1) \otimes (A^k R_1), i + j + k \le q, k \le j\} \end{aligned} \tag{6.51}$$

Then if

$$\begin{aligned} x = Vx_r, \quad C_r = V^T CV, \quad B_r = V^T B, \\ G_{r,1} = V^T G_1 V, \quad G_{r,2} = V^T G_2 V \otimes V, \\ D_{r,1} = V^T D_1 V, \quad D_{r,2} = V^T D_2 V \otimes V, \end{aligned} \tag{6.52}$$

the reduced system (6.53) matches the moments of $H_2(s_1, s_2)$ of the original system up to q-th order.

$$C_r \frac{d}{dt} x_r = G_{r,1}x_r + G_{r,2}x_r \otimes x_r + D_{r,1}x_r u + D_{r,2}(x_r \otimes x_r)u + B_r u \tag{6.53}$$

Proof The proof is similar to that in [106]. ■

Notice that the second and the third set of basis vectors in Theorem 6.4.1 count for the moments of the corresponding bilinear system and quadratic system, respectively – this result also follows from the above theorem by setting $G_2 = D_2 = 0$ and $D_1 = D_2 = 0$ in the QLDAE system.

Table 6.3: Moments of $H_2(s_1, s_2)$

Moments	Bilinear system	Quadratic system
0th order	$G_1^{-1}D_1R_1$	$G_1^{-1}G_2R_1 \otimes R_1$
1st order	$AG_1^{-1}D_1R_1$	$AG_1^{-1}G_2R_1 \otimes R_1$
	$G_1^{-1}D_1AR_1$	$G_1^{-1}G_2(AR_1) \otimes R_1$
2nd order	$A^2G_1^{-1}D_1R_1$	$A^2G_1^{-1}G_2R_1 \otimes R_1$
	$AG_1^{-1}D_1AR_1$	$AG_1^{-1}G_2(AR_1) \otimes R_1$
	$G_1^{-1}D_1A^2R_1$	$G_1^{-1}G_2(A^2R_1) \otimes R_1$
		$G_1^{-1}G_2(AR_1) \otimes (AR_1)$
3rd order	$A^3G_1^{-1}D_1R_1$	$A^3G_1^{-1}G_2R_1 \otimes R_1$
	$A^2G_1^{-1}D_1AR_1$	$A^2G_1^{-1}G_2(AR_1) \otimes R_1$
	$AG_1^{-1}D_1A^2R_1$	$AG_1^{-1}G_2(A^2R_1) \otimes R_1$
	$G_1^{-1}D_1A^3R_1$	$G_1^{-1}G_2(A^3R_1) \otimes R_1$
		$AG_1^{-1}G_2(AR_1) \otimes (AR_1)$
		$G_1^{-1}G_2(A^2R_1) \otimes (AR_1)$

A naive generation of the above subspace might lead to numerical stability problems because of the explicit computation of A^i terms. For quadratic systems, [106] computes the starting vectors directly and suggests to perform an orthogonalization between starting vectors. However, that can still lead to numerical stability problems since the starting vectors are computed explicitly.

Here we present a new algorithm to generate the basis for this subspace, as shown in Algorithm 8. The elementary operation in the algorithm is to generate an orthonormal projection matrix $V = [v_0, \cdots, v_q]$ for a Krylov subspace $\mathcal{K}_{q+1}(A, R)$. We employ Arnoldi algorithm [126] in our implementation.

The computational complexity for this algorithm depends on the number of moments to be matched. Suppose we want to match up to the p_0-th order moments of H_1, up to the p_1-th order moments of the bilinear system, and up to the p_2-th order moments of the quadratic system. From Theorem 6.4.1, we know that the sizes of the subspaces for the three classes of moments are $O(p_0)$, $O(\sum_{i=1}^{p_1+1} i) = O(p_1^2)$ and $O(1 + \sum_{i=1}^{p_2} i) = O(p_2^2)$, respectively. Therefore, the size of the final subspace is of size $O(p_0 + p_1^2 + p_2^2)$, and this is also the time and space complexity of the Krylov basis generation algorithm, with the valid assumption that system matrices are sparse. If we choose $p_0 = p_1 = p_2 = q$, then the complexity of the algorithm is $O(q^2)$.

In this algorithm, we avoid the direct computation of A^i in the starting vector for each Krylov subspace. We use $G_1^{-1}D_1v_i$ instead of $G_1^{-1}D_1A^iR_1$, and $G_1^{-1}G_2v_i \otimes v_j$ instead of $G_1^{-1}G_2(A^iR_1) \otimes (A^jR_1)$, where v_i is the i-th vector in the Krylov subspace $\mathscr{K}(A, R_1)$.

We can prove that $R_1(V) = \{A^iR_1, i \leq q\} \bigcup \{A^iG_1^{-1}D_1A^jR_1, i+j \leq q\} \bigcup \{A^iG_1^{-1}G_2(A^jR_1) \otimes (A^kR_1), i+j+k \leq q, k \leq j\}$ and $R_2(V) = \{v_i, i \leq q\} \bigcup \{A^iG_1^{-1}D_1v_j, i+j \leq q\} \bigcup \{A^iG_1^{-1}G_2v_j \otimes v_k, i+j+k \leq q, k \leq j\}$ span the same subspace (proof by induction). Therefore, in the

computation of the starting vector, no A^i terms are involved.

We can also prove that $v_{i_1} \otimes v_{j_1}$ is orthogonal to $v_{i_2} \otimes v_{j_2}$ if $i_1 \neq i_2$ or $j_1 \neq j_2$ (by definition of Kronecker product). Hence, the starting vectors have better numerical properties than [106]. However, it is not always the case that $G_1^{-1}G_2 v_{i_1} \otimes v_{j_1}$ is orthogonal to $G_1^{-1}G_2 v_{i_2} \otimes v_{j_2}$, even if $v_{i_1} \otimes v_{j_1}$ is orthogonal to $v_{i_2} \otimes v_{j_2}$. Therefore, one further improvement of this algorithm is to perform an orthonormalization of the starting vector to the basis already generated.

Proofs are omitted here due to page limits.

Algorithm 8 Generation of basis for subspace in Theorem 6.4.1

Inputs: C, G_1, G_2, D_1, D_2 matrices defining the QLDAEs;
q: the reduced model will match up to the q-th order moments of H_1 and H_2.
Outputs: V: the projection matrix.

1: $A = G_1^{-1}C$, $R_1 = -G_1^{-1}B$.
2: Generate matrix $V = [v_0, \cdots, v_q]$ for $\mathcal{K}_{q+1}(A, R_1)$
3: **for** $i = 0$ to q **do**
4: Generate matrix $W = [w_0, \cdots, w_{q-i}]$ for $\mathcal{K}_{q-i+1}(A, G_1^{-1}D_1 v_i)$. {subspace for moments of the bilinear system}
5: $V = qr([V, W])$.
6: **end for**
7: **for** $i = 0$ to q **do**
8: **for** $j = 0$ to $\min(q-i, i)$ **do**
9: Generate matrix $W = [w_0, \cdots, w_{q-i-j}]$ for $\mathcal{K}_{q-i-j+1}(A, G_1^{-1}G_2 v_i \otimes v_j)$. {subspace for moments of the quadratic system}
10: $V = qr([V, W])$.
11: **end for**
12: **end for**

6.4.4 Discussion on Local Passivity Preservation

The loss of passivity in [106, 107, 121] are mainly caused by Taylor approximation. QLMOR alleviates this problem in the sense that the conversions to the polynomial system and the quadratic-linear system are exact, and no passivity loss is introduced in that step.

In the reduction step, however, the passivity may again be lost. One may show that our reduced model linearized at any $x_{r,0} \in \mathbb{R}^{N_r}$, is a congruence transformation of the original system linearized at $x_0 = V x_{r,0} \in \mathbb{R}^N$. Under somewhat strict conditions such as the ones in [11], the passivity of the linearized system can be preserved.

6.4.5 Discussion on Constructing Appropriate QLDAEs

As mentioned in Section 6.3, different QLDAE formulations are all equivalent. However, the reduced QLDAEs are generally not equivalent. We discuss how different QLDAE formulations lead to different reduced models.

Consider a nonlinear system,

$$\frac{d}{dt}x = f(x) + Bu, \tag{6.54}$$

from which we can derive QLDAEs

$$\frac{d}{dt}x_q = G_1 x_q + G_2 x_q \otimes x_q + D_1 x_q u + D_2 x_q \otimes x_q u + Bu, \tag{6.55}$$

where the state variables are $x_q = (x, y)$ where $y = g(x)$ are nonlinear functions of x.

According to Theorem 6.4.1, by choosing appropriate projection matrices, we can match the first few moments of (x, y). Therefore, by formulating different QLDAEs where different y's are added, the moments of different y's are matched in different reduced models. This result is of great importance in practice – depending on the output signal, which is a nonlinear function of state variables, we may compute reduced models that matches moments of these output signals.

There is another subtle but important guideline of choosing new state variables – only add variables $y = g(x)$ so that $y_{DC} = g(x_{DC}) = 0$ where x_{DC} is the DC solution of (6.54) for $u = u_{DC}$. (Without loss of generality, we assume that $x_{DC} = 0$.)

To see why this is useful, we consider two QLDAEs that are equivalent to (6.54), and are in terms of x_α and x_β, respectively:

$$\frac{d}{dt}x_\alpha = G_{1,\alpha} x_\alpha + G_{2,\alpha} x_\alpha \otimes x_\alpha + D_{1,\alpha} x_\alpha u + D_{2,\alpha} x_\alpha \otimes x_\alpha u + B_\alpha u, \tag{6.56}$$

$$\frac{d}{dt}x_\beta = G_{1,\beta} x_\beta + G_{2,\beta} x_\beta \otimes x_\beta + D_{1,\beta} x_\beta u + D_{2,\beta} x_\beta \otimes x_\beta u + B_\beta u. \tag{6.57}$$

The state variables x_α and x_β satisfy 1) $x_\alpha = 0$ when $u = u_{DC}$; 2) $x_\beta = x_\alpha + x_0$, where x_0 is a non-zero constant vector.

Generally, $G_{1,\alpha} \neq G_{1,\beta}$, $G_{2,\alpha} \neq G_{2,\beta}$, $D_{1,\alpha} \neq D_{1,\beta}$, $D_{2,\alpha} \neq D_{2,\beta}$, $B_\alpha \neq B_\beta$. Therefore the moments of two QLDAEs are not the same. This, however, does not imply that the two QLDAEs are not equivalent. Rather, loosely speaking, it implies that the moments of systems "quadratic-linearized around different states" are not the same.

To see that, we note that $G_{1,\alpha}$ is the Jacobian matrix of RHS of (6.56) computed at $x_\alpha = 0 = x_{\alpha,DC}$, and $G_{1,\beta}$ is the Jacobian matrix of RHS of (6.57) computed at $x_\beta = 0$. However, when $u = u_{DC}$, $x_\beta = x_\alpha + x_0 = x_0 \neq 0$. Therefore, in order to match moments of the original system (6.54) around x_{DC}, it is inappropriate to use $G_{1,\beta}$ in moment computations,

and it is inappropriate to use (6.57) in QLMOR.

Rather, we can show that the moments of (6.56) linearized around 0 match the moments of (6.54) linearized around x_{DC}, thus leading to the guideline of choosing y such that $y_{DC} = 0$. The proof of this result involves a bit linear algebra, but is quite straightforward, and is omitted here.

6.4.6 Computational Cost of Simulation of the Reduced Model

The computational cost of simulating a dynamical system is composed of two parts: function (including Jacobian) evaluation and linear solve.

The quadratic-linear reduced models only involve quadratic nonlinearities, and therefore the function computation only involves multiplications, and the Jacobian matrix is a linear function of x. This is an advantage over previous higher-order reduced models, as well as complex models that involve expensive exponential function computations and complicated flow-control statements. Additionally, because the size of the model is N_r (much smaller than N), the linear solve is also expected to be much faster.

However, there is one important source that slows down the computation – the sparsity of the system is destroyed in the reduced models. It is not easy to estimate how this sparsity-loss affect the computational cost, but in practice we observe speedups of reduced models over original models.

Compared to trajectory-based methods, our reduced model has a more tractable computational cost for the function and Jacobian evaluations. Assuming that we have N_r state variables in the reduced model, the function evaluation is bounded by $O(N_r^3)$ multiplications (assuming the worst case where there are $O(N_r^2)$ quadratic terms for q equations); the complexity of Jacobian matrix evaluation is $O(N_r^2)$ because it is a linear function of state variables. On the other hand, trajectory-based methods try to compute $V^T f(V x_r)$ by using interpolation among sample points stored in the reduced model. To obtain a model with high-fidelity, however, a very large number of sample points need to be stored, and the computation can become very slow.

6.4.7 Multi-Point Expansion

The multi-point Krylov subspace method [127] can be directly applied in QLMOR, and it potentially leads to a smaller and more accurate model for a specified frequency range of interest.

Also note that the quadratic-linearization procedure in Section 6.3 might render a QLDAE system where G_1 is *singular*. This can be a potential problem for generating Krylov subspaces at $s = 0$. The current workaround for this problem is to generate Krylov subspaces at $s = s_0, s_0 \simeq 0$.

6.5 Examples and Experimental Results

In this section, we illustrate the applicability of the QLMOR method presented in the previous sections, demonstrating its efficiency and accuracy on example circuits and systems.

6.5.1 A System with a $\frac{x}{1+x}$ Nonlinearity

As noted in Section 6.3, x, $x \otimes x$ and $\frac{x}{K+x}$ are common nonlinearities in biochemical rate equations. To test how QLMOR works, a random second-order polynomial system with a $\frac{x}{1+x}$ function was generated to mimic a biochemical reaction system[4]:

$$\frac{d}{dt}x + G_1 x + G_2 x \otimes x + e_1 \frac{10x_1}{1+x_1} + Bu = 0. \tag{6.58}$$

The size of (6.58) is 10, hence $G_1 \in \mathbb{R}^{10\times 10}, G_2 \in \mathbb{R}^{10\times 100}, B \in \mathbb{R}^{10\times 1}, e_1 \in \mathbb{R}^{10\times 1} = [1, 0, \cdots, 0]^T$. Notice that we manually make G_1 dense, non-symmetric, and make its eigenvalues uniformly distributed. Those factors all make MOR for this system more difficult than for many real systems that are normally sparse and symmetric.

To compare with the QLMOR method, we first show that Taylor-expansion based methods [106, 107] are not applicable. The Taylor series of the function $\frac{1}{1+x}$ is

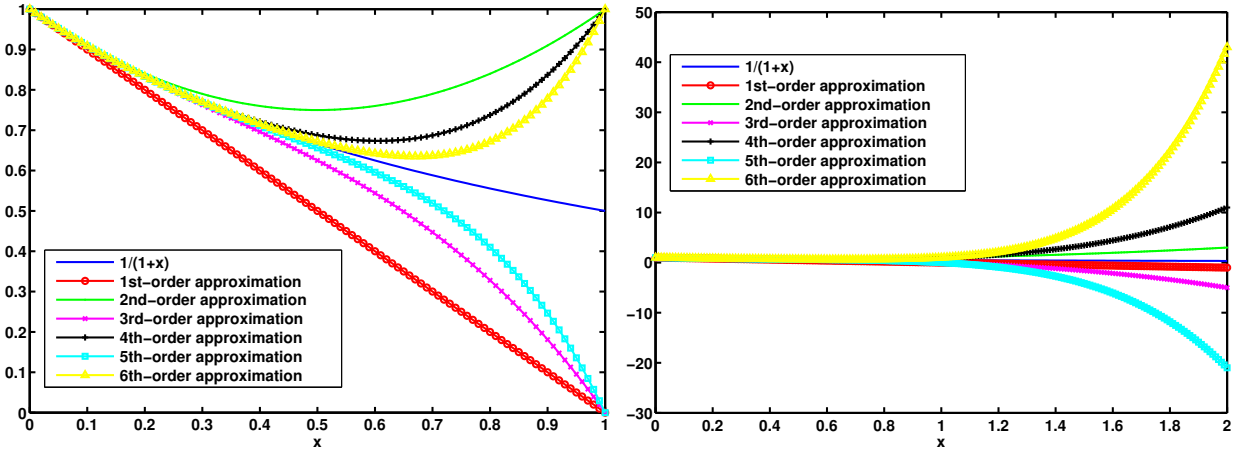
$$\frac{1}{1+x} = \sum_{i=0}^{\infty}(-x)^i = 1 - x + x^2 - x^3 + x^4 - x^5 + \cdots, (|x| < 1). \tag{6.59}$$

Notice that (6.59) converges only when $|x| < 1$ (also clearly seen in Fig. 6.3), making the Taylor approximation irrelevant and highly inaccurate for $|x| \geq 1$; as a result, any reduced model derived from this approximation cannot be accurate for $|x| \geq 1$. Indeed, the Taylor approximation model turns out to be an unstable system for polynomial orders from 2 to 7, *i.e.*, the polynomial model has a finite escape time [73]. As mentioned in [107], such approximations do not preserve properties such as stability and passivity, except by chance.

To apply the QLMOR method, we perform quadratic-linearization of the original system by making the variable change $y = \frac{x_1}{1+x_1}$. Following the procedure in Section 6.3, we obtain a QLDAE system equivalent to (6.58):

$$\begin{aligned} \frac{d}{dt}x + G_1 x + G_2 x \otimes x + e_1 10y + Bu = 0, \\ -x_1 + y + x_1 y = 0. \end{aligned} \tag{6.60}$$

[4]The eigenvalues of G_1 were manipulated to be positive such that the linearized system at 0 is stable. The eigenvalues are uniformly sampled on $[0, 20]$. G_2 was made sparse (5% non-zeros), which is approximately of the same order in real rate equations and circuit equations. The coefficient in front of $\frac{x_1}{1+x_1}$ were made large enough to make the system exhibit high nonlinearity.



(a) (b)

Figure 6.3: Polynomial approximations of $1/(1+x)$. Fig. 6.3(a) shows the interval $x \in [0, 1]$. Fig. 6.3(b) shows the interval $x \in [0, 2]$.

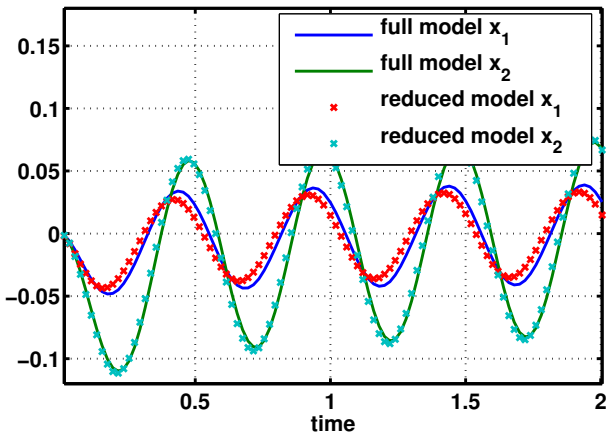
Notice that the size of (6.60) is 11, only one more than the size of the original system.

Applying QLMOR on (6.60), a size 6 reduced model is generated that matches 4 moments of $H_1(s)$ and 2 moments of $H_2(s_1, s_2)$. We then apply a "small" and "fast" input signal $u(t) = \cos(4\pi t)$ so that small higher-order harmonic distortions are generated in the output waveform, as plotted in Fig. 6.4(a). To assess the accuracy of the reduced order model, we also plot the first few Fourier coefficients of $x_1(t)$ in Fig. 6.4(b). (Because $x_1(t)$ exhibits the largest nonlinearities compared to the other state variables, we focus on its waveforms to evaluate the accuracy of the reduced model.) We also excite this system by a "large" and "slow" input signal $u(t) = 10\cos(2\pi t)$, so that large higher-order harmonic distortions are generated in the output waveform, as plotted in Fig. 6.4(c). The speedup of the transient simulation of the reduced model over the original model is about 1.5× – although the model size is nearly halved, the reduced $\hat{G}_2$ matrix becomes dense, and therefore, the speedup is not huge. However, considering that the original model size is very small, a speedup of 1.5× is, in fact, not inconsiderable.

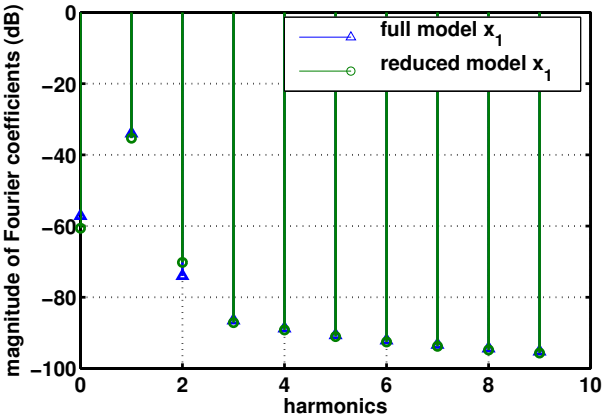
We further manipulate the system (6.58) to make it more nonlinear, and see how QLMOR behaves. In this case, we simply change the nonlinear term to be a function of ${x_2}^5$, *i.e.*,

$$\frac{d}{dt}x + G_1 x + G_2 x \otimes x + e_1 \frac{10x_2}{1+x_2} + Bu = 0. \tag{6.61}$$

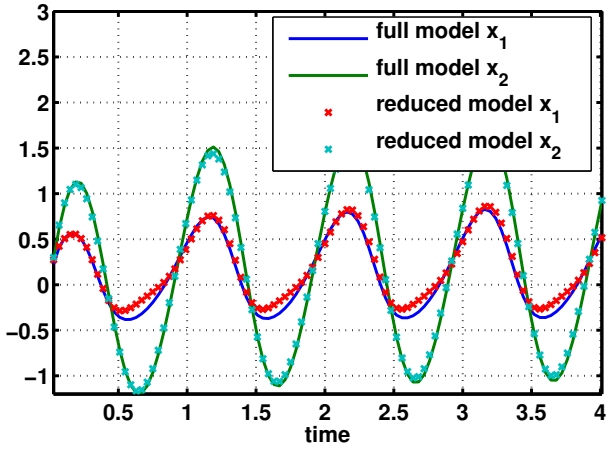
[5]We observe in simulations that the outputs of this system exhibit larger high-order harmonic distortions than (6.60).



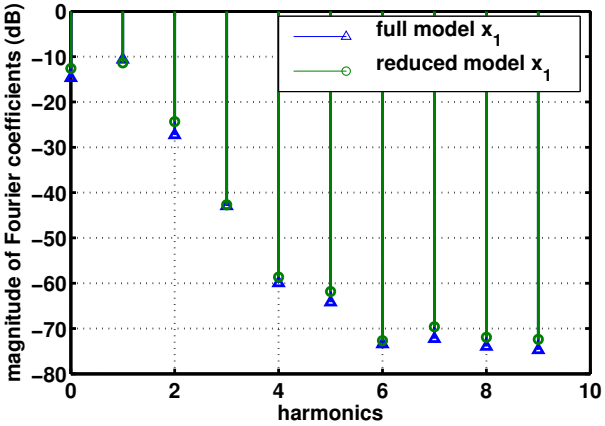
(a) Time domain.



(b) Frequency domain.



(c) Time domain.



(d) Frequency domain.

Figure 6.4: Fig. 6.4(a), Fig. 6.4(b) show the time-domain waveforms of x_1, x_2 and Fourier coefficients of x_1, respectively, when $u(t) = \cos(4\pi t)$. Fig. 6.4(c), Fig. 6.4(d) show the time-domain waveforms of x_1, x_2 and Fourier coefficients of x_1, respectively, when $u(t) = 10\cos(2\pi t)$.

The same model reduction procedure is repeated. Results are plotted in Fig. 6.5 for inputs $u(t) = 8\cos(3\pi t)$ and $u(t) = 10\cos(3\pi t)$. In this case, strong nonlinearities are present in the waveforms, but the QLMOR model still produces quite accurate results. An interesting point to note is that at time $t \simeq 0.5$, the reduced model does not quite capture a fast transient well. This is only to be expected, however, since MOR based on moment matching tends to reduce size by eliminating fast components in the system, while maintaining fidelity for slower components: observe that once the fast transient dies out, the QLMOR model is far more accurate. This inaccuracy at higher frequencies is also apparent from the harmonics shown in Fig. 6.5(d).

6.5.2 A Nonlinear Transmission Line Circuit

We consider the nonlinear transmission line circuit shown in Fig. 6.6 [2]. All resistors and capacitors are set to 1 and the diode I-V characteristic is $i_D = e^{40v_D} - 1$. The input is set to the current source $i = u(t)$; the output is the voltage at node 1.

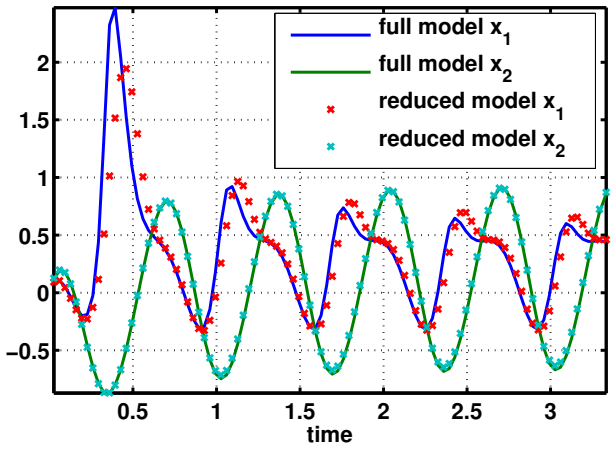
The modified nodal equations for this circuit are

$$\begin{aligned}
\dot{v_1} &= -2v_1 + v_2 + 2 - e^{40v_1} - e^{40(v_1 - v_2)} + u(t), \\
\dot{v_i} &= -2v_i + v_{i-1} + v_{i+1} + e^{40(v_{i-1} - v_i)} - e^{40(v_i - v_{i+1})}, \quad 2 \le i \le N-1, \\
\dot{v_N} &= -v_N + v_{N-1} - 1 + e^{40(v_{N-1} - v_N)}.
\end{aligned} \tag{6.62}$$

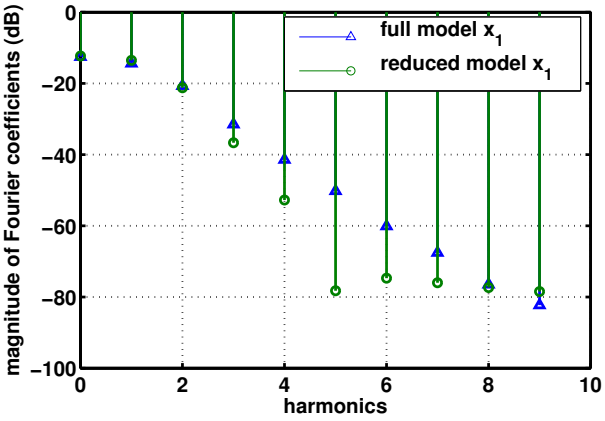
To perform quadratic-linearization of (6.62), if we define new variables to be uni-variable functions, we have to at least make the variable changes $y_{i1} = e^{40v_i}$ and $y_{i2} = e^{-40v_i}$, which results in a third-order polynomial system – the expanded system is at least of size $3N$. However, note that if the state variables are set to be $v_1, v_{i,i+1}, 1 \le i \le N-1$ ($v_{i,i+1} = v_i - v_{i+1}$), we obtain

$$\begin{aligned}
\dot{v_1} &= -v_1 - v_{12} + 2 - e^{40v_1} - e^{40v_{12}} + u(t), \\
\dot{v_{12}} &= -v_1 - 2v_{12} + v_{23} + 2 - e^{40v_1} - 2e^{40v_{12}} + e^{40v_{23}} + u(t), \\
\dot{v_{i,i+1}} &= -2v_{i,i+1} + v_{i-1,i} + v_{i+1,i+2} + e^{40v_{i-1,i}} - 2e^{40v_{i,i+1}} \\
&\qquad + e^{40v_{i+1,i+2}}, \quad (2 \le i \le N-2), \\
\dot{v_{N-1,N}} &= -2v_{N-1,N} + v_{N-2,N-1} + 1 + e^{40v_{N-2,N-1}} - 2e^{40v_{N-1,N}}.
\end{aligned} \tag{6.63}$$

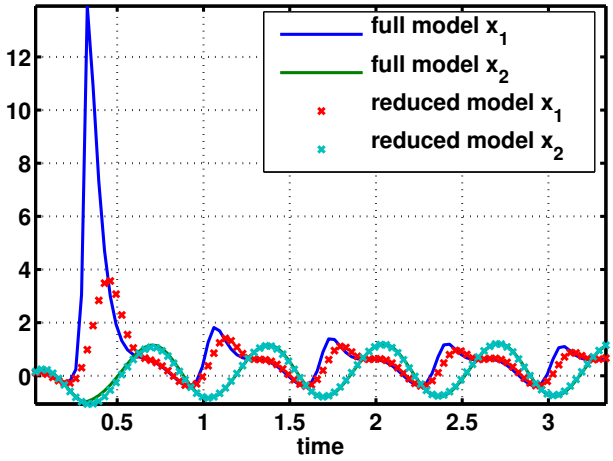
Now, making the variable change $y_1 = e^{40v_1} - 1$ and $y_i = e^{40v_{i-1,i}} - 1, 2 \le i \le N$, the



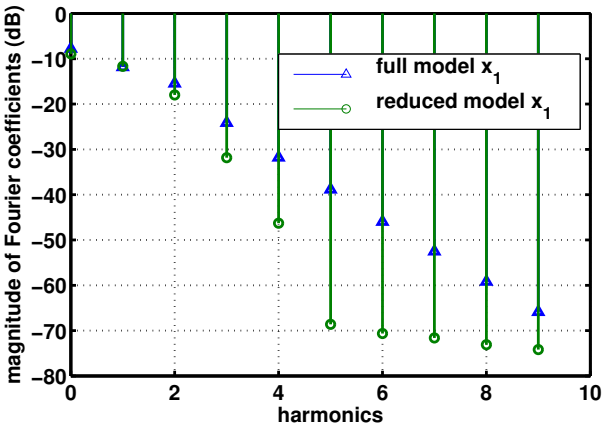
(a) Time domain.



(b) Frequency domain.



(c) Time domain.



(d) Frequency domain.

Figure 6.5: Fig. 6.5(a), Fig. 6.5(b) show the time-domain waveforms of x_1, x_2 and Fourier coefficients of x_1, respectively, when $u(t) = 8\cos(3\pi t)$. Fig. 6.5(c), Fig. 6.5(d) show the time-domain waveforms of x_1, x_2 and Fourier coefficients of x_1, respectively, when $u(t) = 10\cos(3\pi t)$.

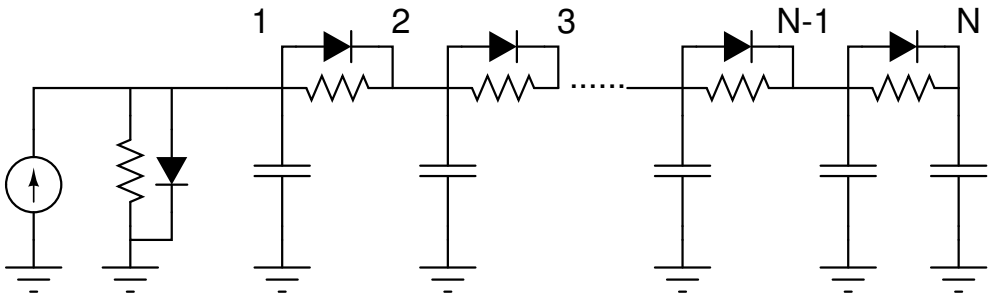


Figure 6.6: Nonlinear transmission line circuit [2].

differential equations for the y_is can be written as

$$\begin{aligned}
\dot{y_1} &= 40(y_1+1)(-v_1 - v_{12} - y_1 - y_2 + u(t)),\\
\dot{y_2} &= 40(y_2+1)(-v_1 - 2v_{12} + v_{23} - y_1 - 2y_2 + y_3 + u(t))a,\\
\dot{y_i} &= 40(y_i+1)(-2v_{i-1,i} + v_{i-2,i-1} + v_{i,i+1} + y_{i-1} - 2y_i + y_{i+1}),\\
\dot{y_N} &= 40(y_N+1)(-2v_{N-1,N} + v_{N-2,N-1} + y_{N-1} - 2y_N).
\end{aligned} \tag{6.64}$$

Therefore, (6.63) and (6.64) together are already in QLDAE form with size $2N$. This size is far less than the system size that would result from a second-order Carleman bilinearization [107] (*i.e.*, $N + N^2$). This example illustrates that if we use **branch voltages** as state variables, a much smaller QLDAE system can result. Intuitively speaking, this is reasonable since the device models are usually defined in terms of **branch voltages** and **branch currents**. This suggests that sparse tableau like circuit equation formulations [128], where the state variables are branch voltages and currents, can be useful for quadratic-linearization.

We apply QLMOR on this nonlinear transmission line circuit with $N = 10$ and compare against weakly-nonlinear Taylor-expansion-based MOR methods, setting the reduced size to 7.[6] To drive the circuit sufficiently hard to produce large harmonic distortions, we apply $u(t) = 1.8\sin(2\pi \times 0.1t)$ and plot waveforms in Fig. 6.7. As expected, the Taylor-based weakly nonlinear reduced model matches the Taylor approximation of the original system well; however, the original Taylor approximation itself does of course not match the full model, because of its validity is limited to "small" inputs. QLMOR, on the other hand, produces a reduced model which matches the original system well, the relative error being about 1%.

To illustrate that QLMOR scales to larger system sizes, we apply QLMOR to nonlinear transmission line circuits of 50,100,150,200 stages. To make a fair comparison, we tune QLMOR-reduced model sizes such that in several simulations, the ratio of the maximum

[6]The size $q = 7$ is obtained by trial-and-error. With q larger than 7, QLMOR gives almost the same result as that of $q = 7$.

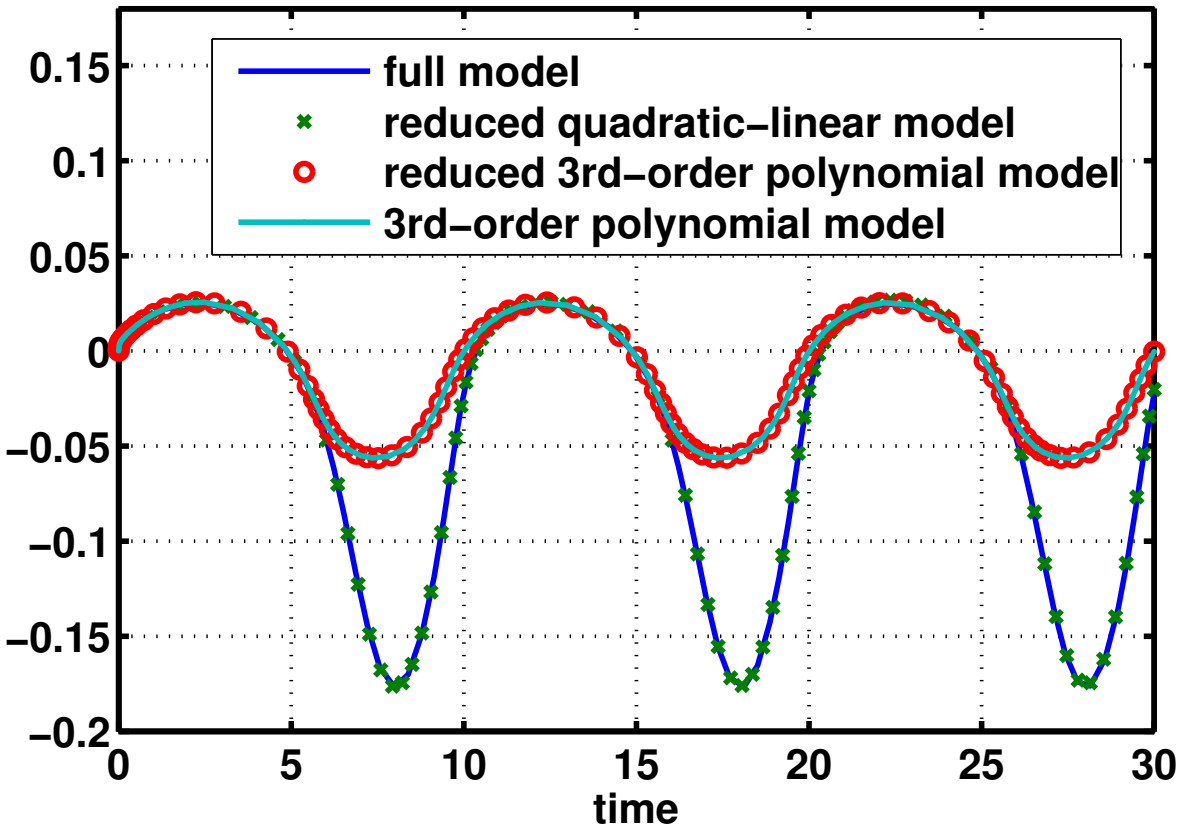
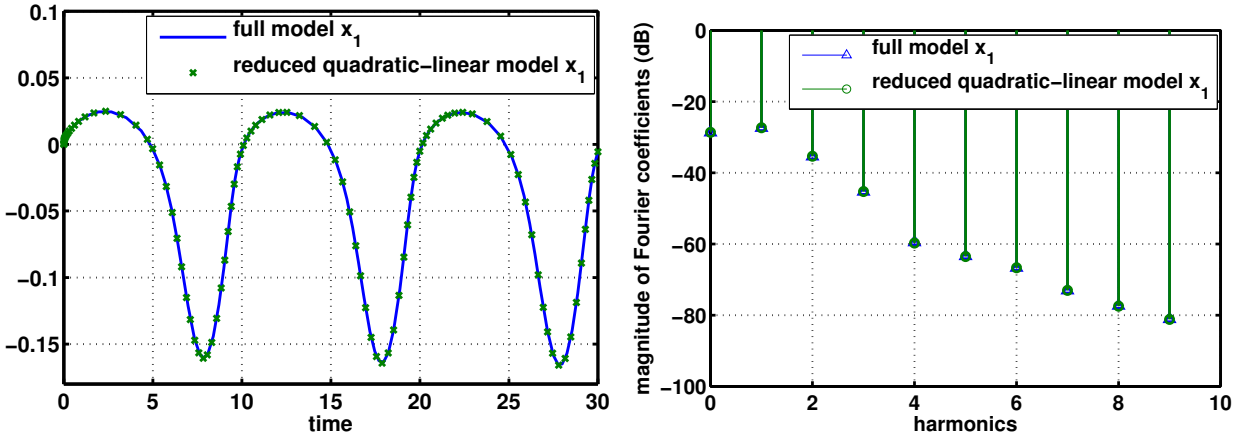


Figure 6.7: Time-domain waveforms (x_1) of full model, quadratic-linear reduced model, 3rd-order polynomial reduced model and 3rd-order polynomial model. ($u(t) = 1.8\sin(2\pi \times 0.1t)$)

absolute error of the time-domain waveform to its amplitude is within 1%. As a result, we produce reduced models of size 11, 15, 18, 20 corresponding to 50,100,150,200 stages, respectively. Again, we excite large harmonic distortions in the simulation, and a typical transient waveform is shown in Fig. 6.8 (for the circuit with 50 stages).

As comparison, Taylor-based weakly nonlinear MOR did not succeed in producing a reduced model on account of shortage of memory for computing $V^T G_3(V \otimes V \otimes V)$. For example, when $N = 50$, $q = 11$, $V \in \mathbb{R}^{50\times 11}$, we have $V \otimes V \otimes V \in \mathbb{R}^{(50\times 11)^3}$. Hence, suppose the size of a double-precision number is 8 bytes, $V \otimes V \otimes V$ requires a minimum memory of $(50 \times 11)^3 \times 8$ Bytes, which is 1.2396 GB.



(a) Time domain.

(b) Frequency domain.

Figure 6.8: Fig. 6.8(a), Fig. 6.8(b) show the time-domain waveforms and Fourier coefficients of x_1, respectively, when $u(t) = 1.8\cos(2\pi \times 0.1t)$. The size of the full model is 50; the size of the reduced model is 11.

Chapter 7

NTIM: Phase/Timing Macromodeling

In this chapter, we present a special macromodeling technique that captures phase responses of nonlinear systems. While the phase response is yet to be defined, it can be intuitively thought of as timing behaviors such as delay or phase modulation, and therefore is an extremely important behavior in many practical systems, especially in circuits.

We formally define the phase response and discuss its applications. We derive a phase macromodel via a nonlinear perturbation analysis, and present numerical methods to compute the phase macromodel. We show its applications in modeling circuits and neurons, and compare the results with the full model in terms of accuracy and computational cost.

7.1 Phase Response and its Applications

In many systems, due to design intents or physical constraints, the waveforms of state variables are almost the same, and therefore can be characterized by the phase information. For example: in phase modulation circuits, we aim to design a system that transforms the input signal $u(t)$ and a carrier signal $x(t)$ to a output signal $y(t) = x(t + u(t))$. In digital circuits, outputs of waveforms typically saturate at the power supply voltages, and the waveforms all look like a ramp waveform, either from "0" to "1" or from "1" to "0". However, the exact shape of waveforms are not the same under different input signals, and these shapes may be characterized by the point-to-point delay information, or equivalently the phase information.

In fact, using phase information alone, we may represent a very rich set of waveforms.

Example 7.1.1 *Suppose* $x(t) = \sin(2\pi t)$*, and*

$$y(t) = x(t + z(t)) = \sin(2\pi(t + z(t))) \tag{7.1}$$

By choosing $z(t)$ *carefully,* $y(t)$ *can represent a very rich set of waveforms. As shown in Fig. 7.1,*

1. $z_1(t) = -1/4$, $y_1(t) = \sin(2\pi(t - 1/4))$ *is* $x(t)$ *delayed by* $1/4$*;*
2. $z_2(t) = t$, $y_2(t) = \sin(4\pi t)$ *doubles the frequency of* $x(t)$*;*
3. $z_3(t) = -t + \frac{1}{2\pi}\sin^{-1}(2s(t - 0.5) - 1)$ *(*$s(t)$ *is a step function)* , $y_3(t) = 2s(t - 0.5) - 1$ *is a square waveform;*
4. $z_4(t) = \cos(2\pi t)$, $y_4(t) = \sin(2\pi(t + \cos(2\pi t)))$ *is a phase modulated signal;*
5. $z_5(t) = -t + \frac{1}{2\pi}\sin^{-1} y_5(t)$, $y_5(t) = \frac{1}{\sqrt{5/4}}\left(\sin(2\pi t) + \frac{1}{2}\sin(6\pi t)\right)$ *is a sinusoidal signal with large third-order harmonic distortion.*

Generally, given $x(t)$ and $y(t)$ with $\max_t x(t) \geq \max_t y(t)$ and $\min_t x(t) \leq \min_t y(t)$, we can always find $z(t)$ such that $y(t) = x(t + z(t))$. In the case where $x(t) = \sin t$, we have $z(t) = \sin^{-1} y(t) - t$. More generally, if $x(t)$ is a vector, representing a trajectory in the state space, then $y(t) = x(t + z(t))$ can be used to represent any other waveforms that have the same trajectory in the state space.

While the phase information alone is powerful in representing waveforms, it is not capable of representing any waveform whose trajectory deviates from the original trajectory. In that case, if the original waveform is $x_S(t)$, and the new waveform is $x_p(t)$, then

$$x_p(t) = x_S(t + z(t)) + y(t), \tag{7.2}$$

where $y(t)$ is not identically equal to 0. However, we may choose $z(t)$ such that $x_S(t + z(t))$ approximates $x_p(t)$ in some sense. In the setting of differential equations, we will define such a $z(t)$ to be the phase response. Formally, we have the following definition.

Definition 7.1.1 (Phase response) *Consider a nonlinear dynamical system described by a set of differential algebraic equations*

$$\frac{d}{dt}q(x(t)) + f(x(t)) + b(t) = 0. \tag{7.3}$$

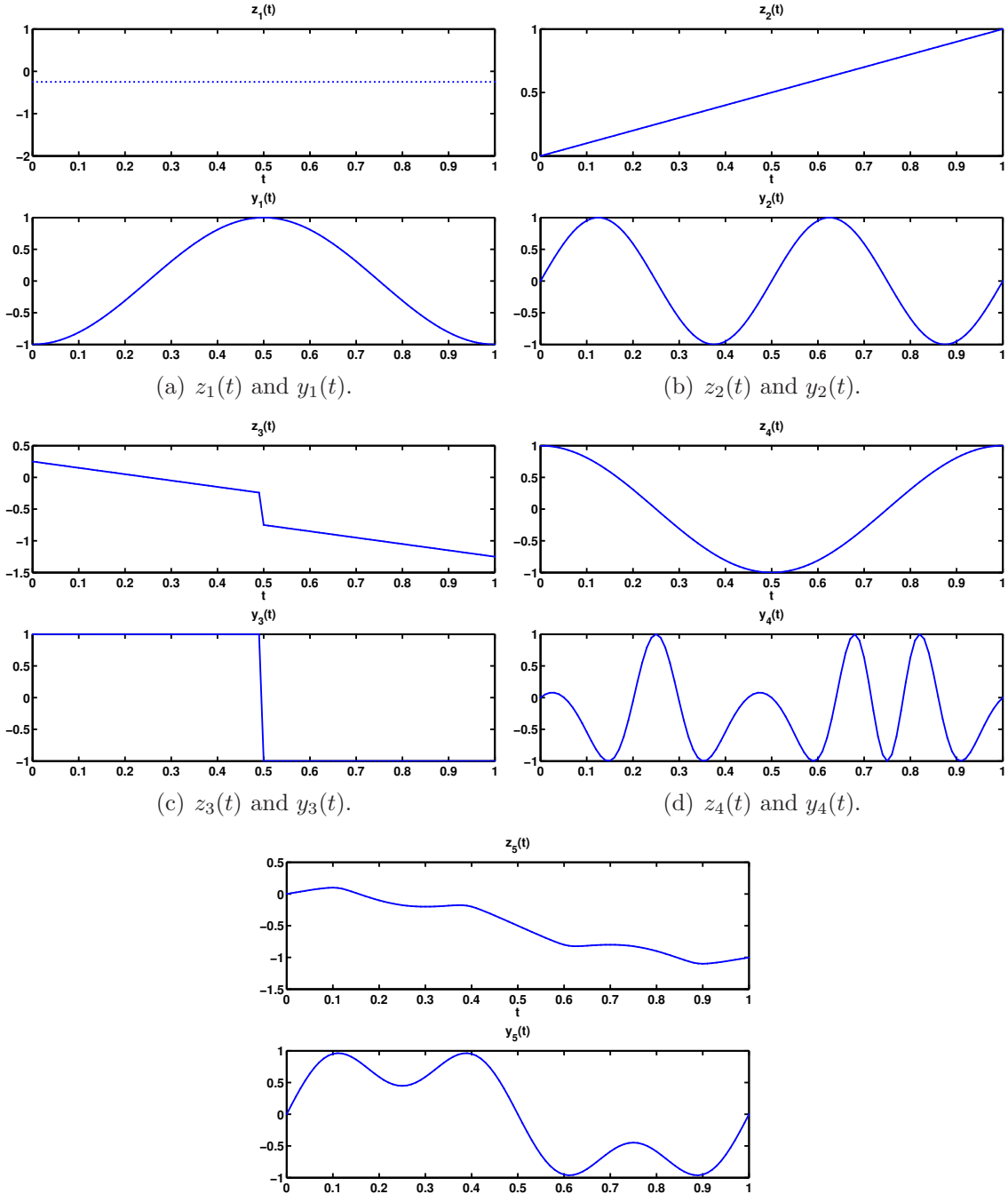
Suppose the (unperturbed) solution of (7.3) to (unperturbed) input $b_S(t)$ *is* $x_S(t)$*, and the (perturbed) solution of (7.3) to (perturbed) input* $b_p(t)$ *is* $x_p(t)$*. There exists* $z(t) : \mathbb{R}^+ \to \mathbb{R}$*, such that* $x_S(t + z(t))$ ***best*** *approximates* $x_p(t)$ *in some sense*[1]*.*

We call z *the phase variable, and* $z(t)$ *the phase response.*

Notice that $x_S(t)$ may be any waveform in this definition. Indeed, we may derive $b_S(t)$ by plug $x_S(t)$ in (7.3), *i.e.*,

$$b_S(t) = -\left(\frac{d}{dt}q(x_S(t)) + f(x_S(t))\right). \tag{7.4}$$

[1]The notion of **best approximation** is defined in Section 7.4.



(a) $z_1(t)$ and $y_1(t)$.

(b) $z_2(t)$ and $y_2(t)$.

(c) $z_3(t)$ and $y_3(t)$.

(d) $z_4(t)$ and $y_4(t)$.

(e) $z_5(t)$ and $y_5(t)$.

Figure 7.1: Illustration of representativeness of phase response.

This definition can also be understood from a geometrical view point. As shown in Fig. 7.2[2], the solid (blue) orbit is the trajectory to input $b_S(t)$, and the dotted (red) orbit is the trajectory to input $b_p(t)$. At time t, on the orbit of $x_S(t)$, there exists a point given by $x_S(t+z(t))$ which best approximates $x_p(t)$ on the perturbed orbit. The remaining difference $y(t) = x_p(t) - x_S(t + z(t))$ is called the **amplitude deviation** [29]. Therefore, we can view the response to input $b_p(t)$ as combined effects of the phase response and the amplitude deviation, *i.e.*,

$$x_p(t) = x_S(t + z(t)) + y(t). \tag{7.5}$$

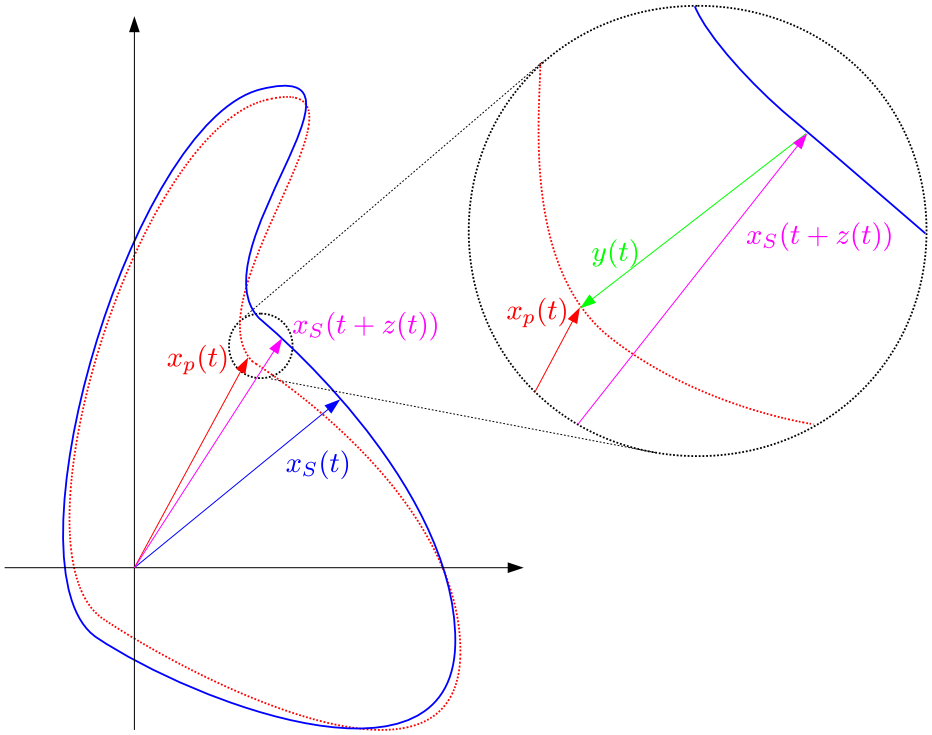


Figure 7.2: Illustration of phase response.

Remark This definition is dependent on the trajectory of $x_S(t)$, and therefore $b_S(t)$. Therefore, it may be called a x_S-PR (phase response).

With this definition of phase response, it is worth noting that although the phase response is an important characteristic of oscillators as pioneered by many researchers [29, 129], it is

[2]We show two closed orbits in Fig. 7.2 for simplicity, but there is no restriction of the orbit being closed in our definition. The orbit can be arbitrary.

not a special characteristic only for oscillators, but generally for all kinds of dynamical systems. It has broader applications in many fields.

Phase response is of great importance in circuits in particular. For examples, in oscillators, phase noise or timing jitter is the phase response in a stochastic setting where noise/uncertainties are present; in digital gates and RC interconnects, timing information such as delay and slope is essentially derived from the phase response; in a pulse generator, the pulse width can also be derived from the phase response.

Scientists in biology and neuroscience also place a great emphasis on quantitative understanding of the phase response. For examples, normal human circadian rhythm disorders, which can be caused by insomnia, fatigue and jet lag, are phase responses of underlying physiological processes under different perturbations; synchronization of a neural network is a phenomenon where the phase responses of all neurons settle down to the same value; some neuron firing/bursting behaviors, which are exhibited when currents are injected, can also be understood as the phase response of the neuron.

7.2 Phase Macromodels

With the above definition of the phase response, the phase macromodel is a model that describes the dynamics of the phase response. In another word, the phase macromodel is composed of a differential equation in terms of $z(t)$, and therefore is a 1-dimensional reduced order model.

To use the phase macromodel, we simulation the differential equation for $z(t)$, instead of (7.3), and use $x_S(t + z(t))$ to approximate the original solution $x_p(t)$. With the assumption that $y(t)$ is small, $x_S(t + z(t))$ will give a very good approximation to $x_p(t)$.

In many systems, the validity of this assumption depends on both the system properties as well as input perturbations. Fortunately, we see a lot of applications that this assumption is satisfied. For examples: in oscillators that are asymptotically orbital stable, the periodic orbit is an attractor which by definition attracts all trajectories nearby; in digital applications, although circuits never reside in a periodic steady state, their trajectories follow almost the same path since inputs are basically signals switching between "0" and "1" with variations of the slope, to the first-order approximation.

Among all possible perturbations, there is also a special one that is very useful and normally makes the above assumption satisfied. This is the phase perturbation, *i.e.*, the perturbed input $b_p(t) = b_S(t + p(t))$ is only a phase-modulated version of $b_S(t)$ by some signal $p(t)$. For example, it can be used to represent an FM/PM signal in RF applications, or a delayed ramp signal with any slope in digital applications. In particular, when $p(t)$ is a constant, the perturbed solution $x_p(t)$ lies exactly on the same orbit of $x_S(t)$ if initial condition is set properly.

7.3 Preliminaries and Notations (of LPTV systems)

To understand derivations in the following sections, we need to introduce a few notations and lemmas. For simplicity, we consider the system defined by a set of ordinary differential equations

$$\frac{d}{dt}x(t) = f(x(t)) + b(t). \tag{7.6}$$

Following [130], the results can be extended to differential algebraic equations.

We assume that the input $b(t)$ is a periodic signal $b_S(t)$ with period T, and that under this input, the asymptotic response of the system is $x_S(t)$ which is also periodic with period T.

A traditional perturbation analysis using linearization can then be carried out: assuming that the response to the perturbed input $b_p(t) = b_S(t) + b_w(t)$ is $x_p(t) = x_S(t) + w(t)$, and substituting $x_p(t)$ and $b_p(t)$ in (7.6), we obtain

$$\frac{d}{dt}\left(x_S(t) + w(t)\right) = f\left(x_S(t) + w(t)\right) + \left(b_S(t) + b_w(t)\right). \tag{7.7}$$

To the first order approximation, we have

$$\frac{d}{dt}w(t) = G(t)w(t) + b_w(t), \tag{7.8}$$

where $G(t) = \frac{\partial f}{\partial x}\big|_{x_S(t)}$ is a time-varying matrix with period T.

(7.8) is an LPTV system, whose solution, according to Floquet theory [131], is

$$\begin{aligned} w(t) =& \Phi(t,0)w_0 + \int_0^t \Phi(t,s)b_w(s)ds \\ =& U(t)D(t)V^T(0)w_0 + U(t)\int_0^t D(t-s)V^T(s)b_w(s)ds \\ =& \sum_{i=1}^n u_i(t)e^{\mu_i t}v_i^T(0)w_0 + \sum_{i=1}^n u_i(t)\int_0^t e^{\mu_i(t-s)}v_i^T(s)b_w(s)ds \\ =& \sum_{i=1}^n u_i(t)\left(e^{\mu_i t}v_i^T(0)w_0 + \int_0^t e^{\mu_i(t-s)}v_i^T(s)b_w(s)ds\right), \end{aligned} \tag{7.9}$$

where w_0 is the initial condition for (7.8), $\Phi(t,s) = U(t)D(t-s)V^T(s)$ is the *state transition matrix* of (7.8), μ_is are the *Floquet exponents*, $D(t) = \mathrm{diag}(e^{\mu_1 t}, \cdots, e^{\mu_n t})$, $U(t) = [u_1(t), \cdots, u_n(t)]$, $V(t) = [v_1(t), \cdots, v_n(t)]$, and $V^T(t)U(t) = I_n$. More theories and proofs about LPTV systems can be found in [131].

We now introduce a lemma showing that $\frac{dx_S(t)}{dt}$, the time derivative of the periodic solution

$x_S(t)$ of (7.6), satisfies an LPTV system.

Lemma 7.3.1 *The time derivative of the periodic solution $x_S(t)$ of (1.2), i.e., $\frac{d}{dt}(x_S(t))$ satisfies*

$$\frac{d}{dt}w(t) = G(t)w(t) + \frac{db_S(t)}{dt}, \tag{7.10}$$

and can be written as

$$\frac{dx_S}{dt}(t) = U(t)c(t) = \sum_{i=1}^{N} u_i(t)c_i(t), \tag{7.11}$$

where

$$c_i(t) = \lim_{r=t+kT,k\to\infty} \left(e^{\mu_i r} v_i^T(0)\frac{dx_S}{dt}(0) + \int_0^r e^{\mu_i(r-s)} v_i^T(s) \left(\frac{d}{ds}b_S(s)\right) ds \right). \tag{7.12}$$

Proof Since $x_S(t)$ satisfies (7.6), we have

$$\frac{d}{dt}x_S(t) = f(x_S(t)) + b_S(t). \tag{7.13}$$

Take the time derivative of (7.13) on both sides, we obtain

$$\frac{d}{dt}\left(\frac{dx_S(t)}{dt}\right) = \frac{d}{dt}\left(f(x_S(t)) + b_S(t)\right) = \left.\frac{\partial f}{\partial x}\right|_{x_S(t)} \frac{dx_S(t)}{dt} + \frac{db_S(t)}{dt}. \tag{7.14}$$

Therefore, $\frac{d}{dt}(x_S(t))$ satisfies (7.10).

Since $\frac{d}{dt}x_S(t)$ is the asymptotic periodic solution to (7.10), according to (7.9), we further have (7.11) with $c(t)$ defined by (7.12). ■

7.4 Derivation of the Phase Macromodel via Nonlinear Perturbation Analysis

With the important assumption that the trajectory of the perturbed system stays close to the trajectory of $x_S(t)$, the key idea is to show that under the perturbed input $b_p(t)$, the perturbed response $x_p(t)$ can be decomposed into the phase response $z(t)$ and the amplitude deviation $y(t)$ in a reasonable way, *i.e.*,

$$x_p(t) = x_S(t + z(t)) + y(t), \tag{7.15}$$

and that by defining the right differential equation for $z(t)$, $y(t)$ is minimized in some sense.

To show this, we start by defining the **phase equation**, *i.e.*, the differential equation for the phase response $z(t)$. We then show that the input $b_p(t)$ can be decomposed into $b_z(t)$ and $b_y(t)$ such that when only $b_z(t)$ is applied to (7.6), the perturbed response is exactly $x_S(t + z(t))$. We then derive the first-order approximation of $y(t)$ by linearizing original differential equations around the phase-shifted solution $x_S(t + z(t))$, and show that certain decomposition of $y(t)$ is minimized.

Definition 7.4.1 (Phase Equation) *We define the phase equation to be*

$$c^T(t+z)c(t+z)\frac{dz}{dt} = c^T(t+z)V^T(t+z)\left[b_p(t) - b_S(t+z)\right], \tag{7.16}$$

where $c(t) = [c_1(t), \cdots, c_N(t)]$ *is defined in (7.12) and* $V(t)$ *is defined in (7.9).*

With the definition of $z(t)$, we present a theorem showing that part of the input $b_p(t)$ contributes only to the phase response.

Theorem 7.4.1 *Given any perturbed input* $\vec{b}_p(t)$*, define*

$$\begin{aligned} b_z(t) =& b_S(t+z) + \frac{c^T(t+z)}{c^T(t+z)c(t+z)}V^T(t+z) \\ & \cdot \left[b_p(t) - b_S(t+z)\right] U(t+z)c(t+z), \end{aligned} \tag{7.17}$$

then $x_S(t + z(t))$ *is the solution to*

$$\frac{d}{dt}\left(x(t)\right) = f(x(t)) + b_z(t). \tag{7.18}$$

Proof Substituting $x_S(t + z(t))$ in (7.18), we have

$$\begin{aligned} \frac{d}{dt}\left[x_S(t+z(t))\right] &= f(x_S(t+z(t))) + b_z(t) \\ &= \frac{dx_S}{dt}(t+z(t))\left(1 + \frac{dz}{dt}\right). \end{aligned} \tag{7.19}$$

Substituting (7.11) in (7.19), and defining $\tau = t + z$, we have

$$\begin{aligned} f(x_S(\tau)) + b_S(\tau) + U(\tau)c(\tau)\frac{dz}{dt} &= f(x_S(\tau)) + b_z(t) \\ U(\tau)c(\tau)\frac{dz}{dt} &= b_z(t) - b_S(\tau). \end{aligned} \tag{7.20}$$

Substituting (7.17) in (7.20), we have

$$U(\tau)c(\tau)\frac{dz}{dt} = \left[\frac{c^T(\tau)}{c^T(\tau)c(\tau)}V^T(\tau)(b_p(t) - b_S(\tau))\right]U(\tau)c(\tau). \tag{7.21}$$

Since $U(\tau)c(\tau)$ is $\frac{dx_S}{dt}(\tau)$ whose elements are not simultaneously zero, we obtain (7.16). Since all above derivations are reversible, we have also shown that $x_S(t + z(t))$ solves (7.18). ■

It remains to show that by decomposing the perturbed response $x_p(t)$ into phase response and amplitude deviation according to (7.16), the amplitude deviation is minimized in some sense. This is proven by the following theorem. The proof also gives another derivation of the phase equation (7.16).

Theorem 7.4.2 *Suppose the perturbed response is $x_p(t) = x_S(t + z(t)) + y(t)$, then to the first-order approximation, $y(t)$ is*

$$y(t) = U(\tau)\int_0^\tau e^{\Lambda(t-s)}r(s)ds, \tag{7.22}$$

where $\tau = t + z$, $\Lambda = diag(\mu_1, \cdots, \mu_n)$ are the Floquet exponents of (7.8), and $y(t)$ is minimized in the sense that $||r(s)||_2$ is minimized.

Proof We first assume that the response to $b_z(t)$ is exactly $x_S(t + z)$, which gives

$$\begin{aligned}\frac{d}{dt}x_S(t + z(t)) =& \frac{dx_S}{d\tau}(\tau)(1 + \frac{dz}{dt}) \\ =& f(x(\tau)) + b_S(\tau) + \frac{dx_S}{d\tau}(\tau)\frac{d}{dt}z(t).\end{aligned} \tag{7.23}$$

Suppose the phase equation is $\frac{dz}{dt} = a(t)$, then

$$b_z(t) = b_S(t + z) + a(t)\frac{dx_S}{dt}(t + z). \tag{7.24}$$

Now substituting $x_p(t) = x_S(t + z) + y(t)$ into (1.2), we have

$$\begin{aligned}&\frac{d}{dt}(x_S(t + z) + y(t)) = f(x_S(t + z) + y(t)) + b_p(t) \\ \simeq& f(x_S(t + z)) + G(t + z)y(t) + b_z(t) + [b_p(t) - b_z(t))]\,.\end{aligned} \tag{7.25}$$

Therefore, we have

$$\frac{d}{dt}y(t) = G(t + z)y(t) + [b_p(t) - b_z(t)]\,. \tag{7.26}$$

With the assumption that $\frac{dz}{dt}$ is small [29], we define $\hat{t} = t + z$, $\hat{y}(\hat{t}) = y(t)$, $\hat{b}_w(\hat{t}) = b_p(t) - b_z(t)$, and therefore

$$\frac{d}{d\hat{t}}\hat{y}(\hat{t}) = G(\hat{t})\hat{y}(\hat{t}) + \hat{b}_w(\hat{t}). \tag{7.27}$$

Therefore, applying (7.9), we have

$$y(t) = \hat{y}(\hat{t}) = U(\hat{t}) \int_0^{\hat{t}} e^{\Lambda(\hat{t}-s)} V^T(s)\hat{b}_w(s)ds, \tag{7.28}$$

Therefore, $r(\hat{t})$ in (7.22) is

$$\begin{aligned} r(\hat{t}) =& V^T(\hat{t})\hat{b}_w(\hat{t}) \\ =& V^T(t+z)\left(b_p(t) - b_S(t+z) - a(t)U(t+z)c(t+z)\right) \\ =& V^T(t+z)\left(b_p(t) - b_S(t+z)\right) - a(t)c(t+z). \end{aligned} \tag{7.29}$$

The minimization of $||r(\hat{t})||_2$ boils down to the problem of minimizing $||Ax - b||_2$ where $A = c(t+z)$, $x = a(t)$ and $b = V^T(t+z)\left[b_p(t) - b_S(t+z)\right]$. The solution to this problem is simply

$$a(t) = \frac{c^T(t+z)}{c^T(t+z)c(t+z)} V^T(t+z)(b_p(t) - b_S(t+z)), \tag{7.30}$$

which again leads to (7.16). ■

7.4.1 Interpretation in the Projection Framework

The model shown in Section 7.4 can be interpreted as a special reduced order model in the projection framework mentioned in Chapter 4, where state variables are forced to lie in an N_r-dimensional manifold defined by $x = v(x_r)$, and the residual is forced to lie in a N_r-dimensional manifold defined by $r_z = w(r_x)$.

To derive the same phase macromodel under the projection framework, we first enforce that the perturbed response $x_p(t)$ only induces a phase shift, *i.e.*, the state variable lie on the same orbit as that of the unperturbed solution. This is equivalent to saying that we project the state space onto the one-dimensional manifold defined by the unperturbed solution $x_S(t)$ which has a natural parameterization by time t. Therefore, the projection function is defined by

$$x(t) = v(z(t)) = x_S(t+z). \tag{7.31}$$

Substituting (7.31) in (7.6), we have

$$\frac{d}{dt}x_S(t+z) = f(x_S(t+z)) + b_p(t). \tag{7.32}$$

Therefore,

$$\begin{aligned}\frac{d}{dt}\left[x_S(t+z)\right] =& \frac{dx_S}{d\tau}(\tau)\frac{d\tau}{dt} = \frac{dx_S}{d\tau}(\tau)\left(1+\frac{dz}{dt}\right)\\ =& \frac{dx_S}{d\tau}(\tau) + \frac{dx_S}{d\tau}(\tau)\frac{dz}{dt}.\end{aligned} \tag{7.33}$$

On the other hand, we have

$$\frac{d}{d\tau}x_S(\tau) = f(x_S(\tau)) + b_S(\tau), \tag{7.34}$$

and therefore, (7.33) can be written as

$$\frac{d}{dt}x_S(t+z) = f(x_S(\tau)) + b_S(\tau) + \frac{d}{d\tau}x_S(\tau)\frac{dz}{dt}. \tag{7.35}$$

Combining (7.32) and (7.35), we obtain

$$\frac{d}{d\tau}x_S(\tau)\frac{dz}{dt} = b_p(t) - b_S(\tau), \tag{7.36}$$

or equivalently,

$$U(\tau)c(\tau)\frac{dz}{dt} = b_p(t) - b_S(\tau). \tag{7.37}$$

Therefore, after the projection onto the periodic orbit, we obtain N equations (7.37) and 1 state variable z. To derive a reduced order model, we further project the residual, *i.e.*, (7.37), to a one-dimensional manifold. In our model, we have chosen the projection to be

$$w(r_x) = c^T(\tau)V^T(\tau)r_x, \tag{7.38}$$

which is shown to minimize the amplitude deviation in the sense of Theorem 7.4.2.

7.4.2 Connections to PPV Macromodel

The PPV phase macromodel [29] is specifically devised to characterize the phase response of an oscillatory system. Its limitation is that it is only applicable to oscillators. This excludes it from wide applications such as traditional timing analysis and excitory firing neuron network simulations.

Following previous derivations, the PPV macromodel is indeed a special case of our general-purpose phase macromodel. We can obtain the PPV model for oscillators by choosing $c(t)$ in (7.16) to be $[1, 0, \cdots, 0]^T$.

This choice of $c(t)$ in fact is the definition in (7.12) for autonomous oscillators. To see that, we note that for autonomous oscillators, the following two conditions are satisfied: (1)

one Floquet exponent of (7.10) to be 0 and others negative (*e.g.*, $\mu_1 = 0$, $\mu_i < 0$, $2 \leq i \leq n$), and (2) $b_S(t)$ is constant. Therefore, (7.12) becomes

$$\begin{aligned} c_1(t) &= \lim_{t\to\infty} \vec{v}_i^T(0)\frac{d\vec{x}_S}{dt}(0) = 1, \\ c_i(t) &= \lim_{t\to\infty} e^{\mu_i t}\vec{v}_i^T(0)\frac{d\vec{x}_S}{dt}(0) = 0, \quad 2 \leq i \leq n. \end{aligned} \tag{7.39}$$

which leads to $c(t) = [1, 0, \cdots, 0]^T$, and the PPV model.

7.4.3 Generalization for Non-Periodic Trajectories

Previous derivations are based on the assumption that $x_S(t)$ is the asymptotic periodic orbit in the state space. This is not generally the case for practical systems such as digital circuits. However, if we force the input to be a periodic signal (for example, a square wave), then the system will converge to a periodic steady state, and transient behaviors (for example, the rising and falling edge of waveforms) are well-approximated by its periodic steady state. Hence, we can force the input to be periodic as long as the periodic trajectory well-approximates the non-periodic ones. As a result, the derived phase model will also characterize the transient phase responses.

7.5 Algorithm

Based on the above analysis, the algorithm to generate a general-purpose phase macromodel is shown as follows.

Algorithm 9 General-Purpose Phase Macromodeling

1: Given input $b_S(t)$, compute the periodic steady state $x_S(t)$.
2: Compute $\frac{dx_S(t)}{dt}$, the time derivative of $x_S(t)$.
3: Perform Floquet decomposition of the LPTV system
$\frac{dw}{dt} = G(x_S(t))w(t)$ (compute $U(t)$, $V(t)$, and $D(t)$).
4: Compute $c(t) = V^T(t)\frac{dx_S(t)}{dt}$.
5: Precompute $c^T(t)c(t)$ and $c^T(t)V^T(t)$ in (7.16) and store them in the model.

7.6 Numerical Methods for Computing the Phase Model

As shown in Algorithm 9, to compute the phase macromodel, we need to compute the periodic steady state $x_S(t)$, its time derivative $\frac{dx_S(t)}{dt}$, and perform a full Floquet decomposition of an LPTV system.

PSS analysis has been widely adopted in commercial tools and usually uses shooting method [132], FDTD method [132] or harmonic balance [132]. The time derivative of the PSS solution can be computed either in the time domain directly by finite difference approximation or in the frequency domain and be transformed back to time domain.

The main numerical challenge lies in the Floquet decomposition, which is numerically much more ill-conditioned. Implementations according to textbook definitions using Monodromy matrix fail without question. In this section, we first demonstrate that methods based on explicitly computing the Monodromy matrix are numerically extremely bad. We then present[3] two much better-behaved methods to solve this challenge, one based on harmonic balance, and the other based on finite-difference time-domain (FDTD) method.

7.6.1 Floquet Decomposition using Monodromy Matrix

We consider the Floquet decomposition of the LPTV system

$$\frac{d}{dt}x(t) + G(t)x(t) = 0. \tag{7.40}$$

Floquet decomposition [131] refers to computing matrices $U(t)$, $V(t)$ and $D = e^{\Lambda t}$ (where Λ is a diagonal matrix of all Floquet exponents) such that the state transition matrix of (7.40) is

$$\Phi(t, s) = U(t)D(t - s)V^T(s). \tag{7.41}$$

The definition for Floquet exponents are derived from the eigenvalues of the **monodromy matrix** defined by

$$B = \Phi(T, 0) = Pe^{\Lambda T}P^{-1}, \tag{7.42}$$

and $U(t)$ is then defined as $\Phi(t, 0)Pe^{-\Lambda t}$, and $V(t) = U^{-1}(t)$.

This definition naturally gives a straightforward implementation of Floquet decomposition:

Algorithm 10 Floquet Decomposition using Monodromy Matrix

1: Starting with initial condition $X(0) = I$, integrate (7.40) to compute $\Phi(t, 0)$ for $t \in [0, T]$.
2: Compute the monodromy matrix $B = \Phi(T, 0)$.
3: Eigen-decompose the monodromy matrix $B = Pe^{\Lambda T}P^{-1}$.
4: Compute $U(t) = \Phi(t, 0)Pe^{-\Lambda t}$ and $V(t) = U^{-1}(t)$.

This algorithm will work in perfect precision. However, it almost never works in practice. For real systems, most Floquet exponents are normally negative. This means that using any initial condition P, the solution to (7.40) (which is $\Phi(t, 0)P$) tends to 0 exponentially. This

[3]While these methods seem to have been developed before, we present our derivation and also explain the details in the derivation.

makes the computed monodromy matrix B extremely numerically singular, and the eigen-decomposition of B provides useless results. Also, computing $V(t)$ using the inverse of $U(t)$ is not a good choice since the smoothness of $V(t)$ can easily be destroyed.

7.6.2 Equations in terms of $U(t)$ and $V(t)$

As the first and the most important step to tackle the problems in the Monodromy matrix method, we derive equations whose unknowns are $U(t)$ and $V(t)$, respectively. Because $U(t)$ and $V(t)$ are periodic, the resulting differential equations have solutions that are periodic, instead of decaying to 0. Therefore, this avoids numerical errors in the Monodromy matrix computation. Moreover, having equations for $U(t)$ and $V(t)$ separately avoids computing $V(t) = U^{-1}(t)$.

To derive these equations, we start with a fundamental matrix

$$X(t) = \Phi(t, 0) = U(t)D(t)V^T(0), \tag{7.43}$$

which solves (7.40). Substituting (7.43) in (7.40), we have

$$\begin{aligned} & \frac{d}{dt}\left[U(t)D(t)V^T(0)\right] + G(t)U(t)D(t)V^T(0) = 0 \\ \Leftrightarrow \quad & \frac{d}{dt}\left[U(t)D(t)\right] + G(t)U(t)D(t) = 0 \\ \Leftrightarrow \quad & \left[\frac{d}{dt}U(t)\right]e^{\Lambda t} + U(t)\left[\frac{d}{dt}e^{\Lambda t}\right] + G(t)U(t)e^{\Lambda t} = 0. \end{aligned} \tag{7.44}$$

Therefore,

$$\frac{d}{dt}U(t) + U(t)\Lambda + G(t)U(t) = 0. \tag{7.45}$$

Similarly, to derive the equations for $V(t)$, we apply the same technique to the adjoint system of (7.40)

$$\frac{dy(t)}{dt} - G^T(t)y(t) = 0, \tag{7.46}$$

which has a fundamental matrix

$$Y(t) = V(t)D(-t)U^T(0). \tag{7.47}$$

Substituting (7.47) in (7.46), we have

$$\begin{aligned} & \frac{d}{dt}(V(t)D(-t)) - G^T(t)V(t)D(-t) = 0 \\ \Leftrightarrow & \left(\frac{d}{dt}V(t)e^{-\Lambda t} + V(t)(-\Lambda)e^{-\Lambda(t)}\right) - G^T(t)V(t)e^{-\Lambda(t)} = 0. \end{aligned} \tag{7.48}$$

Therefore,

$$\frac{d}{dt}V(t) - V(t)\Lambda - G^T(t)V(t) = 0. \tag{7.49}$$

Therefore, we have shown that $U(t)$ and $V(t)$ satisfy differential equations (7.45) and (7.49), respectively. To solve for $U(t)$ and $V(t)$, we note that they are by definition periodic with period T, and therefore we need to *force* the their periodicity during the computation. Two numerical methods that force the periodicity are HB and FDTD, and we now show how to apply these methods to solve for $U(t)$ and $V(t)$.

7.6.3 Floquet Decomposition via Harmonic Balance

Notations

We follow the notations in [133]: For any periodic matrix or vector $A(t)$ which has a truncated Fourier series $A(t) = \sum_{i=-M}^{M} A_i e^{ji\omega t}$ ($\omega = \frac{2\pi}{T}$), we define the block vector $V_{A(t)}$ of A_is and the block-Toeplitz matrix $T_{A(t)}$ of A_is to be

$$V_{A(t)} = \begin{bmatrix} \vdots \\ A_2 \\ A_1 \\ A_0 \\ A_{-1} \\ A_{-2} \\ \vdots \end{bmatrix}, T_{A(t)} = \begin{bmatrix} & \vdots & \vdots & \vdots & \vdots & \vdots & \\ \cdots & A_0 & A_1 & A_2 & A_3 & A_4 & \cdots \\ \cdots & A_{-1} & A_0 & A_1 & A_2 & A_3 & \cdots \\ \cdots & A_{-2} & A_{-1} & A_0 & A_1 & A_2 & \cdots \\ \cdots & A_{-3} & A_{-2} & A_{-1} & A_0 & A_1 & \cdots \\ \cdots & A_{-4} & A_{-3} & A_{-2} & A_{-1} & A_0 & \cdots \\ & \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}. \tag{7.50}$$

Using this notation, two lemmas follow [133]:

Lemma 7.6.1 *If $Z(t) = X(t)Y(t)$, and $X(t)$, $Y(t)$ are T-periodic, then,*

$$\begin{aligned} V_{Z(t)} &= T_{X(t)}V_{Y(t)}, \quad V_{Z^T(t)} = T_{Y^T(t)}V_{X^T(t)}, \\ T_{Z(t)} &= T_{X(t)}T_{Y(t)}, \quad T_{Z^T(t)} = T_{Y^T(t)}V_{X^T(t)}. \end{aligned} \tag{7.51}$$

Lemma 7.6.2 *If $X(t)$ is T-periodic, then*

$$V_{\dot{X}(t)} = \Omega V_{X(t)}, \quad T_{\dot{X}(t)} = \Omega T_{X(t)} - T_{X(t)}\Omega, \tag{7.52}$$

$$\textit{where} \quad \Omega = \Omega_0 \otimes I_n, \quad \Omega_0 = j\omega\, diag(\cdots, 2, 1, 0, -1, -2, \cdots). \tag{7.53}$$

HB for Floquet decomposition

Theorem 7.6.3 *The eigenvalues and eigenvectors of the HB Jacobian matrix for (7.40) ($J_{HB} = \Omega + T_{G(t)}$) are given by the diagonal of $\Omega_0 \otimes I_n - I_{2M+1} \otimes \Lambda$ and $T_{U(t)}$, respectively.*

The eigenvalues and eigenvectors of the HB Jacobian matrix for (7.46) ($\hat{J}_{HB} = \Omega - T_{G^T(t)}$) are given by the diagonal of $\Omega_0 \otimes I_n + I_{2M+1} \otimes \Lambda$ and $T_{V(t)}$, respectively.

Proof Applying lemmas 7.6.1 and 7.6.2 to to (7.45), we have

$$\begin{aligned} & \Omega T_{U(t)} - T_{U(t)}\Omega + T_{U(t)}T_\Lambda + T_{G(t)}T_{U(t)} = 0 \\ \Leftrightarrow \quad & (\Omega + T_{G(t)})T_{U(t)} + T_{U(t)}(T_\Lambda - \Omega) = 0, \end{aligned} \tag{7.54}$$

which is already in the eigen-decomposed form of $J_{HB} = \Omega + T_{G(t)}$.

Similarly, applying lemmas 7.6.1 and 7.6.2 to to (7.49), we have

$$\begin{aligned} & (\Omega T_{V(t)} - T_{V(t)}\Omega) - T_{V(t)}T_\Lambda - T_{G^T(t)}T_{V(t)} = 0 \\ \Leftrightarrow \quad & (\Omega - T_{G^T(t)})T_{V(t)} - T_{V(t)}(\Omega + T_\Lambda) = 0, \end{aligned} \tag{7.55}$$

which is already in the eigen-decomposed form of $\hat{J}_{HB} = \Omega - T_{G^T(t)}$. ■

Therefore, solving for $U(t)$ and $V(t)$ boils down to the matrix eigenvalue problem of J_{HB} and $\hat{J}_{HB}$. To pick up the right eigenvectors for $U(t)$ and $V(t)$, we note that eigenvalues of two matrices are $\lambda_{i,k} = ji\omega \mp \mu_k, -M \le i \le M, 1 \le k \le n.$ and the block vector $V_{U(t)}$ and $V_{V(t)}$ correspond to eigenvalues $\lambda_{0,k} = \mu_k$. Therefore, assuming μ_ks are real, we can pick up the eigenvectors corresponding to real eigenvalues to obtain $V_{U(t)}$ and $V_{V(t)}$. This leads to the following algorithm:

Algorithm 11 Floquet Decomposition via Harmonic Balance

1: Construct Jacobian matrices J_{HB} and $\hat{J}_{HB}$ in Thm. 7.6.3.
2: Eigen-decompose J_{HB} and $\hat{J}_{HB}$.
3: Select real eigenvalues to be the Floquet exponents μ_ks, and eigenvectors corresponding to μ_ks to be $V_{U(t)}$ and $V_{V(t)}$.
4: Perform inverse FFT to compute $U(t)$ and $V(t)$ in time-domain.

7.6.4 Floquet Decomposition via FDTD Method

Notations

Given any T-periodic matrix or vector $A(t)$, sampled at N time points $t_0, \cdots, t_{N-1}$, we denote block vector $V_{A(t)}$ and the block diagonal matrix $D_{A(t)}$ to be

$$V_{A(t)} = \begin{bmatrix} A(t_0) \\ A(t_1) \\ \vdots \\ A(t_{N-1}) \end{bmatrix}, \quad D_{A(t)} = \begin{bmatrix} A(t_0) & & & \\ & A(t_1) & & \\ & & \ddots & \\ & & & A(t_{N-1}) \end{bmatrix}. \tag{7.56}$$

Using these notations, two lemmas follow [133]:

Lemma 7.6.4 *If $Z(t) = X(t)Y(t)$, and $X(t)$, $Y(t)$ are T-periodic, then,*

$$V_{Z(t)} = D_{X(t)} V_{Y(t)}, \quad D_{Z(t)} = D_{X(t)} D_{Y(t)}. \tag{7.57}$$

Lemma 7.6.5 *If $X(t)$ is T-periodic, then*

$$V_{\dot{X}(t)} = \Omega_{TD} V_{\dot{x}(t)}, \tag{7.58}$$

where Ω_{TD} depends on different finite difference schemes.

$$\Omega_{TD} = H\Omega_{0,TD} \otimes I_n, \quad H = diag(\frac{1}{h_0}, \cdots, \frac{1}{h_{N-1}}). \tag{7.59}$$

For example, $\Omega_{0,TD}$ for backward Euler and central difference schemes are

$$\Omega_{0,BE} = \begin{bmatrix} 1 & & & -1 \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}, \quad \Omega_{0,CE} = \begin{bmatrix} 0 & 1 & & -1 \\ -1 & 0 & 1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & 0 & 1 \end{bmatrix}. \tag{7.60}$$

FDTD for Floquet decomposition

Theorem 7.6.6 *The diagonal of $-\Lambda$ and $V_{U(t)}$ are n eigenvalues and eigenvectors of the FDTD Jacobian matrix for (7.40) ($J_{FDTD} = \Omega_{TD} + T_{G(t)}$), and the diagonal of Λ and $V_{V(t)}$ are n eigenvalues and eigenvectors of the FDTD Jacobian matrix for (7.46) ($\hat{J}_{FDTD} = \Omega_{TD} - T_{G^T(t)}$).*

Proof Applying lemmas 7.6.4 and 7.6.5 to to (7.45), we have

$$(\Omega_{TD} + D_{(G(t)})V_{U(t)} + V_{U(t)}\Lambda = 0, \tag{7.61}$$

which is in partially eigen-decomposed form of $J_{FDTD} = \Omega + T_{G(t)}$.

Similarly, applying lemmas 7.6.4 and 7.6.5 to to (7.49), we have

$$(\Omega_{TD} - D_{G^T(t)})V_{V(t)} - V_{V(t)}\Lambda = 0, \tag{7.62}$$

which is in partially eigen-decomposed form of $J_{FDTD} = \Omega + T_{G(t)}$. ■

Without proof shown here, one can further show that if uniform time step h is used, then all the eigenvalues of J_{FDTD} and $\hat{J}_{FDTD}$ are the eigenvalues of $\frac{1}{h}\Sigma(\Omega_{0,TD}) \otimes I_n - I_N \otimes \Lambda$ and $\frac{1}{h}\Sigma(\Omega_{0,TD}) \otimes I_n + I_N \otimes \Lambda$, respectively, where $\Sigma(\Omega_{0,TD})$ is the diagonal matrix composed of eigenvalues of $\Omega_{0,TD}$. Using this result, one can pick up the Floquet exponents from all the eigenvalues of J_{FDTD} and $\hat{J}_{FDTD}$.

This leads to the following algorithm:

Algorithm 12 Floquet Decomposition via FDTD

1: Construct Jacobian matrices J_{FDTD} and $\hat{J}_{FDTD}$ in Thm. 7.6.6.
2: Eigen-decompose J_{FDTD} and $\hat{J}_{FDTD}$.
3: Select the Floquet exponents μ_ks (according to the finite-difference scheme) and eigenvectors $V_{U(t)}$ and $V_{V(t)}$.

7.6.5 Forcing Bi-orthogonality

Theoretically, matrices $U(t)$ and $V(t)$ from Floquet decomposition satisfies bi-orthonormality $V^T(t)U(t) = I$ which is an important result that is used in our derivation of the phase model.

However, bi-orthonormality may be destroyed in numerical methods, especially in our methods where $U(t)$ and $V(t)$ are solved separately in two boundary-value problems of two LPTV systems. To preserve bi-orthonormality, we need to scale one of them to restore bi-orthonormality or nearly bi-orthonormality.

The workaround we employed in our implementation is: given $U(t) = [u_1(t), \cdots, u_n(t)]$ and $V(t) = [v_1(t), \cdots, v_n(t)]$, we scale $V(t)$ to obtain $\tilde{V}(t) = [\frac{v_1(t)}{v_1^T(t)\vec{u}_1}, \cdots, \frac{v_n(t)}{v_n^T(t)u_n(t)}]$. This simple procedure forces $v_i^T(t)u_i(t) = 1$. One may further scale $V(t)$ to make it bi-orthogonal to $U(t)$, but in our experiments, we observe good bi-orthogonality between $U(t)$ and $V(t)$.

7.7 Optimization-Based Phase Macromodel Computation

The phase equation, in a simplified form, is essentially

$$\frac{dz}{dt} = v(t+z)(b_p(t) - b_S(t+z)). \tag{7.63}$$

The only unknown in (7.63) is the periodic function $v(t)$.

While the HB/FDTD methods in the previous section provides a theoretically rigorous way of computing $v(t)$, they are potentially computationally expensive and numerically problematic for large-scale systems.

In this section, we fit a model in the form of (7.63) from input/output trajectories of the original system.

Consider the simple case where we only have one input/output trajectories $(b_p(t), x_p(t))$. We hope to choose $v(t)$ such that the resulting $z(t)$ minimizes $y(t)$ in

$$x_p(t) = x_S(t + z(t)) + y(t). \tag{7.64}$$

Therefore, the optimization problem is

$$\begin{aligned} \underset{v(t)}{\text{minimize}} \quad & ||x_p(t) - x_S(t+z(t))|| \\ \text{subject to} \quad & \frac{dz}{dt} = v(t+z)(b_p(t) - b_S(t+z)) \end{aligned} \tag{7.65}$$

We may either parameterize $v(t)$ by a piecewise-linear waveform, or by its Fourier coefficients.

7.8 Examples and Experimental Results

In this section, we illustrate applications of NTIM to several nonlinear systems.

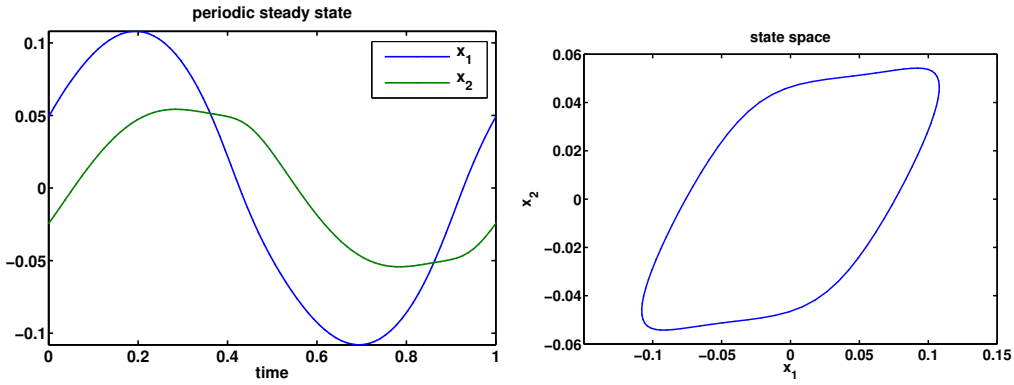
We first look at a simple two-dimensional nonlinear system in great detail – we present results on phase macromodel construction, and show that our numerical methods to compute the phase macromodel (which mainly involves the full Floquet decomposition) have superior accuracy over the straightforward Monodromy matrix method. We then compare the simulation results of the phase model to those of the full model, and analyze the results for different types of perturbations, including perturbations on amplitude, frequency, phase and initial condition (impulse perturbation). Following this case study, we present results on a firing neuron model and an inverter chain circuit, where special types of perturbations that are of interest in practice are discussed.

7.8.1 A Simple Nonlinear System

The first nonlinear system we consider is described by

$$\begin{aligned} \frac{d}{dt}x_1 + x_1 + 1000(x_1 - x_2)^3 - b(t) &= 0 \\ \frac{d}{dt}x_2 + x_2 - 1000(x_1 - x_2)^3 &= 0. \end{aligned} \tag{7.66}$$

The (nominal) periodic input $b_S(t)$ is set to $b_S(t) = \cos(2\pi t)$, and the periodic steady state $x_S(t)$ is solved and shown in Fig. 7.3.

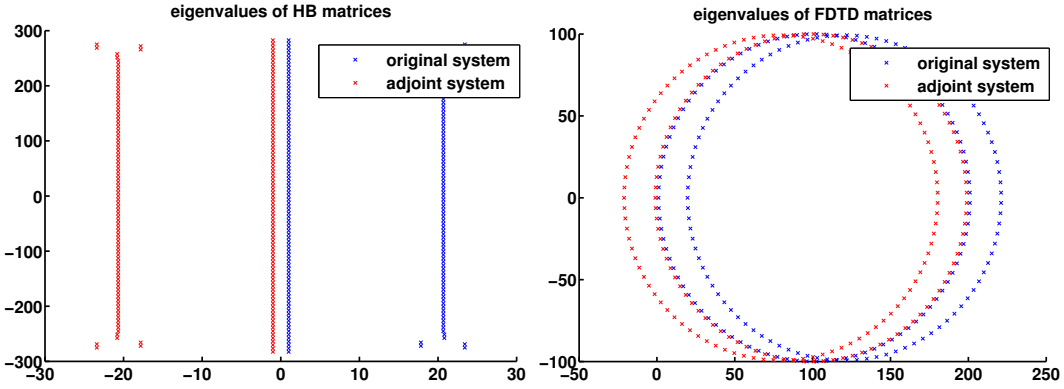


(a) Time domain. (b) State space.

Figure 7.3: Periodic steady state of (7.66).

To derive the phase model described in Section 7.4, we perform a full Floquet decomposition of the LPTV system obtained by linearizing (7.66) around its periodic steady state $x_S(t)$. We apply our methods described in Section 7.6 which boil down to solve eigenvalue problems of HB Jacobian matrices and FDTD Jacobian matrices. The computed eigenvalues of the HB Jacobian matrix (using 45 harmonics) and the FDTD Jacobian matrix (using 100 time points) for the LPTV system and its adjoint system are depicted in Fig. 7.4(a) and Fig. 7.4(b), respectively. From Fig. 7.4(a), we see that eigenvalues are approximately $\pm 1 \pm ji2\pi$ and $\pm 20.68 \pm ji2\pi$ $(i = -45, \cdots, 45)$. These eigenvalues match the theoretical results, although the eigenvalues at the end of the line $x = \pm 20.68$ are off a little bit, Similarly, from Fig. 7.4(b), the eigenvalues are approximately $\pm 1 + 100(1 - e^{ji2\pi/100})$ $(i = 0, \cdots, 99)$ and $\pm 20.68 + 100(1 - e^{ji2\pi/100})$ $(i = 0, \cdots, 99)$. They also match the theoretical results since the eigenvalues of $\Omega_{0,BE}$ are $\sigma_i = 1 - e^{ji2\pi/100}$ for $i = 0, \cdots, 99$.

The eigenvectors of these matrices corresponding to the Floquet exponents -1 and -20.68 are then extracted to construct $U(t)$ and $V(t)$, which are plotted in Fig. 7.5(a) and Fig. 7.5(b), respectively. To examine the bi-orthonormality of the two matrices, we also plot



(a) Eigenvalues of HB Jacobians. (b) Eigenvalues of FDTD Jacobians.

Figure 7.4: Eigenvalues of HB Jacobians and FDTD Jacobians.

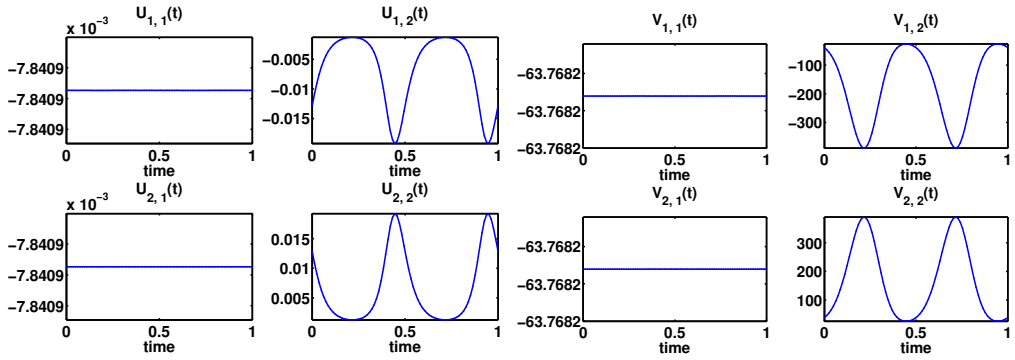
the matrix $V^T(t)U(t)$ in Fig. 7.5(c). From Fig. 7.5(c), we see that $v_1^T(t)u_1(t)$ and $v_2^T(t)u_2(t)$ equal to constant 1, and $v_1^T(t)u_2(t)$ and $v_2^T(t)u_1(t)$ are much less than 10^{-12}. Therefore, the bi-orthonormality is preserved.

We have not shown results using the Monodromy matrix method, since these results are extremely inaccurate. For the fact that this system (as well as almost all practical systems) is stable and Floquet exponents are all negative, the Monodromy matrix is numerically extremely singular, and the eigen-decomposition produces meaningless results.

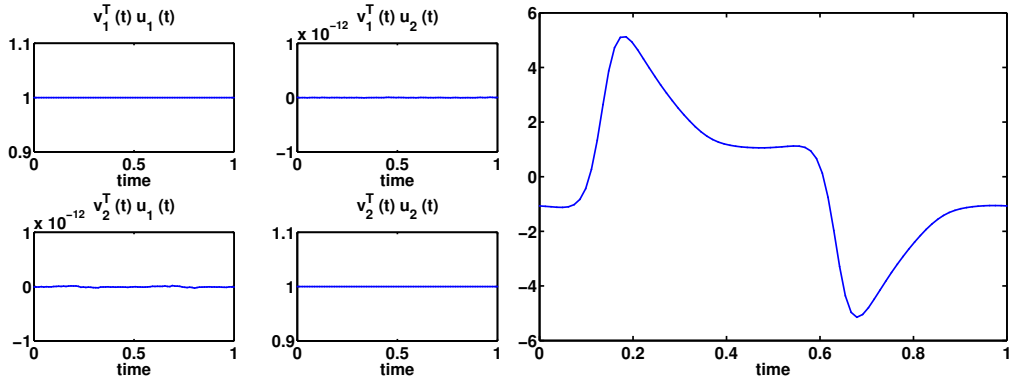
Using results from Floquet decomposition, we construct the phase macromodel, *i.e.*, the phase equation (7.16). The time-varying function $\frac{c^T(t)V^T(t)B}{c^T(t)c(t)}$ (assuming input $b(t) = Bb(t)$) is plotted in Fig. 7.5(d). This key function models the nonlinearities that contribute to the phase response, and can be viewed as the sensitivity of inputs to the phase response. From Fig. 7.5(d), we can infer that inputs applied at time $t \simeq 0.2$ and $t \simeq 0.7$ have the most significant impact on the phase response.

We then examine results of our phase model under different types of input perturbations. Although any perturbation is applicable to the model, we specifically consider four types of perturbations here: for a periodic input $b_S(t)$, we consider phase perturbation $b_S(t+p)$, frequency perturbation $b_S(ft)$, amplitude perturbation $Ab_S(t)$, and impulse perturbation $b_S(t) + D\delta(t)$.

The simulation results under a constant phase perturbation $b_p(t) = b_S(t+0.4)$ with initial condition $z = 0$ are plotted in Fig. 7.6. Fig. 7.6(a) depicts the waveform of $z(t)$, which shows that the phase is converging to $z = -0.6$. This corresponds to the time-domain solution $x_S(t + 0.4)$. Fig. 7.6(b) plots the time-domain waveforms of x_1 and x_2. It is observed that the initial transient behaviors are not matched exactly, but the waveforms stay quite close. To better understand the dynamics, we examine the time-evolution of state variables in the state space, which is depicted in the $x - t$ space in Fig. 7.6(c). It is observed that state



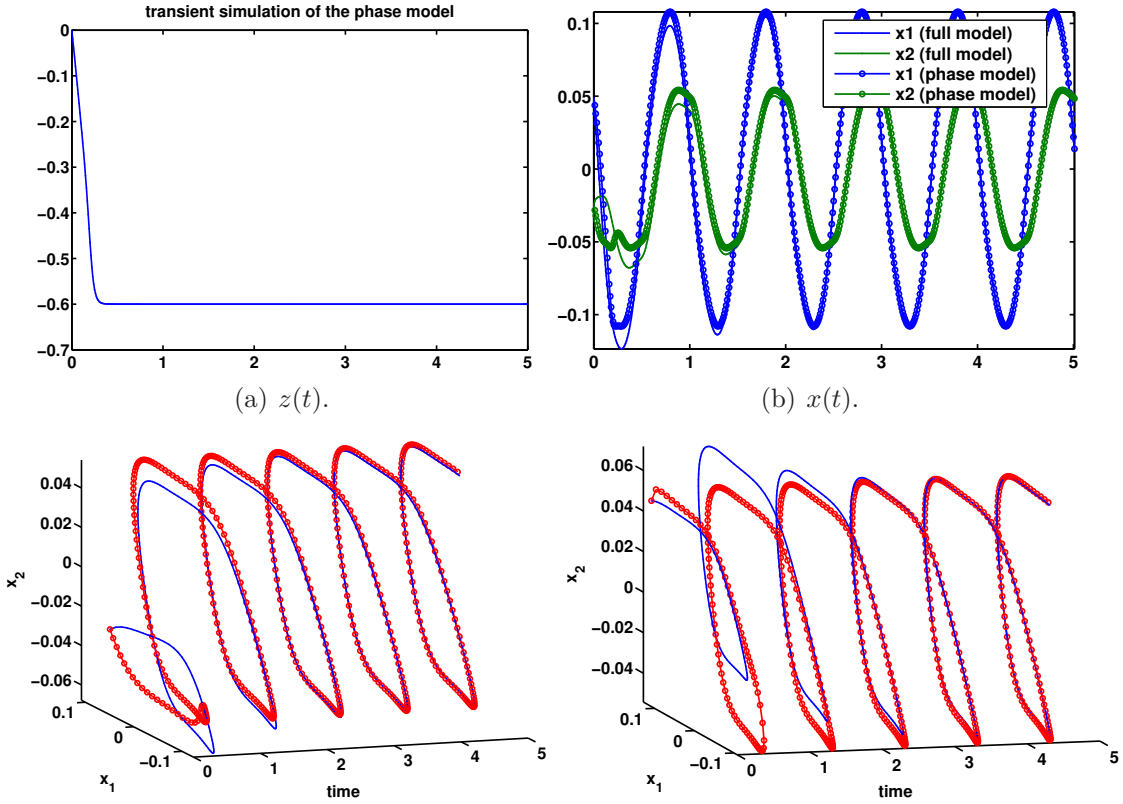
(a) $U(t)$. (b) $V(t)$.



(c) $V^T(t)U(t)$. (d) $\frac{c^T(t)V^T(t)B}{c^T(t)c(t)}$ in (7.16).

Figure 7.5: Floquet decomposition and phase macromodel construction.

variables of the phase model are confined on the orbit of $x_S(t)$, but they try to follow the correct trajectory by choosing almost the closest point on the orbit of $x_S(t)$. The simulation results for another initial condition $z = 0.2$ is also plotted in the $x - t$ space in Fig. 7.6(d), and this is equivalent to applying an impulse function at time $t = 0$. Similar behaviors of the phase model are observed, and it finally converges to the correct asymptotic response. Indeed, if a constant phase-shifted input $b_S(t + p)$ is applied, no matter what the initial condition is, our phase model will reproduce the correct phase-shifted asymptotic output. This can be seen by the fact that $z(t) = p$ is a fixed point of (7.16).



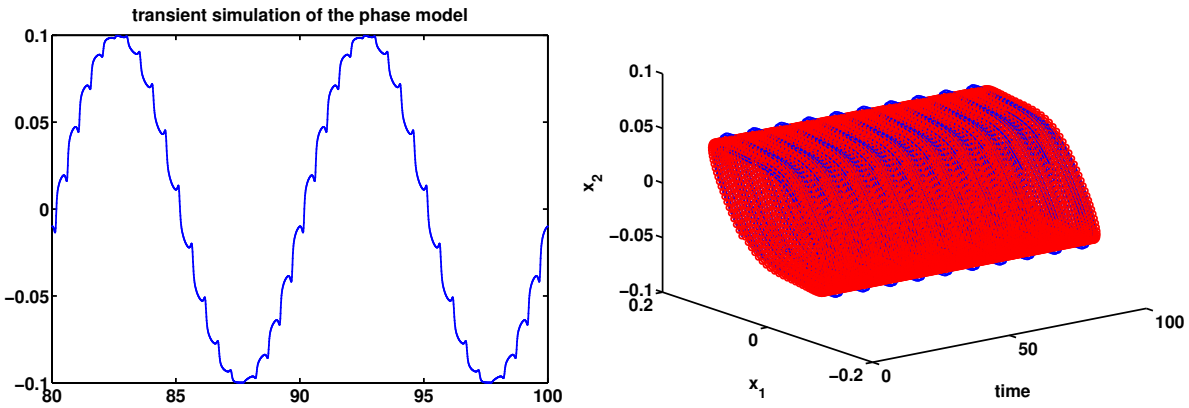
(a) $z(t)$. (b) $x(t)$.

(c) State space with time, $z_0 = 0$. (d) State space with time, $z_0 = 0.2$.

Figure 7.6: Transient simulation of the phase model when $b(t) = \cos(2\pi(t + 0.4))$. In Fig. 7.6(c) and Fig. 7.6(d), red(circled): phase model, Blue(solid): full model.

We then make the phase perturbation time-varying – we apply an PM signal $b_p(t) = \cos(2\pi(t + 0.1\sin(0.2\pi t)))$, and the simulation results for 100 cycles are shown in Fig. 7.7. It is seen in Fig. 7.7(b) that the response of the full model almost lies on the periodic orbit of $x_S(t)$, and therefore, the phase model works perfectly.

We then apply a frequency perturbed input $b_p(t) = b_S(1.2t)$, and the simulation results



(a) $z(t)$. (b) State space with time.

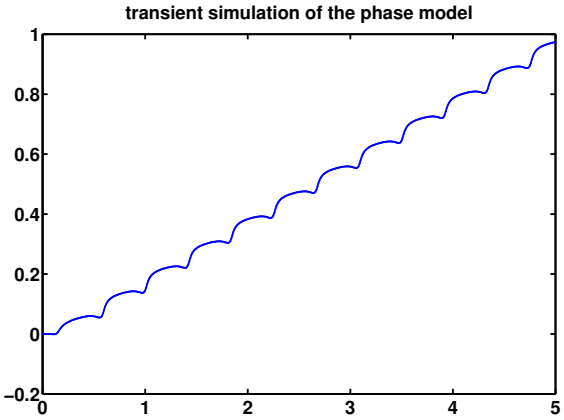
Figure 7.7: Transient simulation when $b_p(t) = \cos(2\pi(t + 0.1\sin(0.2\pi t)))$. In Fig. 7.7(b), red(circled): phase model, Blue(solid): full model.

are shown in Fig. 7.8. It is seen that the periodic orbit has a large deviation from that of $x_S(t)$, and the phase model is doing its best to approximate the right trajectory using points on $x_S(t)$. Most importantly, although the resulting time-domain waveforms do not match exactly, the timing information is captured – the frequency of the output waveform is 1.2 which is the same as that of $b_p(t)$. Also note that the frequency perturbation can be interpreted as a phase perturbation $p(t) = 0.2t$ which can grow unboundedly as time evolves. This shows that the perturbation can be very large as long as the underlying assumption (that the trajectory does not change much) is satisfied.

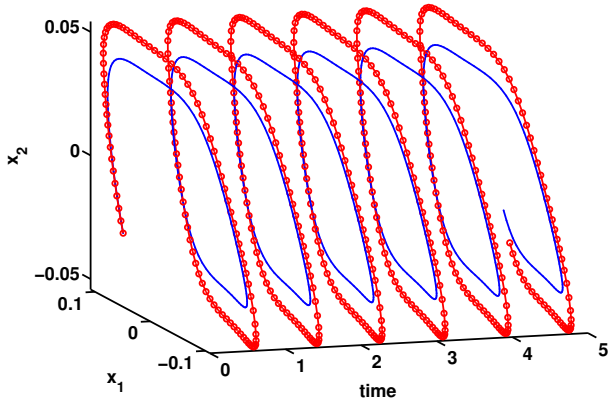
Then an amplitude perturbed signal $b_p(t) = 1.2b_S(t)$ is applied, and the simulation results are shown in Fig. 7.9. Similar to previous results, the periodic orbit deviates from that of $x_S(t)$, and the phase model produces a reasonably well-approximated waveforms. Note that in many applications such as digital circuits and neuron models, voltage/potential waveforms reach a saturated value in the nominal periodic solution $x_S(t)$. This fact makes the periodic orbit insensitive to the amplitude perturbation, and therefore the assumption that the trajectory stays close to $x_S(t)$ is satisfied. Therefore, the phase model generates good results in these cases, as we will show in next two examples.

We have not provided results of LTI and LPTV reduced models since they simply produce meaningless results. The assumption that the additive perturbation input must be "small" is generally not satisfied. Specifically, in the case of the phase and frequency perturbations, suppose the magnitude of $b_S(T)$ is A, then the corresponding additive perturbation $\Delta b(t) = b_S(t + p(t)) - b_S(t)$ can have a magnitude of $2A$, and this usually breaks the small signal assumption.

The speedups are not easy to measure considering various factors including memory allocation and MATLAB optimization. For the examples we show in this paper, the measured

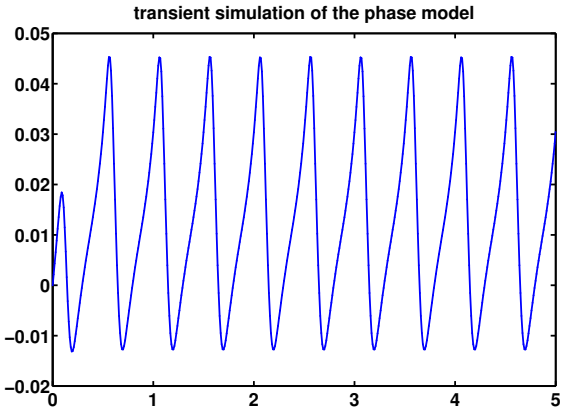


(a) $z(t)$.

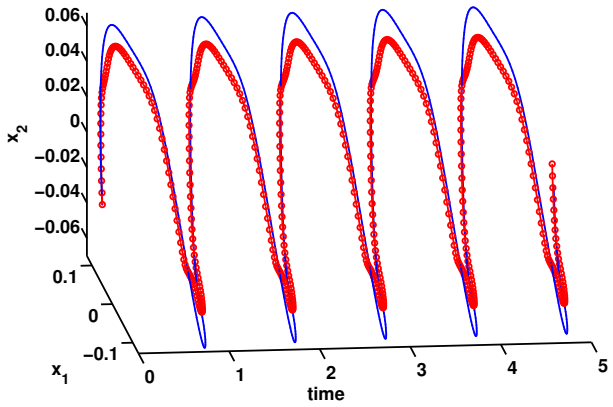


(b) State space with time.

Figure 7.8: Transient simulation of the phase model when $b_p(t) = \cos(2\pi(1.2t))$. In Fig. 7.8(b), red(circled): phase model, Blue(solid): full model.



(a) $z(t)$.



(b) State space with time.

Figure 7.9: Transient simulation of the phase model when $b_p(t) = 1.2\cos(2\pi t)$. In Fig. 7.9(b), red(circled): phase model, Blue(solid): full model.

transient runtime speedup is generally about 10× to 20×. However, since the size of the phase model is 1, the speedups are expected to be much larger for larger systems, similar to PPV models [29].

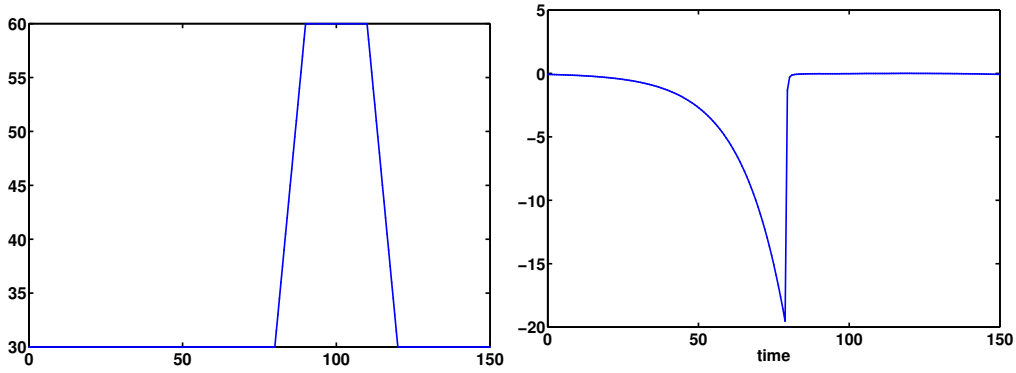
7.8.2 A Firing Neuron Model

The firing neuron model we consider is known as Morris-Lecar model [134]. The differential equations for a single neuron are

$$\begin{aligned}\frac{d}{dt}V =& \frac{1}{C_M}\left(-g_L(V-V_L) - g_{Ca}M_\infty(V-V_{Ca}) - g_K N(V-V_K) + I\right)\\ \frac{d}{dt}N =& (N_\infty - N)\frac{1}{\tau_N},\end{aligned} \tag{7.67}$$

where $M_\infty = 0.5(1+\tanh((V-V_1)/V_2), N_\infty = 0.5(1+\tanh((V-V_3)/V_4), \tau_N = 1/(\Phi\cosh((V-V_3)/(2V_4)))$. The input is the injection current I, and other parameters are chosen as $C_M = 20, g_K = 8, g_L = 2, V_{Ca} = 120, V_K = -80, V_L = -60, V_1 = -1.2, V_2 = 18, V_4 = 17.4, g_{Ca} = 4, \Phi = 1/15, V_3 = 12$, which are adapted from [134].

Using this set of parameters, the neuron is not self-oscillatory, and therefore the PPV model for oscillators is not applicable. However, the neuron fires when adequate currents are injected. We set the input current to be a pulse signal shown in Fig. 7.10(a). The phase macromodel of this neuron is computed, and the time-varying function $\frac{c^T(t)V^T(t)B}{c^T(t)c(t)}$ is plotted in Fig. 7.11(a).



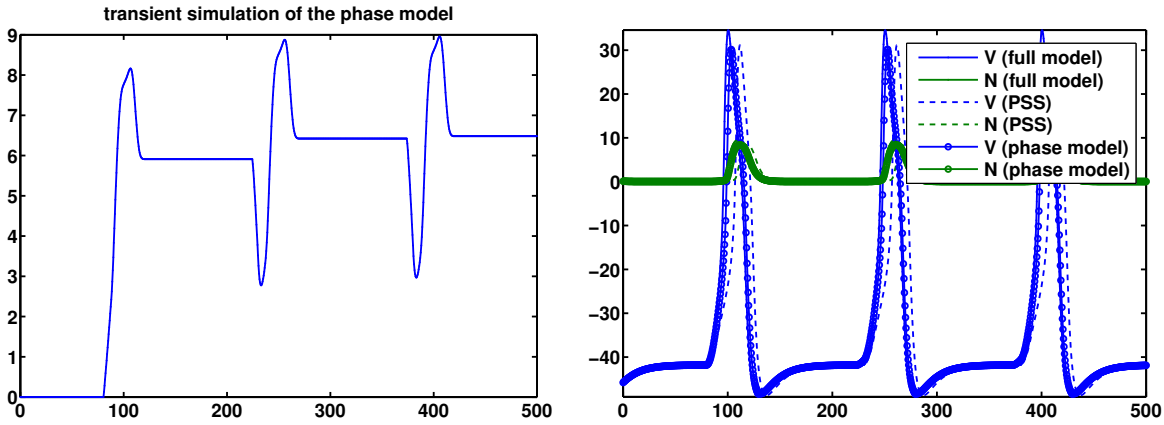
(a) Input pulse.

(b) $\frac{c^T(t)V^T(t)B}{c^T(t)c(t)}$ in (7.16).

Figure 7.10: Input signal and the phase model.

We then apply a pulse input whose magnitude is twice of the original one, and the simulation results are plotted in Fig. 7.11. It is seen that although the input amplitude is

doubled, the periodic orbit only deviates a little bit, and the phase model results almost match those of the full model.



(a) $z(t)$. (b) $V(t)$ and $N(t)$.

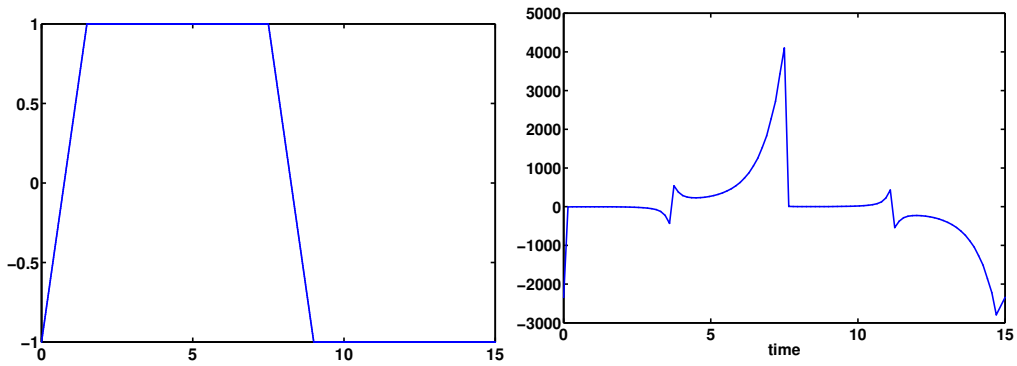
Figure 7.11: Transient simulation under amplitude perturbation.

7.8.3 An Inverter Chain Circuit

We apply the phase macromodeling technique to a nine-stage inverter chain circuit. The nominal periodic input is set to a square wave shown in Fig. 7.12(a), and the time-varying function $\frac{c^T(t)V^T(t)B}{c^T(t)c(t)}$ is plotted in Fig. 7.12(b). This circuit is a typical "digital circuit" in the sense that each node voltage settles to a saturated value in a short time, and this makes the trajectory in the state space less sensitive to input perturbations. However, the timing properties are sensitive to input perturbations.

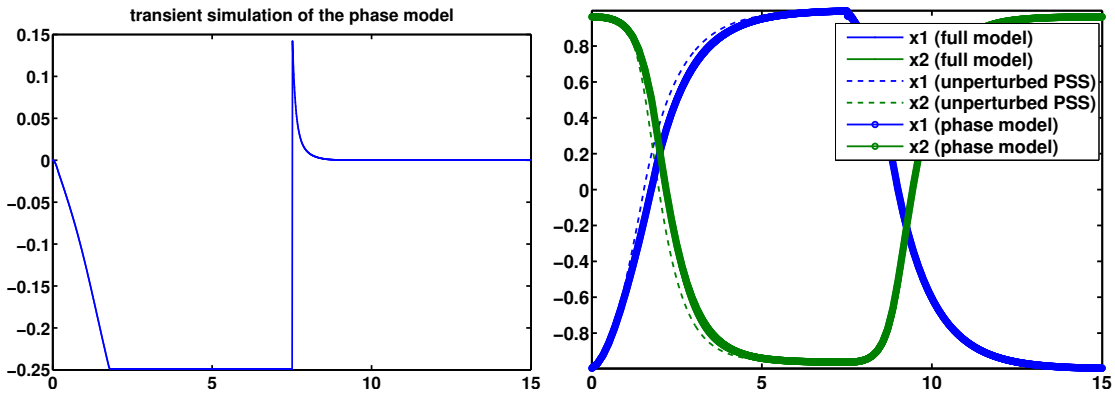
We have tried several kinds of perturbations and similar results to those of previous examples are observed. Here, we show a special perturbation that is of interest in timing analysis – the perturbation is on the slope of the input ramp. In timing analysis and library generation, people normally compute a look-up table storing a map from input slope to output delay and slope. We show that our phase model reproduces accurate timing properties.

In Fig. 7.13, we plot the transient simulation results of the phase model and the full model when the slope of the rising edge is 0.6 (instead of 0.5 for the nominal input). It is observed that delays of waveforms change due to perturbation, and the phase model matches that of the full model.



(a) Input signal. (b) $\frac{c^T(t)V^T(t)B}{c^T(t)c(t)}$ in (7.16).

Figure 7.12: Input signal and the phase model.



(a) $z(t)$. (b) $\vec{x}(t)$.

Figure 7.13: Transient simulation under slope perturbation.

Chapter 8

DAE2FSM: Finite State Machine Abstraction from Differential Equations

In this chapter, we study a different problem in which a finite state machine, instead of a smaller set of differential equations, is abstracted from original differential equations. The problem is motivated by the fact that in modern digital designs, circuit behaviors may not be accurately modeled by the ideal pre-specified finite state machines, due to parameter variations, crosstalk and coupling, environment changes, *etc.*. As a result, a ground-up abstraction of the finite state machine from differential equations, deemed as the golden model, is needed. Other than the circuit applications, the model is also found useful in other engineering problems, such as biological systems.

We start with a detailed comparison of DAE/ODE models and FSM models that shed light on the validity of such abstractions and the approach we may follow. Then we review several existing approaches in the community of machine learning[1], and discuss how we adapt these algorithms in learning differential equations. The key challenge is to develop corresponding concepts/subroutines in those of machine learning, such as state merging and equivalence checking. We present experimental results of several systems, and show applications of the finite state machines learned in further system-level simulation and analysis.

8.1 Problem Formulation

The problem of FSM abstraction is one variation of the problem (2.1) in Section 2.1, where $\mathcal{M}$ is the set of ODEs/DAEs, and $\mathcal{M}_r$ is the set of finite state machines. The fundamental questions are similar, *i.e.*,

[1]The term *machine learning* here refers to the actual finite state machine learning, rather than statistical learning such as learning Bayesian networks.

1. What are $F(\cdot,\cdot)$, $G(\cdot,\cdot)$ and $H(\cdot,\cdot)$ that lead to a reasonable FSM model?

2. Can we, or how do we efficiently solve the optimization problem?

To answer these questions, we compare DAE models and FSM models to identify challenges and clarify ambiguities.

8.1.1 Differential Equations and Finite State Machines

DAEs and FSMs are defined in a similar way – they both model the dynamics of states under input excitations and produce outputs as a function of time. The major difference is that the DAE model is a continuous model while the FSM model is a discrete model.

Take Mealy FSMs as an example. A Mealy machine is defined by a 6-tuple $(Q, Q_0, \Sigma, \Lambda, \delta, G)$, where $Q = \{q_1, \cdots, q_{|Q|}\}$ is a finite set of states, $Q_0 \in Q$ is the initial state, $\Sigma = \{\sigma_1, \cdots, \sigma_{|\Sigma|}\}$ is a finite set of input alphabet, $\Lambda = \{\lambda_1, \cdots, \lambda_{|\Lambda|}\}$ is a finite set of output alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function that maps a pair of a state and an input symbol to the next state, and $G : Q \times \Sigma \to \Lambda$ is the output function that maps a pair of a state and an input symbol to an output symbol. The evolution of the state in an FSM is determined by the transition function, and is a discrete-time discrete-value trace $s_i, i = 0, 1, 2, \cdots$, where $s_i \in Q$. Similarly, the input-output pair of an FSM is a pair of discrete-time discrete-value traces $(u_i, y_i), i = 0, 1, 2, \cdots$, where $u_i \in \Sigma$ and $y_i \in \Lambda$.

In contrast, in differential equations, the evolution of state variables $x \in \mathbb{R}^N$ is determined by the underlying differential equation, or equivalently, a state transition equation

$$x(t) = \phi(x_0, u(t)), \tag{8.1}$$

which maps a pair of current state x_0 and input signals $u(t)$ to states in the future $x(t)$. The evolution $x(t)$, $t \in \mathbb{R}$ is therefore a continuous-time continuous-value trajectory. Similarly, the input-output pair of the DAEs is a pair of continuous-time continuous-value trajectories $(u(t), y(t))$, $t \in \mathbb{R}$.

Comparison of different components in DAE models and FSM models are summarized in Table 8.1.

Table 8.1: Comparison of differential equations and finite state machines

Model	Differential Equations	Finite State Machines
Time	continuous, $t \in \mathbb{R}$	discrete, $t \in \mathbb{Z}$
State Space	$x \in \mathbb{R}^N$ (infinite)	$s \in Q = \{q_1, \cdots, q_{\|Q\|}\}$ (finite)
Input	$u \in \mathbb{R}$ (infinite)	$u \in \Sigma = \{\sigma_1, \cdots, \sigma_{\|\Sigma\|}\}$ (finite)
Output	$y \in \mathbb{R}$ (infinite)	$y \in \Lambda = \{\lambda_1, \cdots, \lambda_{\|\Lambda\|}\}$ (finite)
Update Equation	$x(t) = \phi(x_0, u)$, $\phi : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N$	$s_{i+1} = \delta(s_i, u_i)$, $\delta : Q \times \Sigma \to Q$
Output Equation	$y = h(x, u)$, $h : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}$	$y_i = G(s_i, u_i)$, $G : Q \times \Sigma \to \Lambda$

8.1.2 Equivalence between DAE Models and FSM Models

The difficulty of formulating the FSM abstraction problem lies in the inherent differences between DAE models and FSM models: DAE models are continuous and the state space is infinite, while FSM models are discrete and the state space is finite. Then, does it make sense for an FSM model to be equivalent to a DAE model? If not, what does it mean for an FSM model to approximate a DAE model?

For the first question, the answer is at least partially yes. In digital circuit design, we essentially build up circuits that realize a specific FSM (*e.g.*, a latch). With the valid consensus that circuits are well modeled by differential equations, it makes sense to say that these DAEs are equivalent to the corresponding FSMs.

Example 8.1.1 (An RC circuit) *Consider an RC circuit whose state variable x is governed by the differential equation*

$$\frac{d}{dt}x = -x + u, \tag{8.2}$$

where u is the binary input being 0 or 1. The output $y = x$ is sampled at time $t = n$ where $n \in \mathbb{Z}$, and is deemed as 0 if $x(n) > 0.5$, and 1 otherwise.

Let the initial condition be x_0. If input is 0, the output at the next sample time is

$$x(1) = x_0 e^{-1}. \tag{8.3}$$

If input is 1, the output at the next sample time is

$$x(1) = (x_0 - 1)e^{-1} - 1. \tag{8.4}$$

From (8.3) and (8.4), we see that $x(t)$ cannot overshoot, and therefore $0 \leq x(t) \leq 1$.

In order for the circuit to emulate a latch circuit, i.e., the next output always equals to the current input, we compute the worst-case scenarios. Assuming the current input is 0, then the next output is

$$x(1) = x_0 e^{-1} \leq e^{-1} \simeq 0.38 < 0.5. \tag{8.5}$$

Therefore, the next output is always 0 if the current input is 0. Similarly, we can show that the next output is always 1 if the current input is 1.

In summary, (8.2) realizes the FSM shown in Fig. 8.1, which is the FSM of an ideal latch.

From Example 8.1.1, we have the following notion of equivalence between a DAE model and an FSM model:

Definition 8.1.1 (Equivalence between a DAE model and an FSM model) *Suppose we have a mapping $\alpha : \{u_i\} \mapsto u(t)$ from discrete-time discrete-value input sequences (of the FSM model) to continuous-time continuous-value input waveforms (of the DAE model), and*

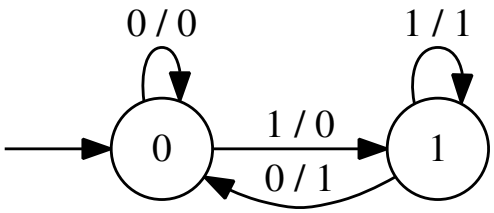


Figure 8.1: The FSM of an ideal latch.

a mapping $\beta : y(t) \mapsto \{\hat{y}_i\}$ *from continuous-time continuous-value output waveforms (of the DAE model) to discrete-time discrete-value output sequences (of the FSM model.*

If for all input sequences $\{u_i\}, i = 0, 1, 2, \cdots$*, the discretized output of the DAE model matches the output of the FSM model,* $\hat{y}_i = y_i, i = 0, 1, 2, \cdots$*, then the DAE model and the FSM model are called to be equivalent with each other.*

This definition is illustrated in Fig. 8.2.

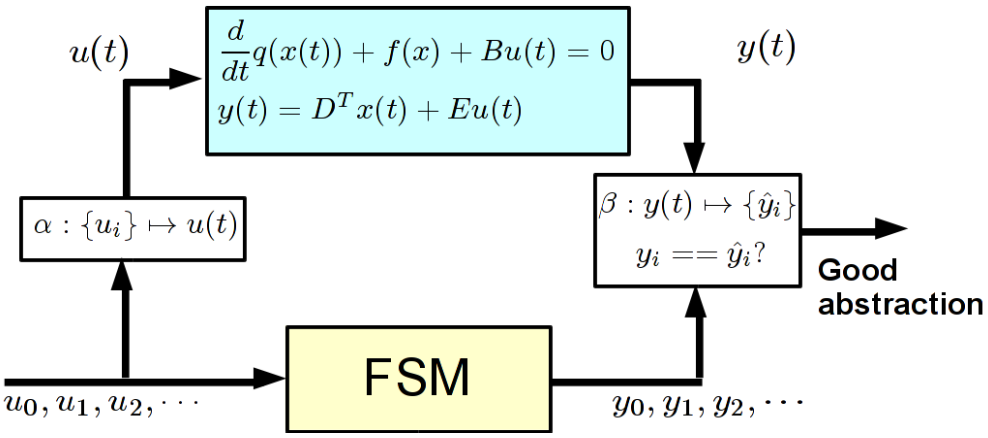


Figure 8.2: Equivalence between a DAE model and an FSM model.

This definition of equivalence involves two mappings α and β that are application dependent. In Example 8.1.1, the mapping α maps a binary sequence to a binary waveform, and the mapping β samples and discretized a continuous waveform to a binary sequence.

This definition also involves conditions of responses to all input sequences. However, it is usually a hard property to prove for general DAEs except for simple cases. Therefore, we also define a relaxed equivalence relationship which involves responses to specified input sequences.

Definition 8.1.2 ($\mathcal{U}$-Equivalence between a DAE model and an FSM model) *Suppose we have a mapping $\alpha : \{u_i\} \mapsto u(t)$ from discrete-time discrete-value input sequences (of the FSM model) to continuous-time continuous-value input waveforms (of the DAE model), and a mapping $\beta : y(t) \mapsto \{\hat{y}_i\}$ from continuous-time continuous-value output waveforms (of the DAE model) to discrete-time discrete-value output sequences (of the FSM model.*

If for all input sequences $\{u_i\} \in \mathcal{U}$ which is a subset of input sequences, the discretized output of the DAE model matches the output of the FSM model, $\hat{y}_i = y_i, i = 0, 1, 2, \cdots$, then the DAE model and the FSM model are called to be $\mathcal{U}$-equivalent with each other.

For example, $\mathcal{U}$ can be the set of input sequences of fixed length L. This is indeed reasonable in many practical applications because system state evolves from a steady state for some time and gets reset once in a while. For example, in a communication system or a high-speed link, the circuits are reset after each packet is transmitted/received. Therefore, the models will be good enough if they matches the responses to input sequences of length equal to the packet size.

This relaxed definition therefore specifies an FSM approximation to the original DAE model. By specifying different $\mathcal{U}$'s in different applications, we obtain different levels of approximations.

Another notion of equivalence may be defined by discretizing the continuous state space of the DAE model and adding transitions among different regions. Denote $X(s_i)$ to be the region in $\mathbb{R}^N$ corresponding to the state s_i. The transition from one state s_i to another state s_j is added if there exists a point $x_i \in X(s_i)$ and an input $u_i \in \Sigma$ such that $\phi(x_i, \alpha(u_i)) \in X(s_j)$. For example, in Fig. 8.2, the state 0 corresponds to the half space $x < 0.5$ and the state 1 corresponds to the half space $x \geq 0.5$. and the transitions are added by the rule above.

While this conversion from a DAE model to an FSM model seems to be straightforward for the above simple example, it is not generally the case. Consider a similar example as Example 8.1.1, where the dynamics of x is defined by

$$\frac{d}{dt}x = -0.5x + u. \tag{8.6}$$

In this case, if we still use the same partitioning, then we may have an FSM with more than 4 transitions, leading to a non-deterministic FSM. Also, in general, it may not be possible to examine every point in a region, and some transitions may be lost. While a non-deterministic FSM may be useful, it may be useless if there are too many transitions that little information of the DAE behaviors is modeled.

8.1.3 FSM Abstraction Formulation

Based on the above definition, we can formulate a simplistic FSM abstraction problem in the form of (2.1):

$$\begin{array}{ll} \underset{\text{FSMs}}{\text{minimize}} & 1 \\ \text{subject to} & \text{The FSM model is } \mathcal{U}\text{-equivalent to the DAE model} \end{array} \tag{8.7}$$

where we set $F(\cdot,\cdot)$ to be constant 1, $H(\cdot,\cdot)$ to be empty, and $G(\cdot,\cdot)$ is the difference between the discretized outputs of the DAE model and the outputs of the FSM model.

In another word, in problem (8.7), we are given a set of input-output trajectories of a DAE model, and we try to find an FSM model that matches these input-output trajectories. It appears that there have been efficient algorithms developed in other areas such as software verification and machine learning that solves similar problems. We will study some of them in the following sections, and develop sufficient adaptations and modifications of these algorithms to solve our problem.

The optimization problem (8.7), however, does not have a unique solution. To ensure a unique solution, we may formulate more sophisticated problems by adding/choosing other F, G, H functions. For example, we may minimize the number of states of the FSM model, or enforcing deterministicity of the FSM model, by modifying F and H respectively.

8.2 Review of FSM Abstraction Methods

8.2.1 Angluin's DFA Learning Algorithm

A DFA (Deterministic Finite Automaton) is a machine consisting of a 5-tuple: $(Q, \Sigma, \delta, Q_0, F)$, where Q is a finite set of states, Σ is a finite set of input alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, Q_0 is the start state, $F \subset Q$ is a set of accept states. Under the input sequence w, if the final state is in the set of accept states, the machine is said to accept w. Otherwise, it rejects w.

Given a DFA, Angluin's algorithm aims to learn the exact DFA from I/O traces of the DFA. The basic idea is to repeatedly conjecture machines according to I/O traces until an equivalent machine is obtained. The key data structure of the algorithm is an *observation table* which keeps track of necessary simulation results from which a machine can be conjectured.

An example of the observation table T is shown Table 8.2, where λ means "empty", the set of row labels is called S and the set of column labels is called E, according to Angluin's notation. To fill in each cell of the table, we concatenate the row label with the column label, use that as the input sequence to the DFA to perform a membership query (simulation), and fill in the yes/no answer (to whether the DFA accepts this input sequence) in the cell. The

observation table is further split into the top part and the bottom part, where row labels of the bottom part are obtained by concatenating row labels of the top part with all input alphabets. We will regard each row in the top part of the table as one state in the state machine, and therefore for each state in the top table, there must exist a row in the table that corresponds to its next state.

With the observation table, the key idea is then to differentiate different states by experiments, *i.e.*, we regard two rows as the same state if entries of the two rows are the same. For example, row "1" has the same entries as row "110" in Table 8.2, and therefore represent the same state. However, to conjecture a DFA from an observation table, we need to place two constraints on the table:

1. The table is *closed*, *i.e.*, every row in the bottom part of the table has a corresponding row in the top part with identical results in all columns. This guarantees that none of the rows in the bottom part corresponds to a new state.

2. The table is *consistent*, *i.e.*, every pair of rows in the top part of the table with identical results in all columns also has identical results when any alphabet symbol is added. Mathematically, $\forall s_1, s_2 \in S$, if $\text{row}(s_1) = \text{row}(s_2)$, then $\forall a \in \Sigma$, $\text{row}(s_1 \cdot a) = \text{row}(s_2 \cdot a)$. This guarantees that any pair of states that we think are the same are still the same if the experiment string is extended by one input alphabet symbol.

The procedures to make the observation table closed and consistent are simple, and can be found in [135].

Given a closed and consistent observation table (S, E, T), we can then conjecture a DFA $(Q, \Sigma, \delta, Q_0, F)$ defined by

$$\begin{aligned} Q &= \{\text{row}(s) : s \in S \text{ of top table}\}, \\ Q_0 &= \text{row}(\lambda), \\ \delta(\text{row}(s), a) &= \text{row}(s \cdot a), a \in \Sigma, \\ F &= \{\text{row}(s) : T(s, \lambda) = 1\}. \end{aligned} \tag{8.8}$$

Using this DFA, we can feed it to an "equivalence checker" to check if the DFA is equivalent to the original one. If equivalence checker returns "yes", we terminate the algorithm. Otherwise, it returns an counter-example. In that case, we add the counter-example into the observation table, and repeat the procedure of conjecturing a new DFA until convergence.

Summarizing the above procedure, the Angluin's algorithm is sketched in Algorithm 13.

8.2.2 Extension to Learning Mealy Machines

As defined in Section 8.1.1, the Mealy machine is similar to a DFA. It does not have accept states in DFA, but has an output function. This difference requires us to re-define

Algorithm 13 Angluin's DFA learning algorithm

1: Construct the initial observation table (S, E, T) with $S = \{\lambda\}$ and $E = \{\lambda\}$.
2: **repeat**
3: Make the observation table (S, E, T) closed and consistent.
4: Conjecture a DFA according to (8.8).
5: Check the equivalence between the conjectured DFA and the original DFA.
6: **if** not equivalent **then**
7: Add the counter-example in the observation table.
8: **end if**
9: **until** Equivalence checker returns "yes".

the meaning of the observation table.

In Angluin's algorithm, each cell in the table is filled in by yes/no answers to whether the DFA accepts an input sequence. This can be viewed as the "output" of the DFA. Similarly, for Mealy machines, we can fill in each cell in the table by the output of the Mealy machine. It turns out that using this new definition of the observation table, the concepts of closedness and consistence of the observation table still holds [136], and we can conjecture a Mealy machine $(Q, Q_0, \Sigma, \Lambda, \delta, G)$ from this new observation table by

$$\begin{aligned} Q &= \{\text{row}(s) : s \in S \text{ of top table}\}, \\ Q_0 &= \text{row}(\lambda), \\ \delta(\text{row}(s), a) &= \text{row}(s \cdot a), a \in \Sigma, \\ G(\text{row}(s), a) &= T(s, a), a \in \Sigma. \end{aligned} \tag{8.9}$$

Therefore, in order to learn a Mealy machine from another Mealy machine using Angluin's idea, we just need to use the new observation table, and replace step 4 in Algorithm 13 by "Conjecture a Mealy machine according to (8.9)".

8.3 Adaptation of Angluin's Algorithm

In this section, we describe adaptations of Angluin's algorithm to learn Mealy machines from differential equations. We show how we implement two key subroutines that are repeatedly called by Angluin's algorithm, and we interpret the states in the resulting Mealy machine in terms of regions in the original continuous state space. This connection also leads to a partitioning of the continuous state space.

In Angluin's algorithm, there are two subroutines that are repeatedly called:

1. The simulator. When building the observation table, the table entries are obtained by making membership queries (*i.e.*, is w accepted by the DFA?). In the case of learning Mealy machines, this corresponds to a simulation of the Mealy machine to

obtain the output. Therefore, we need a simulator that can produce I/O traces of a machine/system.

2. The equivalence checker. The algorithm decides whether to terminate by making equivalence queries (*i.e.*, is the new machine M equivalent to the original machine?).

8.3.1 Implementation of the Simulator

The simulation of circuit differential equations is straightforwardly implemented by SPICE transient analysis (numerical integration of the differential equations). However, three problems remain to initiate the simulation and interpret the results:

1. To start the simulation, we need to specify an initial condition that corresponds to the initial state of the Mealy machine;

2. We need to transform the input sequence $\{u_i\}$ (to the FSM) to the input waveform $u(t)$ (to differential equations).

3. After the simulation, we need to translate the continuous output waveforms to discrete outputs of the FSM.

To determine the initial condition, we propose to use the DC solution of the circuit. This is usually an important state for circuits since circuits typically settle down to this state when the input is fixed (except for special cases like oscillators).

The transformation from the input sequence $\{u_i\}$ to the input waveform $u(t)$ and the transformation from the output waveform $y(t)$ to output sequences $\{y_i\}$ are implemented by standard interpolation and quantization. However, one can define arbitrary transformations according to different applications. For example, the input sequence can specify phases of the input waveform at different time points, in which case the input waveforms are obtained by a transformation from the phase-domain to the voltage-domain. We can also specify binary values in the input sequence to be corresponding voltage levels of a digital square waveform during each clock cycle.

8.3.2 Implementation of the Equivalence Checker

To implement the equivalence checker, we use the $\mathcal{U}$-equivalence defined in Section 8.1.2. Therefore, we call the SPICE simulator to perform a set of simulations of circuit differential equations using $u(t)$ derived from $\{u_i\} \in \mathcal{U}$. If the sampled outputs $\{y(ih)\}$ lie in the corresponding quantization regions defined for $\{y_i\}$ (the output of the FSM to input sequence $\{u_i\}$), then we return "yes", *i.e.*, the FSM is equivalent to the differential equations. Otherwise, the input sequence is reported as an counter-example, and is added into the observation table to refine the FSM.

The training set used in the equivalence checking is important. One common implementation is to use a Monte-Carlo sampling of input signals if *a priori* statistics of inputs are known. Another implementation useful in digital applications is to try out all digital sequences of a fixed length N. Although this leads to 2^N possible sequences, it is expected that for practical circuits with small number of I/O ports, short sequences are enough to characterize behaviors of the circuit. Furthermore, we can add/remove some special sequences into the training set to ensure some desirable properties of the system. For example, we may say that we want to capture results by applying five "1"s at the input, and therefore we add the input sequence of five "1"s into the training set. For another example, we may have a constraint on the input signal that it is not possible to have inputs of six or more consecutive "1"s (due to properties of the circuit generating the input signal), and then the training set can be pruned to speedup the equivalence checking procedure.

8.3.3 Connections between FSM States and the Continuous State Space

The Mealy machine extracted from differential equations has a finite set of states. Since the FSM is sometimes extracted in order to understand system behaviors, it poses a question of what these states actually mean in the original system.

According to Angluin's algorithm, a state is defined to be the state that the system reaches by applying a corresponding input sequence. Therefore, strictly speaking, each state in the FSM corresponds to a single point in the continuous state space.

On the other hand, similar to $\mathcal{U}$-equivalence of two FSMs, we may define the $\mathcal{U}$-equivalence of two states. Namely, suppose x_1, x_2 are two states in the full continuous state space, if for all $u(t) \in \mathcal{U}$, the sampled and quantized versions of $y_1 = h(x_1, u)$ and $y_2 = h(x_2, u)$ match, then x_1 and x_2 are $\mathcal{U}$-equivalent. Using this definition, we see that there is a region around each point so that all point in that region are $\mathcal{U}$-equivalent.

For $\mathcal{U}$ being all binary inputs of length k, this definition partitions the continuous state space into 2^{k2^k} regions. This is a huge number. However, in real circuits, many of these regions are either never/rarely visited or extremely small, giving the premise that our algorithm can work well.

Solving for the boundaries of such regions, however, may be complicated and expensive. It is therefore preferable to obtain an approximation. To do this, we use the discrete points obtained above to construct a Voronoi tessellation/decomposition/diagram [137] which partitions a space with n points into convex polygons such that each polygon contains exactly one generating point and every point in a given polygon is closer to the generating point than to any other. These polygons, known as Voronoi polygons, are then associated with states in the FSM. As an example, a 3D Voronoi diagram with 4 points is shown in Fig. 8.7 in Section 6.5.

Due to properties of the Voronoi tessellation, it can be viewed as a natural partitioning

of the continuous state space given that the n points are obtained by the definition of each conjectured state in the algorithm. With this partitioning, it is then possible to apply other techniques to further exploit low-level dynamics of the circuit. For example, we can construct simple nonlinear/linear systems for each partition, and this is very similar to ideas in hybrid system abstraction of continuous dynamical systems [138], and trajectory-based MOR techniques [2].

The key difference between this approach and many previous methods (described in Section 8.2) on partitioning the continuous state space is that previous methods often suffer from curse of dimensionality due to the discretization of all state variables, while our approach avoids this problem by discretizing the state space directly. As long as the original system can be represented by a small FSM, our approach will be much more efficient. The number of partitions we obtain is the same as the size of the FSM, and we just need to store one point for each state to store the partitioning.

8.4 Examples and Experimental Results

8.4.1 A Latch Circuit

We consider a simplified latch circuit shown in Fig. 8.3. The latch is composed of a sampler (consisting of a multiplexer) and a regenerator (consisting of two cross-coupled inverters). Each output of the multiplexer and inverters is followed by an RC circuit that is not shown in Fig. 8.3. The latch samples the $DATA$ input when CLK is "1". When CLK is "0", two inverters are cross-coupled, and re-generate the output (QBB).

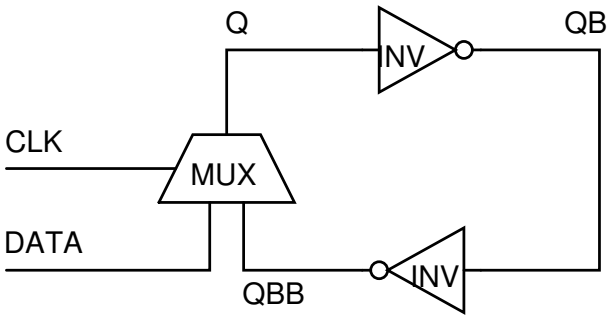


Figure 8.3: A simplified latch model.

The differential equations of this latch circuit are

$$\begin{aligned} RC\frac{dx_1}{dt} + (x_1 - f_{MUX}(u_1, u_2, x_3)) =& 0, \\ RC\frac{dx_2}{dt} + (x_2 - f_{INV}(x_1)) =& 0, \\ RC\frac{dx_3}{dt} + (x_3 - f_{INV}(x_2)) =& 0, \end{aligned} \tag{8.10}$$

where state variables x_1, x_2, x_3 are node voltages v_Q, v_{QB}, v_{QBB}, respectively; inputs u_1, u_2 are CLK and $DATA$, respectively; f_{INV} and f_{MUX} are I/O functions defined by

$$\begin{aligned} f_{INV}(x) =& -\tanh(10x), \\ f_{MUX}(C, A, B) =& \frac{A+B}{2} + \frac{B-A}{2}\tanh(10C), \end{aligned} \tag{8.11}$$

for the inverter and the multiplexer shown in Fig. 8.4.

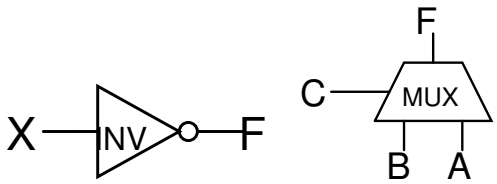


Figure 8.4: An inverter and a multiplexer.

One important erroneous behavior of this latch (also seen in other latch circuits) is that when the input swing is reduced due to power budget, the latch needs several consecutive "1"s or "0"s at the input in order to make a successful transition. For example, Fig. 8.5 shows the result of a transient simulation of the simplified model when $V_{sw} = 1.6V$ and the input sequence is "010001100011100". When a single "1" is present at the input, it is not strong enough to turn the output to "1" at the end of the cycle. However, when two or more consecutive "1"s arrive, the output transits from "0" to "1".

We expect to generate FSM abstractions that captures such error behaviors.

To apply our algorithm, we set the training set to be all input sequences of length 7 (7 clock cycles); the mapping $\alpha : \{u_i\} \mapsto u(t)$ is implemented to transform zero/one sequences to square waveforms of fixed duty cycle; the sampling time step h is chosen to be the clock cycle; and we set $y_i = 1$ if $y(ih) > 0$ and $y_i = 0$ otherwise.

Using this setting, we extract Mealy machines for this latch circuit with respect to input swing being $2V$, $1.6V$ and $1.3V$, as depicted in Fig. 8.6[2]. The numbers shown in each state

[2]Note that Mealy machines shown in Fig. 8.6 are indeed Moore machines, simply due to the fact that the

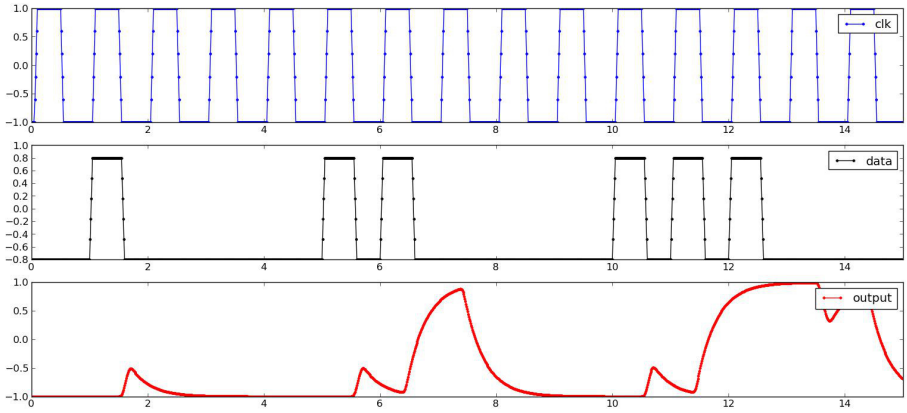


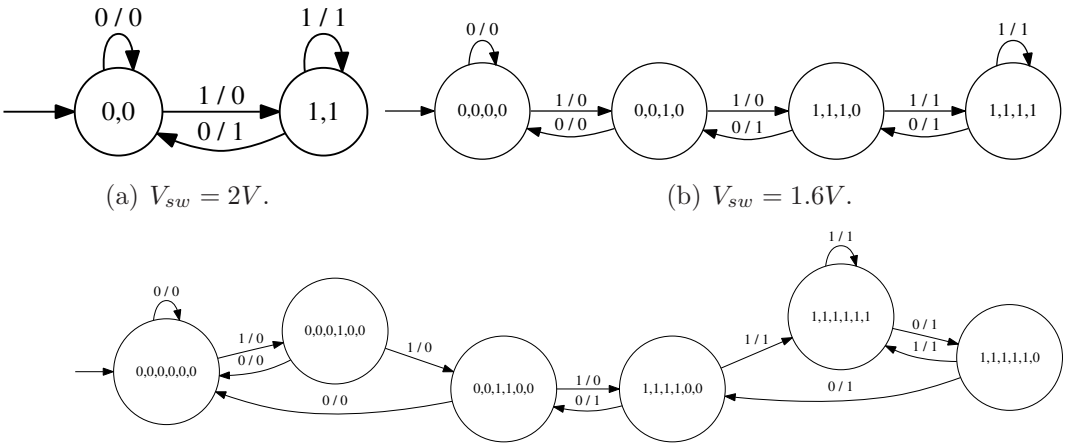
Figure 8.5: Response of the latch when the input sequence is "1001100000011001100000" and $V_{sw} = 1.6V$.

are not arbitrary. They are values of rows in the observation table, *i.e.*, the quantized output with respect to corresponding input sequences. As an example, the observation table of the latch with input swing 1.6 is shown in Table 8.2. We see that the state "0,0,0,0" corresponds to row λ, the state "0,0,1,0" corresponds to row "1", *etc.*. From Table 8.2, we can derive the FSM in Fig. 8.6(b) using (8.9).

Based on the Mealy machines in Fig. 8.6, we can claim that when input swing is $2V$, the latch behaves like an ideal latch; when input swing is $1.6V$, the input signal must persist for two consecutive cycles to avoid errors; when input swing is $1.3V$, the input signal must persist for three consecutive cycles to avoid errors. These properties can either be directly observed in simple FSMs, or be checked by model checking tools. To validate these claims, we perform many further SPICE simulations, and the same qualitative behaviors are observed.

Using these Mealy machines, we then construct Voronoi diagrams using discrete points corresponding to each state. For example, the Voronoi diagram of the Mealy machine in Fig. 8.6(b) is shown in Fig. 8.7. The four points correspond to four states in Fig. 8.6(b), and the (blue) trajectories in Fig. 8.7 are simulation traces that leads to the discrete points of different states, and show typical transition dynamics of the latch. This partitioning of the state space also provides direct intuition on how circuit behave: the two states at two corners represent solid "0" and solid "1" states; the two states in the middle capture dynamics in the metastable region, and can be viewed as weak "0" and weak "1" states.

output y is just the output voltage which corresponds to a state variable x_3. This is very common in circuit simulations, and a simple post-processing can be performed to obtain a Moore machine.



(a) $V_{sw} = 2V$. (b) $V_{sw} = 1.6V$.

(c) $V_{sw} = 1.3V$.

Figure 8.6: FSMs of the latch in Fig. 8.3.

Table 8.2: Observation table for $V_{sw} = 1.6V$

S \ E	0	1	10	00
λ	0	0	0	0
1	0	0	1	0
11	1	1	1	0
110	0	0	1	0
111	1	1	1	1
1110	1	1	1	0
11100	0	0	1	0
0	0	0	0	0
10	0	0	0	0
1100	0	0	0	0
1101	1	1	1	0
1111	1	1	1	1
11101	1	1	1	1
111000	0	0	0	0
111001	1	1	1	0

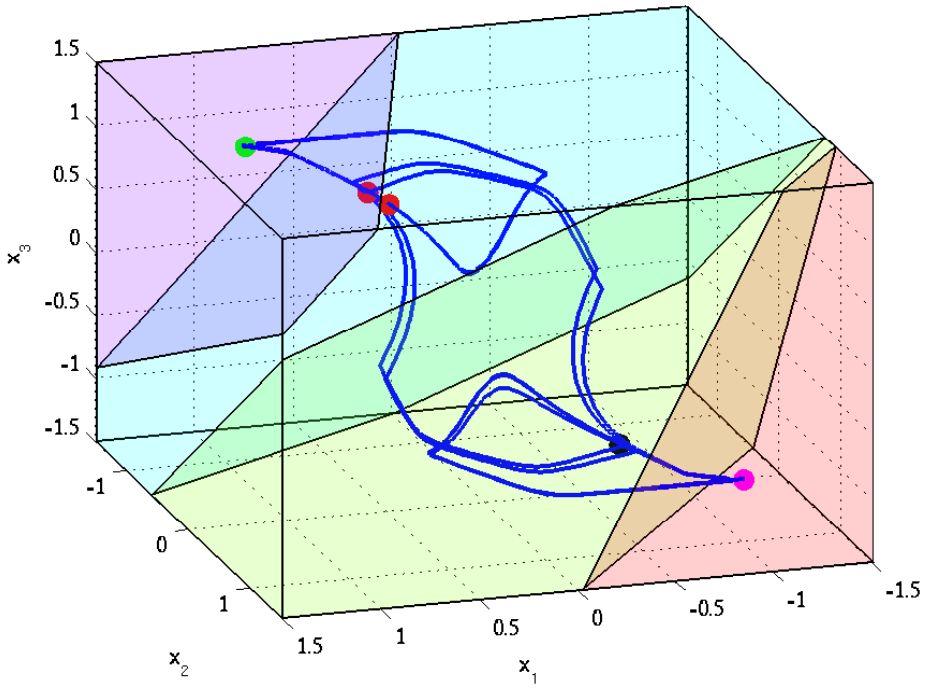


Figure 8.7: Interpretation of states in the FSM ($V_{sw} = 1.6V$).

8.4.2 An Integrator Circuit

The second example is an integrator circuit shown in Fig. 8.8, where $R = 1$, $C = 1$, and the opamp is approximated by a controlled voltage source which clips at $\pm 1V$ and has gain 10 and resistance 0.1. The input is a binary sequence where "1" and "0" are at voltage level $+V_{in}$ and $-V_{in}$, respectively, and each bit lasts for 1 second.

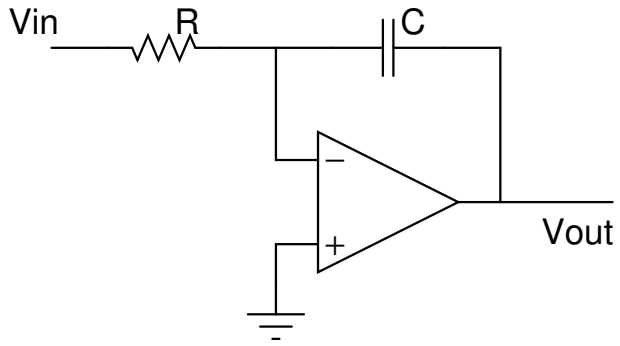


Figure 8.8: Circuit diagram of an integrator circuit.

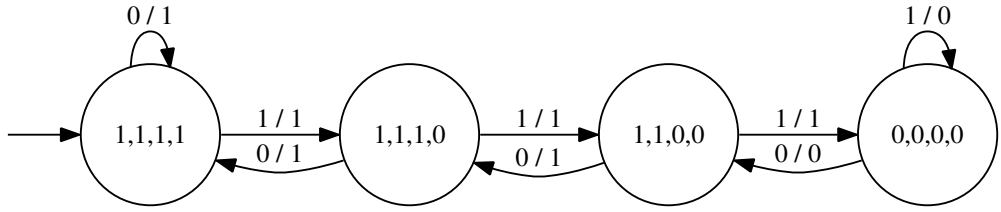
To initiate the algorithm, the step size h is chosen to be 1 second, *i.e.*, the period for each bit. The training set are all input sequences of length 8. The output is quantized by setting $y_i = 1$ when $y(ih) > 0$ and $y_i = 0$ otherwise. The initial state is chosen as the DC solution when the input is at the "0" level.

We first extract the FSM when the input level $V_{in} = 2/3V$. As shown in Fig. 8.9(a), the resulting FSM has 4 states. The "1,1,1,1" state corresponds to the state where the output is saturated at $1V$ and the "0,0,0,0" state corresponds to the state where the output is saturated at $-1V$. From this state machine, we immediately see how the integrator charges and discharges as different inputs are applied.

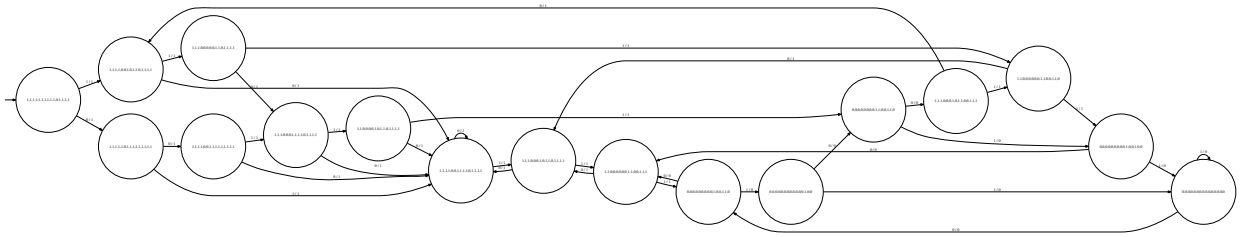
We stress that the initial condition/state in the learning algorithm is extremely important. Choosing a different initial condition may lead to FSMs with unnecessarily more states. For example, if the initial state is at a strange point which is almost never visited by the normally operating circuit, then this state is not representative of original circuit trajectories, and it makes the algorithm harder to explore important states by performing experiments. The heuristic we have used is to choose the initial state to be a DC solution. The DC state is always important in almost all circuits, and therefore is always good to be modeled in the FSM. Experiments also show that this heuristic leads to better FSMs (FSMs with less states and meaningful transitions).

However, in cases where the exact DC solution is not available or a random initial condition is chosen, we also have a simple heuristic to reduce the number of transient states in the FSM. The key idea is that since some transient states are never visited from other states, we may well discard these states. Equivalently, we should look for the strongly connected

components of the DAG (Directed Acyclic Graph) of the FSM, and this typically filters out a lot of transient states. Therefore, to discard useless transient states, we apply algorithms for finding strongly connected components, and fortunately, linear-time (with respect to the number of states) are available, such as [139].



(a) $V_{in} = 2/3V$.



(b) $V_{in} = 1/2V$.

Figure 8.9: FSMs of the integrator circuit.

We also extract the FSM of the integrator when the input level $V_{in} = 1/2V$. The FSM is shown in Fig. 8.9(b). (Since the number of states is relatively large, the labels are not shown clearly.) This state machine, however, is much more complicated than what we would imagine of an ideal integrator – we may think that this machine should have 5 states that are connected like the one in Fig. 8.9(a). This wrong intuition stems from the fact that the circuit is not an ideal integrator – the amount output increases/decreases depends not only on the input, but also on the current state, and this behavior is exhibited when the input level is relatively small. To capture the non-idealities, more states are identified by the algorithm.

While the FSM in Fig. 8.9(b) does not give direct intuition to designers, it does capture qualitative dynamics of the integrator. Therefore, we may further apply model checking techniques to this FSM to check high-level properties of the system. For example, we may check "whether there exists an input sequence of length 4 so that we can change the output of the integrator from $-1V$ to $1V$". When the FSM is a good approximation of the continuous system, we can trust the result returned by the model checker, or we can use that result as a guidance to check circuit properties by simulations.

Chapter 9

Conclusion and Future Work

Model order reduction for nonlinear dynamical systems has been a difficult problem for many years. This dissertation provides a self-contained view of the MOR problem, and presents several algorithms, ManiMOR, QLMOR, NTIM, DAE2FSM, to solve the problem from different perspectives.

Starting with the linear projection framework used by most previous techniques, we study and analyze its drawbacks in applications to reduce nonlinear systems. With that, we present a more general nonlinear projection framework which we believe is the natural and correct framework for nonlinear model order reduction. Indeed, the techniques presented in previous chapters may all be interpreted in this framework.

In its most straightforward form, we have presented ManiMOR which explicitly constructs and parameterizes a nonlinear manifold, and projects the full model on the manifold to obtain a reduced model. The projected model, by construction, matches the DC and AC responses of the full model, and can be proven to be much more efficient than linear projection-based methods.

By exploiting internal structures of the full model, we have proposed QLMOR which creates a QLDAE reduced model. The key contribution of this work is the development of a symbolic transformation procedure that creates an equivalent quadratic-linear model to the full model. By doing so, it has avoided all problems caused by Taylor series truncation, such as local convergence, high computational cost, passivity/stability loss, *etc.*. The other key contribution is the development of a reduction algorithm of QLDAEs which guarantees to match certain moments of the Volterra kernel for QLDAEs. Therefore, this reduced model is especially of great use for capturing harmonic distortions in nonlinear systems.

Using a very specific nonlinear manifold/projection, we have presented NTIM, a phase macromodel that is proven to capture phase responses in nonlinear systems. While we develop the model using Floquet theory, the phase macromodel naturally fits into the nonlinear projection framework. Due to the importance of timing/phase responses in practical systems, this model is shown to be useful in many scenarios.

Besides reduced order models in the form of differential equations, we also try to see if

other types of reduced models are appropriate. In particular, we presented DAE2FSM that abstracts a finite state machine from a set of differential equations. This model is especially useful if the full model behaves as a finite state machine – in that case, the algorithm guarantees to find the corresponding finite state machine. FSM models can be further fed into model checking tools to verify higher-level system properties, and have various applications such as mixed-signal system simulation and biological system abstraction.

Rather than completely solving the nonlinear MOR problem, our exploration leads to many more research directions to pursue. In the following, we comment on further improvements that may be done for the methods we present in this dissertation. We then sketch a few potential theoretical and practical improvements/extensions/considerations.

9.1 Future Work on Proposed Methods

9.1.1 ManiMOR

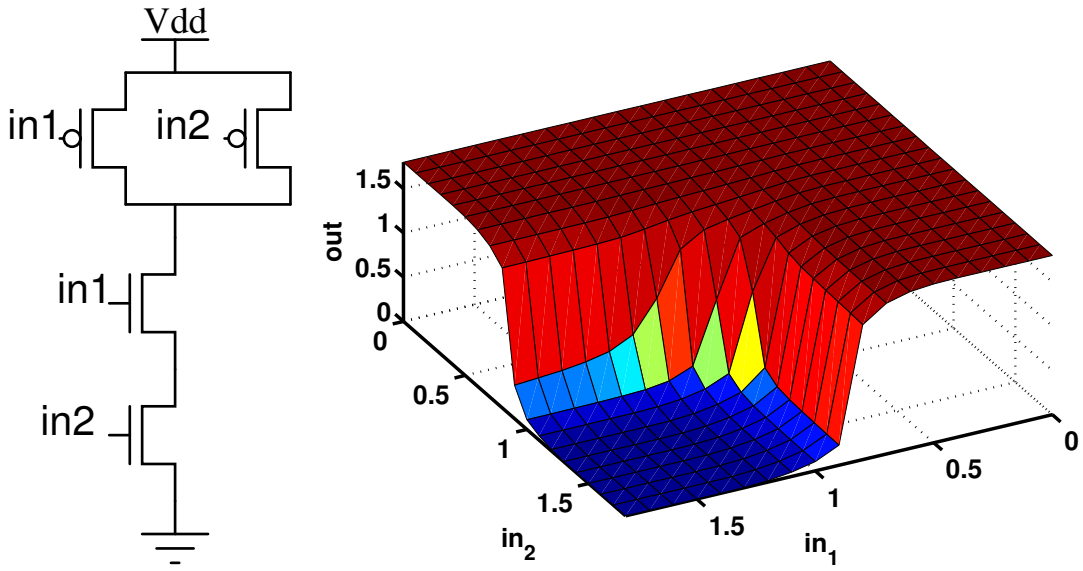
1. The Krylov distribution defined by (4.22) is generally not involutive. However, there are systems of which (4.22) is involutive. It is interesting to see what systems have such properties and what such properties correspond to in real circuit implementations.

2. By construction, ManiMOR ensures that the DC attractors are included in the manifold. This satisfies the necessary condition that the reduced order model can reproduce attractors of the full model. However, this is not sufficient for the stability to be preserved. It is useful to show under what conditions can the stability be preserved.

3. In the derivation of ManiMOR, we have assumed that $f(\cdot)$ in the differential equations is invertible. However, this does not always hold. For example, in bi-stable systems such as SRAMs, one DC input corresponds to two stable DC solutions. One possible solution is to find a manifold that covers all DC solutions. Another possible solution is to modify the circuit to a decoupled (open-loop) one such that $f(\cdot)$ becomes invertible, and then reduce the open-loop model.

4. MIMO systems may be handled by generalizing the manifold definition. For example, the DC manifold, which is a one-dimensional manifold for SIMO models, may be defined as a manifold of larger size. For example, Fig. 9.1 shows a 2-input NAND circuit and the 2-dimensional DC manifold. Following this idea, we may also extend the definition of AC and ManiMOR manifolds.

5. For oscillators, DC equilibria are not the attractors. Instead, the periodic solution, or the limit cycle in the state space is the attractor. Therefore, the definition of the manifold should be changed for oscillators. It seems that this has several connections to the NTIM reduced model that deserve further investigations.



(a) circuit diagram (b) DC surface

Figure 9.1: Asymptotic DC behavior of a NAND gate.

6. In data mining and statistical learning community, there have been several work in manifold learning (from data). These techniques may be borrowed to construct a manifold for the purpose of model order reduction. The main difficulty is that most of these methods do not define projection functions, and are highly data-driven. Appropriate adaptation must be done for these methods to be applicable in model order reduction.

9.1.2 QLMOR

1. Polynomial systems are easier to handle not only in model order reduction, but also in simulation and verification. For example, SOStools [140] may be used to construct Lyapunov functions for polynomial systems; the simulation of quadratic systems may be much faster than more complicated nonlinear systems. Therefore, the quadratic-linearization procedure serves as a key transformation to enable the application of tools that handle polynomial systems to non-polynomial systems.

2. Chemical rate equations contain only polynomial functions in the ODEs. Therefore, there is a one-to-one correspondence between a set of polynomial ODEs and a set of chemical reactions. Therefore, we may synthesize chemical reactions from ODEs even if they are non-polynomial.

3. From a different perspective, the quadratic-linearization procedure shows that any nonlinear system may be written as a QLDAE. Therefore, the QLDAE form is an acceptable form for nonlinear system identification, and it represents a nonlinear system by matrices. We may modify current system identification methods to identify QLDAE models.

9.1.3 NTIM

1. NTIM in Chapter 7 generates phase models for only single-input systems. It is not applicable to multiple-input systems because these system will settle in a quasi-periodic steady state. From a projection view point, the manifold is no longer the limit cycle, but a torus in the state space. It appears that multiple phase variables and multiple phase equations need to be defined for systems in quasi-periodic steady states.

2. The phase model may be used in timing analysis of a large digital circuit – we may replace each digital gate by its phase model, therefore reducing the number of equations describing each gate to 1. This greatly speeds up the simulation, while preserves timing behaviors.

9.1.4 DAE2FSM

1. DAE2FSM learns an FSM from trajectories. An alternative is to directly construct an FSM from the differential equations. This again requires us to define an appropriate notion of equivalence between the DAEs and the FSMs. Some initial study show that for simple linear or nonlinear systems, we may have rigorous definitions and efficient algorithms. We are following this line of research, and hope to reach more general results and algorithms.

2. With the FSM learned, we may apply model checking or probabilistic model checking tools to check higher-level system behaviors. In particular, for the latch circuit (to be used in a communication link), we have developed an efficient algorithm to compute the BER (bit-error rate) of the link given the finite state machine of the latch, assuming the errors are caused by non-idealities in the latch.

3. Besides deterministic FSM learning techniques, there have also been techniques to learn non-deterministic or probabilistic machines. How to employ these techniques in learning from DAE models remains a question. We are studying a few of these representative algorithms and exploring the possibility of adaptation of these algorithms.

9.2 Other Future Work

1. While we have mainly considered the model reduction of ODE/DAE models, the model order reduction of other models in Table 1.1 deserve further investigation. For example, we may reduce an FSM model by merging/removing states. The question is again what are the "important" states, and how to identify these states. A straightforward idea is to preserve the attractors of the FSM. Similarly, we can explore the model order reduction of other models.

2. Chemical reactions may also be viewed as a model, although it not listed in Table 1.1. While it describes chemical kinetics, it is not strictly a mathematical model because it may be interpreted as many different models, deterministic or stochastic, finite-state or infinite-state. The reduction of the chemical reactions is possible – by ignoring the reactions that have a high reaction rate, we may obtain a model that captures the slow kinetic dynamics. However, this method is not "optimal" generally in other senses (such as attractor-preserving, moment-matching, *etc.*). It is interesting to explore the application of other MOR techniques or develop new MOR techniques for the chemical reactions.

3. We may also look at the problem of abstracting one model from another. For example, from an ODE/DAE model to a hybrid model. Such abstractions can have a lot of applications, *e.g.*, to simplify the implementations/synthesis of a controller.

4. One important aspect of the projection framework is that it is computationally efficient to obtain a projection that leads to a reasonable reduced order model. However, recently there have been many progresses in improving the efficiency of optimization algorithms. The direct application of the state-of-the-art optimization routines may be possible.

5. We have mentioned in Section 2.1.3 the problem of formal verification of analog/mixed-signal systems. However, such algorithms and tools are extremely immature, and deserves further investigation. While this is not about model order reduction, it closely relates to MOR in that such tools may be used to check the correctness of reduced models.

6. Practical systems may be redundant. For example, the LTI system matrices may be rank-deficient. It looks like MOR method may remove the redundancy in such systems. It is interesting to see if MOR methods can actually preserve certain redundancies.

7. In several structure-preserving MOR techniques, certain structures (of LTI systems) are preserved so that the reduced model corresponds to an RLC circuit implementation. A similar problem is how to obtain a nonlinear model that corresponds to a transistor-level implementation.

8. There have been some literature discussing how to do nonlinear balanced truncation to obtain a reduced model. However, the theory of nonlinear balancing has not reached a consensus, and the existing methods are computationally extremely inefficient.

9. The theory for the balanced realization of LTV systems seems to be well-established. However, the circuit community have not been aware of these techniques. We may try these techniques and see how they work for circuit examples.

10. We have shown that some nonlinear systems, under appropriate coordinate transformations, may be converted to equivalent linear systems. However, the question is why would we want to implement an LTI system using a nonlinear system? One possibility is that the LTI implementation may require unrealistic resistance and capacitance values that cannot be mapped to an integrated circuit. In that case, it would be great if we can map the LTI system to a nonlinear system that can be implemented on chip.

11. The sparsity is typically destroyed by MOR methods. However, the discussion in Chapter 3 reveals that sometimes we have extra degree of freedom in choosing a projected model. These degree of freedom may be used to achieve the sparsity of the reduced order model.

Bibliography

[1] R.T. Chang et al. Near Speed-of-Light Signaling Over On-chip Electrical Interconnects. *Solid-State Circuits, IEEE Journal of*, 38:834–838, May 2003.

[2] Michal Rewienski and Jacob White. A Trajectory Piecewise-Linear Approach to Model Order Reduction and Fast Simulation of Nonlinear Circuits and Micromachined Devices. *IEEE Transactions on Computer-Aided Design*, 22(2), February 2003.

[3] Y. Wang and H.G. Dohlman. Pheromone Signaling Mechanisms in Yeast: A Prototypical Sex Machine. *Science*, 306:1508–1509, Nov 2004.

[4] K.C. Sou, A. Megretski, and L. Daniel. A quasi-convex optimization approach to parameterized model order reduction. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(3):456–469, 2008.

[5] L. Daniel, O.C. Siong, L.S. Chay, K.H. Lee, and J. White. A multiparameter moment-matching model-reduction approach for generating geometrically parameterized interconnect performance models. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(5):678–693, 2004.

[6] B.N. Bond and L. Daniel. A piecewise-linear moment-matching approach to parameterized model-order reduction for highly nonlinear systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(12):2116–2129, 2007.

[7] P. Li, F. Liu, X. Li, L.T. Pileggi, and S.R. Nassif. Modeling interconnect variability using efficient parametric model order reduction. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 2*, pages 958–963. IEEE Computer Society, 2005.

[8] X. Li, P. Li, and L.T. Pileggi. Parameterized interconnect order reduction with explicit-and-implicit multi-parameter moment matching for inter/intra-die variations. In *iccad*, pages 806–812. IEEE, 2005.

[9] T. Bui-Thanh, K. Willcox, and O. Ghattas. Model reduction for large-scale systems with high-dimensional parametric input space. *SIAM Journal on Scientific Computing*, 30(6):3270–3288, 2008.

[10] Y. Liu, L.T. Pileggi, and A.J. Strojwas. Model order-reduction of RC (L) interconnect including variational analysis. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 201–206. ACM, 1999.

[11] 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*, pages 58–65. IEEE Computer Society, 1997.

[12] J.M. Wang, C.C. Chu, Q. Yu, and E.S. Kuh. On projection-based algorithms for model-order reduction of interconnects. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 49(11):1563–1585, 2002.

[13] I.M. Elfadel and D.D. Ling. A block rational Arnoldi algorithm for multipoint passive model-order reduction of multiport RLC networks. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pages 66–71. IEEE Computer Society, 1997.

[14] Y. Shin and T. Sakurai. Power distribution analysis of VLSI interconnects using model order reduction. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(6):739–745, 2002.

[15] A.C. Cangellaris, M. Celik, S. Pasha, and L. Zhao. Electromagnetic model order reduction for system-level modeling. *Microwave Theory and Techniques, IEEE Transactions on*, 47(6):840–850, 1999.

[16] H. Wu and A.C. Cangellaris. Model-order reduction of finite-element approximations of passive electromagnetic devices including lumped electrical-circuit models. *Microwave Theory and Techniques, IEEE Transactions on*, 52(9):2305–2313, 2004.

[17] T. Wittig, I. Munteanu, R. Schuhmann, and T. Weiland. Two-step Lanczos algorithm for model order reduction. *Magnetics, IEEE Transactions on*, 38(2):673–676, 2002.

[18] R.D. Slone, R. Lee, and J.F. Lee. Multipoint Galerkin asymptotic waveform evaluation for model order reduction of frequency domain FEM electromagnetic radiation problems. *Antennas and Propagation, IEEE Transactions on*, 49(10):1504–1513, 2001.

[19] M.E. Kowalski and J.M. Jin. Model-order reduction of nonlinear models of electromagnetic phased-array hyperthermia. *Biomedical Engineering, IEEE Transactions on*, 50(11):1243–1254, 2003.

[20] H. Yu, Y. Shi, and L. He. Fast analysis of structured power grid by triangularization based structure preserving model order reduction. In *Proceedings of the 43rd annual Design Automation Conference*, pages 205–210. ACM, 2006.

[21] H. Su, E. Acar, and S.R. Nassif. Power grid reduction based on algebraic multigrid principles. 2003.

[22] T.V. Nguyen and J.M. Wang. Extended Krylov subspace method for reduced order analysis of linear circuits with multiple sources. In *dac*, pages 247–252. ACM, 2000.

[23] J. Wood, D.E. Root, and N.B. Tufillaro. A behavioral modeling approach to nonlinear model-order reduction for RF/microwave ICs and systems. *Microwave Theory and Techniques, IEEE Transactions on*, 52(9):2274–2284, 2004.

[24] P. Li and L.T. Pileggi. NORM: compact model order reduction of weakly nonlinear systems. 2003.

[25] J.R. Phillips. Projection-based approaches for model reduction of weakly nonlinear, time-varying systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(2):171–187, 2003.

[26] X. Li, P. Li, Y. Xu, and L.T. Pileggi. Analog and RF circuit macromodels for system-level analysis. 2003.

[27] A. Zhu and T.J. Brazil. Behavioral modeling of RF power amplifiers based on pruned Volterra series. *Microwave and Wireless Components Letters, IEEE*, 14(12):563–565, 2004.

[28] J. Roychowdhury. Reduced-order modeling of time-varying systems. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 46(10):1273–1288, 1999.

[29] A. Demir, A. Mehrotra, and J. Roychowdhury. Phase noise in oscillators: A unifying theory and numerical methods for characterization. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 47(5):655–674, 2000.

[30] X. Lai and J. Roychowdhury. TP-PPV: piecewise nonlinear, time-shifted oscillator macromodel extraction for fast, accurate PLL simulation. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 269–274. ACM, 2006.

[31] X. Lai and J. Roychowdhury. Macromodelling oscillators using Krylov-subspace methods. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pages 527–532. IEEE Press, 2006.

[32] B. Gu, K.K. Gullapalli, S. Hamm, B. Mulvaney, X. Lai, and J. Roychowdhury. Implementing nonlinear oscillator macromodels using Verilog-AMS for accurate prediction of injection locking behaviors of oscillators. In *Behavioral Modeling and Simulation*

Workshop, 2005. BMAS 2005. Proceedings of the 2005 IEEE International, pages 43–47. IEEE, 2005.

[33] J.S. Han, E.B. Rudnyi, and J.G. Korvink. Efficient optimization of transient dynamic problems in MEMS devices using model order reduction. *Journal of Micromechanics and Microengineering*, 15:822, 2005.

[34] J. Lienemann, E.B. Rudnyi, and J.G. Korvink. MST MEMS model order reduction: Requirements and benchmarks. *Linear algebra and its applications*, 415(2-3):469–498, 2006.

[35] E. Rudnyi and J. Korvink. Model order reduction for large scale engineering models developed in ANSYS. In *Applied Parallel Computing*, pages 349–356. Springer, 2006.

[36] J. Chen and S.M. Kang. Model-order reduction of nonlinear mems devices through arclength-based Karhunen-Loéve decomposition. In *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, volume 3, pages 457–460. IEEE, 2001.

[37] E.B. Rudnyi and J.G. Korvink. Review: Automatic Model Reduction for Transient Simulation of MEMS-based Devices. *Sensors Update*, 11(1):3–33, 2002.

[38] YC Liang, WZ Lin, HP Lee, SP Lim, KH Lee, and H. Sun. Proper orthogonal decomposition and its applications-part II: Model reduction for MEMS dynamical analysis. *Journal of Sound and Vibration*, 256(3):515–532, 2002.

[39] P. Benner, V. Mehrmann, and D.C. Sorensen. Dimension reduction of large-scale systems. 2005.

[40] K. Willcox and J. Peraire. Balanced model reduction via the proper orthogonal decomposition. *AIAA journal*, 40(11):2323–2330, 2002.

[41] P.V. Kokotovic et al. Singular perturbations and order reduction in control theory–an overview. *Automatica*, 12(2):123–132, 1976.

[42] M. Bordemann and J. Hoppe. The Dynamics of Relativistic Membranes I: Reduction to 2-dimensional Fluid Dynamics. *Arxiv preprint hep-th/9307036*, 1993.

[43] C.W. Rowley, T. Colonius, and R.M. Murray. Model reduction for compressible flows using POD and Galerkin projection. *Physica D: Nonlinear Phenomena*, 189(1-2):115–129, 2004.

[44] B.F. Farrell and P.J. Ioannou. Accurate low-dimensional approximation of the linear dynamics of fluid flow. *Journal of the atmospheric sciences*, 58(18):2771–2789, 2001.

[45] X. Ma and G.E. Karniadakis. A low-dimensional model for simulating three-dimensional cylinder flow. *Journal of Fluid Mechanics*, 458(-1):181–190, 2002.

[46] B.R. Noack, M. Schlegel, M. Morzyński, and G. Tadmor. System reduction strategy for Galerkin models of fluid flows. *International Journal for Numerical Methods in Fluids*, 63(2):231–248, 2010.

[47] I.K. Fodor. A survey of dimension reduction techniques. *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*, 2002.

[48] K.R. Schneider and T. Wilhelm. Model reduction by extended quasi-steady-state approximation. *Journal of mathematical biology*, 40(5):443–450, 2000.

[49] C.W. Rowley, I.G. Kevrekidis, J.E. Marsden, and K. Lust. Reduction and reconstruction for self-similar dynamical systems. *Nonlinearity*, 16:1257, 2003.

[50] D. Lebiedz, V. Reinhardt, and J. Kammerer. Novel trajectory based concepts for model and complexity reduction in (bio) chemical kinetics. *Model Reduction and Coarse-Graining Approaches for Multiscale Phenomena*, pages 343–364.

[51] S. Danø, M.F. Madsen, H. Schmidt, and G. Cedersund. Reduction of a biochemical model with preservation of its basic dynamic properties. *FEBS Journal*, 273(21):4862–4877, 2006.

[52] T.B. Kepler, LF Abbott, and E. Marder. Reduction of conductance-based neuron models. *Biological Cybernetics*, 66(5):381–387, 1992.

[53] D. Golomb, J. Guckenheimer, and S. Gueron. Reduction of a channel-based model for a stomatogastric ganglion LP neuron. *Biological cybernetics*, 69(2):129–137, 1993.

[54] W.M. Kistler, W. Gerstner, and J.L. Hemmen. Reduction of the Hodgkin-Huxley equations to a single-variable threshold model. *Neural Computation*, 9(5):1015–1045, 1997.

[55] E.M. Izhikevich. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569–1572, 2003.

[56] D. Hansel, G. Mato, and C. Meunier. Phase dynamics for weakly coupled Hodgkin-Huxley neurons. *EPL (Europhysics Letters)*, 23:367, 1993.

[57] Y. Maeda, K. Pakdaman, T. Nomura, S. Doi, and S. Sato. Reduction of a model for an Onchidium pacemaker neuron. *Biological cybernetics*, 78(4):265–276, 1998.

[58] B. Cartling. Response characteristics of a low-dimensional model neuron. *Neural computation*, 8(8):1643–1652, 1996.

[59] K. Doya and A.I. Selverston. Dimension reduction of biological neuron models by artificial neural networks. *Neural Computation*, 6(4):696–717, 1994.

[60] S. Peleš, B. Munsky, and M. Khammash. Reduction and solution of the chemical master equation using time scale separation and finite state projection. *The Journal of chemical physics*, 125:204104, 2006.

[61] L. Petzold and W. Zhu. Model reduction for chemical kinetics: An optimization approach. *AIChE journal*, 45(4):869–886, 1999.

[62] M.S. Okino and M.L. Mavrovouniotis. Simplification of mathematical models of chemical reaction systems. *Chemical reviews*, 98(2):391–408, 1998.

[63] D. Lebiedz. Computing minimal entropy production trajectories: An approach to model reduction in chemical kinetics. *The Journal of chemical physics*, 120:6890, 2004.

[64] S. Lall, J.E. Marsden, and S. Glavaski. A subspace approach to balanced truncation for model reduction of nonlinear control systems. *International Journal of Robust and Nonlinear Control*, 12(6):519–535, 2002.

[65] U. Maas and S.B. Pope. Simplifying chemical kinetics: intrinsic low-dimensional manifolds in composition space. *Combustion and Flame*, 88(3-4):239–264, 1992.

[66] Z. Ren, S.B. Pope, A. Vladimirsky, and J.M. Guckenheimer. The invariant constrained equilibrium edge preimage curve method for the dimension reduction of chemical kinetics. *The Journal of chemical physics*, 124:114111, 2006.

[67] E.A. Mastny, E.L. Haseltine, and J.B. Rawlings. Two classes of quasi-steady-state model reductions for stochastic kinetics. *The Journal of chemical physics*, 127:094106, 2007.

[68] G. Kerschen, J.C. Golinval, A.F. Vakakis, and L.A. Bergman. The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: an overview. *Nonlinear Dynamics*, 41(1):147–169, 2005.

[69] S. Lall, P. Krysl, and J.E. Marsden. Structure-preserving model reduction for mechanical systems. *Physica D: Nonlinear Phenomena*, 184(1-4):304–318, 2003.

[70] P. Koutsovasilis and M. Beitelschmidt. Comparison of model reduction techniques for large mechanical systems. *Multibody System Dynamics*, 20(2):111–128, 2008.

[71] V. Duindam and S. Stramigioli. Energy-based model-reduction of nonholonomic mechanical systems. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4584–4589. IEEE, 2004.

[72] T. Kailath. *Linear systems*, volume 1. Prentice-Hall Englewood Cliffs, NJ, 1980.

[73] S. Sastry. *Nonlinear systems: analysis, stability, and control*. Springer Verlag, 1999.

[74] A. Odabasioglu, M. Celik, and L. T. Pileggi. Prima: passive reduced-order interconnect macromodeling algorithm. 17(8):645–654, 1998.

[75] C. Gu and J. Roychowdhury. ManiMOR: Model Reduction via Projection onto Nonlinear Manifolds, with Applications to Analog Circuits and Biochemical Systems. In *Proceedings of the IEEE International Conference on Computer Aided Design*, 10-13 Nov. 2008.

[76] C. Gu. QLMOR: a New Projection-Based Approach for Nonlinear Model Order Reduction. In *Proceedings of the IEEE International Conference on Computer Aided Design*, 2-5 Nov. 2009.

[77] C. Gu and J. Roychowdhury. Generalized Nonlinear Timing/Phase Macromodeling: Theory, Numerical Methods and Applications. In *Proceedings of the IEEE International Conference on Computer Aided Design*, 7-11 Nov. 2010.

[78] C. Gu and J. Roychowdhury. FSM Model Abstraction for Analog/Mixed-Signal Circuits by Learning from I/O Trajectories. *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2011.

[79] S. Gupta, B. H. Krogh, and R. A. Rutenbar. Towards formal verification of analog designs. In *Proc. ICCAD-2004 Computer Aided Design IEEE/ACM Int. Conf*, pages 210–217, 2004.

[80] S. Blouin. *Finite-state machine abstractions of continuous systems*. PhD thesis, Queen's University, 2003.

[81] P. Ye, E. Entcheva, R. Grosu, and S.A. Smolka. Efficient modeling of excitable cells using hybrid automata. In *Proc. of CMSB*, pages 216–227. Citeseer, 2005.

[82] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, 2007.

[83] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[84] J. Raisch and S.D. O'Young. Discrete approximation and supervisory control of continuous systems. *Automatic Control, IEEE Transactions on*, 43(4):569–573, 1998.

[85] M.T. Heath. *Scientific computing*. McGraw-Hill, 1997.

[86] R.K. Brayton. *Logic minimization algorithms for VLSI synthesis*. Springer, 1984.

[87] S. Devadas, A. Ghosh, and K. Keutzer. *Logic synthesis*. McGraw-Hill, Inc. New York, NY, USA, 1994.

[88] R. Harjani, R.A. Rutenbar, and L.R. Carley. OASYS: A framework for analog circuit synthesis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 8(12):1247–1266, 1989.

[89] E.J. Grimme. *Krylov Projection Methods for Model Reduction*. PhD thesis, University of Illinois, EE Dept, Urbana-Champaign, 1997.

[90] B.N. Bond. *Stability-Preserving Model Reduction for Linear and Nonlinear Systems Arising in Analog Circuit Applications*. PhD thesis, Massachusetts Institute of Technology, 2010.

[91] B.N. Bond and L. Daniel. Guaranteed stable projection-based model reduction for indefinite and unstable linear systems. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 728–735. IEEE Press, 2008.

[92] S.K. Tiwary, A. Gupta, J.R. Phillips, C. Pinello, and R. Zlatanovici. fSpice: a boolean satisfiability based approach for formally verifying analog circuits. In *Proc. Workshop on Formal Verification of Analog Circuits (FAC), Princeton*, 2008.

[93] S.K. Tiwary, A. Gupta, J.R. Phillips, C. Pinello, and R. Zlatanovici. First steps towards SAT-based formal analog verification. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 1–8. ACM, 2009.

[94] A. Singh and P. Li. On behavioral model equivalence checking for large analog/mixed signal systems. In *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, pages 55–61. IEEE.

[95] David C. Walter. *Verification of Analog and Mixed-Signal Circuits Using Symbolic Methods*. PhD thesis, University of Utah, ECE Dept, 2007.

[96] Scott Little. *Efficient Modeling and Verification of Analog/Mixed-Signal Circuits Using Labeled Hybrid Petri Nets*. PhD thesis, University of Utah, ECE Dept, 2008.

[97] L. Ljung and E.J. Ljung. *System identification: theory for the user*, volume 280. Prentice-Hall Upper Saddle River, NJ, 1987.

[98] Rolf Isermann and Marco Mnchhof. *Identification of Dynamic Systems: An Introduction with Applications*. Springer, 2011.

[99] G.A. Baker and P.R. Graves-Morris. *Padé approximants*. Cambridge Univ Pr, 1996.

[100] L.T. Pillage and R.A. Rohrer. Asymptotic waveform evaluation for timing analysis. *IEEE Transactions on Computer-Aided Design*, 9:352–366, April 1990.

[101] 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*, 14(5):639–649, 1995.

[102] G.H. Golub and C.F. Van Loan. *Matrix computations*. Johns Hopkins Univ Pr, 1996.

[103] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial Mathematics, 2003.

[104] Bruce Moore. Principal Component Analysis in Linear Systems: Controllability, Observability, and Model Reduction. *IEEE Trans. Automatic Control*, 26:17–32, February 1981.

[105] W. Rugh. *Nonlinear System Theory - The Volterra-Wiener Approach*. Johns Hopkins Univ Press, 1981.

[106] 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*, 24(2):184–203, 2005.

[107] Joel R. Phillips. Projection-Based Approaches for Model Reduction of Weakly Nonlinear Time-Varying Systems. *IEEE Transactions on Computer-Aided Design*, 22(2):171–187, 2003.

[108] L. Pillage. *Electronic circuit and system simulation methods*. McGraw-Hill Professional, 1998.

[109] J. Roychowdhury. Numerical Simulation and Modelling of Electronic and Biochemical Systems. *Foundations and Trends® in Electronic Design Automation*, 3(2-3):97–303, 2008.

[110] D. Garfinkel, L. Garfinkel, M. Pring, S.B. Green, and B. Chance. Computer applications to biochemical kinetics. *Annual Review of Biochemistry*, 39(1):473–498, 1970.

[111] P.A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton Univ Pr, 2008.

[112] J.M. Lee. *Introduction to smooth manifolds*. Springer Verlag, 2003.

[113] M. Spivak and M. Spivak. A comprehensive introduction to differential geometry. 1979.

[114] Dmitry Vasilyev, Michal Rewienski, and Jacob White. A TBR-based Trajectory Piecewise-Linear Algorithm for Generating Accurate Low-order Models for Nonlinear Analog Circuits and MEMS. *Proceedings of the IEEE Design Automation Conference*, 2003.

[115] J. Savoj and B. Razavi. A 10-Gb/s CMOS clock and data recovery circuit with a half-rate linear phase detector. *IEEE Journal of Solid-State Circuits*, 36:761–768, May 2001.

[116] Ning Dong and Jaijeet Roychowdhury. Piecewise Polynomial Nonlinear Model Reduction. *Proceedings of the IEEE Design Automation Conference*, 2003.

[117] Ning Dong and Jaijeet Roychowdhury. Automated Nonlinear Macromodelling of Output Buffers for High-Speed Digital Applications. *Proceedings of the IEEE Design Automation Conference*, 2005.

[118] BSIM3. http://www-device.eecs.berkeley.edu/~bsim3.

[119] B. Kofahl and E. Klipp. Modelling the dynamics of the yeast pheromone pathway. *Yeast*, 21:831–850, 2004.

[120] T.M. Thomson and D. Endy. Rapid Characterization of Cellular Pathways Using Time-Varying Signals. In *Sixth International Conference on Systems Biology*, Oct 2005.

[121] J. Roychowdhury. Reduced-order modelling of time-varying systems. *IEEE Trans. Ckts. Syst. – II: Sig. Proc.*, 46(10), November 1999.

[122] T. Sakurai and A.R. Newton. Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. *Solid-State Circuits, IEEE Journal of*, 25(2):584–594, 1990.

[123] G. Gildenblat, X. Li, W. Wu, H. Wang, A. Jha, R. van Langevelde, G.D.J. Smit, A.J. Scholten, and D.B.M. Klaassen. PSP: An advanced surface-potential-based MOSFET model for circuit simulation. *Electron Devices, IEEE Transactions on*, 53(9):1979–1993, 2006.

[124] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge Univ Pr, 2004.

[125] H. Shichman and D.A. Hodges. Modeling and simulation of insulated-gate field-effect transistor switching circuits. *Solid-State Circuits, IEEE Journal of*, 3(3):285–289, 1968.

[126] D.L. Boley. Krylov space methods on state-space control models. *IEEE Trans. Ckts. Syst. – II: Sig. Proc.*, 13(6):733–758, 1994.

[127] J. Phillips. Model Reduction of Time-Varying Linear Systems Using Approximate Multipoint Krylov-Subspace Projectors. In *Proceedings of the IEEE International Conference on Computer Aided Design*, November 1998.

[128] G. Hachtel, R. Brayton, and F. Gustavson. The sparse tableau approach to network analysis and design. *Circuit Theory, IEEE Transactions on*, 18(1):101–113, 1971.

[129] E.M. Izhikevich. *Dynamical systems in neuroscience: The geometry of excitability and bursting*. The MIT press, 2007.

[130] A. Demir. Floquet theory and non-linear perturbation analysis for oscillators with differential-algebraic equations. *International journal of circuit theory and applications*, 28(2):163–185, 2000.

[131] E.A. Coddington and N. Levinson. *Theory of ordinary differential equations*. Tata McGraw-Hill, 1972.

[132] K.S. Kundert, J.K. White, and A. Sangiovanni-Vincentelli. *Steady-state methods for simulating analog and microwave circuits*. Kluwer Academic Publishers, 1990.

[133] A. Demir and J. Roychowdhury. A reliable and efficient procedure for oscillator PPV computation, with phase noise macromodeling applications. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(2):188–197, 2003.

[134] K. Tsumoto, H. Kitajima, T. Yoshinaga, K. Aihara, and H. Kawakami. Bifurcations in Morris-Lecar neuron model. *Neurocomputing*, 69(4-6):293–316, 2006.

[135] D. Angluin. Learning regular sets from queries and counterexamples* 1. *Information and computation*, 75(2):87–106, 1987.

[136] K. Li, R. Groz, and M. Shahbaz. Integration testing of components guided by incremental state machine learning. In *Testing: Academic and Industrial Conference-Practice And Research Techniques, 2006. TAIC PART 2006. Proceedings*, pages 59–70. IEEE, 2006.

[137] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag New York Inc, 2008.

[138] G.J. Pappas. *Hybrid systems: Computation and abstraction*. PhD thesis, UNIVERSITY of CALIFORNIA, 1998.

[139] B. Aspvall. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inform. Process. Lett.*, 8:121–123, 1979.

[140] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*, 2004.

[141] B. law Jakubczyk. Introduction to Geometric Nonlinear Control; Controllability and Lie Bracket.

[142] Michal Rewienski. *A Trajectory Piecewise-Linear Approach to Model Order Reduction of Nonlinear Dynamical Systems*. PhD thesis, MIT, 2003.

Appendix A

Introduction of Differential Geometry

The following introduction is based on notes in [141].

A.1 Vector Fields and Flows

Let X be an open set of $\mathbb{R}^N$. We denote T_pX the space of tangent vectors to X at the point p.

A **vector field** on X is a mapping

$$p \in X \longrightarrow f(p) \in T_pX, \tag{A.1}$$

which assigns a tangent vector at p to any point p in X. In a given coordinate system, f is expressed as a column vector $f = (f_1, \cdots, f_N)^T$.

For any vector field, we can write the differential equation

$$\dot{x} = f(x). \tag{A.2}$$

Then starting from a point p, the solution of (A.2), is denoted by a map $(t, p) \mapsto \gamma_t(p)$. The family γ_t of local maps are called the **flow** of the vector field f. The following properties hold for $\gamma_t(p)$:

$$\gamma_{t_1} \circ \gamma_{t_2} = \gamma_{t_1+t_2}, \quad \gamma_{-t} = (\gamma_t)^{-1}, \quad \gamma_0 = \mathrm{id}. \tag{A.3}$$

If $\forall t, p,\ \gamma_t(p)$ are well-defined, the f is called **complete**, and its flow forms a one-parameter group of diffeomorphisms of X. On the other hand, given any $\gamma_t(p)$, we can define a unique vector field $f(p)$ by

$$f(p) = \left.\frac{\partial}{\partial t}\right|_{t=0} \gamma_t(p). \tag{A.4}$$

We will denote the flow of f by γ_t^f or $\exp(tf)$.

A.2 Lie Bracket and its Properties

The Lie bracket of two vector fields is another vector field which, roughly speaking, measures the non-commutativeness of the flows of both vector fields. Non-commutativeness means the dependence of the result of applying these flows.

There are several definitions (interpretations) of the Lie bracket:

Given a coordinate system, the Lie bracket of two vector fields f and g, denoted by $[f, g]$ is

$$[f, g](x) = \frac{\partial g}{\partial x}(x) f(x) - \frac{\partial f}{\partial x}(x) g(x). \tag{A.5}$$

We will call this the **Jacobian definition of Lie bracket**.

Proposition A.2.1 *The Lie bracket of vector fields f and g is equal identically to zero if and only if their flows commute, i.e.,*

$$[f, g] \equiv 0 \quad \Leftrightarrow \quad \gamma_t^f \circ \gamma_s^g(p) = \gamma_s^g \circ \gamma_t^f(p), \quad \forall s, t \in \mathbb{R}^N, \forall p \in X. \tag{A.6}$$

Two vector fields having the property in (A.6) are called **commuting**.

Proposition A.2.2 *Consider a curve $\alpha(t) = \gamma_{-t}^g \circ \gamma_{-t}^f \circ \gamma_t^g \circ \gamma_t^f(p)$. Then we have that its first derivative at zero vanishes, $\alpha'(0) = 0$, and the second derivative is given by the Lie bracket $\alpha''(0) = 2[f, g](p)$.*

A.3 Coordinate Changes and Lie Bracket

We consider a global diffeomorphism $\Phi : X \to X$. Since tangent vectors are transformed through the Jacobian map of a diffeomorphism, the diffeomorphism defines a transformation of a vector field f

$$\mathrm{Ad}_\Phi(f)(p) = D\Phi(q) f(q), \quad q = \Phi^{-1}(p), \tag{A.7}$$

where $D\Phi$ is the tangent map of Φ which in coordinates, is $\frac{\partial \Phi}{\partial x}$. We also write

$$\Phi_* f = \mathrm{Ad}_\Phi f \tag{A.8}$$

Note that the coordinate change $p = \Phi(q)$ transforms the differential equations $\dot{p} = f(p)$ to $\dot{q} = \mathrm{Ad}_\Phi f(q)$. We also have $\mathrm{Ad}_\Phi(\lambda_1 f_1 + \lambda_2 f_2) = \lambda_1 \mathrm{Ad}_\Phi(f_1) + \lambda_2 \mathrm{Ad}_\Phi(f_2)$, and $\mathrm{Ad}_{\Phi \circ \Psi} = \mathrm{Ad}_\Phi \mathrm{Ad}_\Psi(f)$.

Proposition A.3.1 *Consider the vector field $Ad_\Phi(f)$. The local flow of this vector field is*

$$\sigma_t = \Phi \circ \gamma_t \circ \Phi^{-1}. \tag{A.9}$$

Geometric definition of Lie bracket: we define the Lie bracket of f and g as the derivative with respect t, at $t = 0$, of the vector field g transformed by the flow of f, *i.e.*, we define

$$[f, g](p) = \frac{\partial}{\partial t} D\gamma^f_{-t}(\gamma^f_t(p))g(\gamma^f_t(p)) = \frac{\partial}{\partial t}(\mathrm{Ad}_{\gamma^f_{-t}} g)(p). \tag{A.10}$$

Proposition A.3.2 *If Φ is a diffeomorphism of X, then*

$$[Ad_\Phi f, Ad_\Phi g] = Ad_\Phi [f, g]. \tag{A.11}$$

In the following, we will also use $\mathrm{ad}_f g$ for $[f, g]$, therefore ad_f is a linear operator in the space of vector fields $V(X)$. We also denote

$$\mathrm{ad}^0_f g = g, \quad \mathrm{ad}^i_f g = \mathrm{ad}_f \cdots \mathrm{ad}_f g. \tag{A.12}$$

Proposition A.3.3 *If the vector fields f and g are real analytic, then we have the following expansion formula for the vector field g transformed by the flow of the vector field f:*

$$(Ad_{\gamma^f_t} g)(p) = \sum_{i=0}^{\infty} \frac{(-t)^i}{i!} (ad^i_f g)(p), \tag{A.13}$$

where the series converges absolutely for t in a neighborhood of zero.

A.4 Vector Fields as Differential Operators

A smooth vector field f on X defines a linear operator L_f on the space of smooth functions $C^\infty(X)$ in the following way

$$(L_f \phi)(p) = \frac{\partial}{\partial t}\Big|_{t=0} \phi(\gamma^f_t(p)) = \sum_{i=1}^{n} f_i(p) \frac{\partial}{\partial x_i} \phi(p). \tag{A.14}$$

The operator is called **directional derivative along** f or **Lie derivative along** f.

Any differential operator of order one can be written as

$$L = \sum_{i=1}^{n} a_i(x) \frac{\partial}{\partial x_i}, \tag{A.15}$$

and it defines a unique vector field defined by $f = (a_1(x), \cdots, a_n(x))^T$. Therefore, there is a one-to-one correspondence between the vector field f and the differential operator L_f.

Define the commutator of L_f and L_g by

$$[L_f, L_g] = L_f L_g - L_g L_f. \tag{A.16}$$

We have

$$[L_f, L_g] = L_{[f,g]} \tag{A.17}$$

The Lie bracket also satisfies two important properties, **antisymmetry**, and **Jacobi identity**:

$$\begin{aligned} &[f, g] = -[g, f] \\ &[f, [g, h]] + [g, [h, f]] + [h, [f, g]] = 0 \end{aligned} \tag{A.18}$$

The linear space $V(X)$ of smooth vector fields f and g on X, with the Lie bracket as product, is called the **Lie algebra of vector fields on** X.

A.5 Equivalence between Families of Vector Fields

Consider two general families of analyticvector fields on X and $\bar{X}$, respectively, parameterized by the same parameter $u \in U$, *i.e.*

$$F = \{f_u\}_{u\in U}, \quad \bar{F} = \{\bar{f}_u\}_{u\in U} \tag{A.19}$$

We shall see these points **locally equivalent** at the points p and $\bar{p}$, respectively, if there is a local analytic diffeomorphism $\Phi : X \to \bar{X}, p \mapsto \bar{p}$ which transforms the vector fields f_u into $\bar{f}_u$ locally, *i.e.*, $\mathrm{Ad}_\Phi f_u = \bar{f}_u, \forall u \in U$ locally around $\bar{p}$.

Denote by $\mathcal{L}$ and $\bar{\mathcal{L}}$ the Lie algebras of vector fields generated by the families of F and $\bar{F}$.

A family of vector fields is called **transitive** at a point if its Lie algebra is of full rank at this point, *i.e.*, the vector fields in this Lie algebra span the whole tangent space at this point.

We will use the notation

$$f_{[u_1,\cdots,u_n]} = [f_{u_1}, [f_{u_2}, [\cdots, [f_{u_{k-1}}, f_{u_k}], \cdots,]] \tag{A.20}$$

for any $k \geq 1$, and for any $u_1, \cdots, u_k \in U$. In particular, $f_{[u_1]} = f_{u_1}$.

Definition A.5.1 *A **distribution** on X is a map Δ which assigns to each point in X a subspace of the tangent space at this point, i.e.,*

$$p \in X \to \Delta(p) \in T_pX. \tag{A.21}$$

*The distribution Δ is called of class C^∞ if, locally around each point in X, there is a family of vector fields $\{f_\alpha\}$ (called **local generators** of Δ) which spans Δ, i.e., $\Delta(p) = span_\alpha f_\alpha(p)$.*

*Δ is called **locally generated** if the above family of vector fields is finite.*

Finally, the distribution Δ is called of dimension k, if $dim\Delta(p) = k$ for all points in X.

Definition A.5.2 *A vector field f belongs to a distribution Δ, i.e., $f \in \Delta$, if $f(p) \in \Delta(p)$ for all p in X.*

A distribution Δ is called ***involutive*** *if for any vector fields $f, g \in \Delta$, the Lie bracket is also in Δ, i.e., $[f, g] \in \Delta$.*

Theorem A.5.1 (Frobenius theorem (local)) *If Δ is an involutive distribution of class C^∞ and of dimension k on X, then locally around any point in X, there exists a smooth change of coordinates which transforms the distribution Δ to the following constant distribution span$\{e_1, \cdots, e_k\}$, where $e_i = (0, \cdots, 0, 1, 0, \cdots, 0)^T$ with 1 at the i-th place.*

Proof The proof consists of two steps.

Firstly, we show the distribution Δ is locally generated by k pairwise commuting vector fields. We fix a point p in X and let $F = (f_1, \cdots, f_k) \in \mathbb{R}^{N \times k}$ be any vector fields which generate the distribution Δ in a neighborhood of p. Multiplying F from the right by an invertible $k \times k$ matrix of smooth functions does not change the distribution spanned by the columns of F (it only changes the generators). By a possible permutation of variables we can make the upper $k \times k$ matrix of F to be identity matrix. Therefore, the first k coefficients of the Lie bracket $[f_i, f_j]$ are 0. On the other end, from involutivity, it follows that the Lie bracket is a linear combination of columns of F, and therefore, the Lie bracket has to be 0. This shows that the new vector fields commute.

Secondly, assume that the vector fields $f_1, \cdots, f_k$ generate the distribution Δ locally around p, and they commute. We can then choose $f_{k+1}, \cdots, f_N$ so that $f_1, \cdots, f_N$ are linearly independent at p. Define a map Φ by

$$(t_1, \cdots, t_n) \to \exp(t_1 f_1) \circ \exp(t_2 f_2) \circ \cdots \circ \exp(t_N f_N)(p). \tag{A.22}$$

Therefore, an integral curve of a vector field e_i is transformed to an integral curve of vector field f_i. If follows that the map Φ sends the vector fields $e_1, \cdots, e_k$ to the vector fields $f_1, \cdots, f_k$ and conversely does the inverse map Φ^{-1}. This inverse map is the one that transforms the distribution Δ spanned by $f_1, \cdots, f_k$ to the constant distribution spanned by $e_1, \cdots, e_k$. ■

A.6 Sub-Manifolds and Foliations

Definition A.6.1 *A subset $S \subset X$ is called a* ***regular sub-manifold*** *of X of dimension k if for any $x \in S$ there exists a neighborhood U of x and a diffeomorphism $\Phi : U \to V \subset \mathbb{R}^N$ onto an open subset V such that*

$$\Phi(U \bigcap S) = \{x = (x_1, \cdots, x_N) \in V | x_{k+1} = 0, \cdots, x_N = 0\} \tag{A.23}$$

Definition A.6.2 *We call a subset $S \subset X$ an **immersed sub-manifold** of X of dimension k if*

$$S = \bigcup_{i=1}^{\infty} S_i, \quad \text{where } S_1 \subset S_2 \subset \cdots \subset S, \tag{A.24}$$

and S_i are regular sub-manifolds of X of dimension k.

Proposition A.6.1 *If two vector fields f, g are tangent to an sub-manifold S, then also their Lie bracket $[f, g]$ is tangent to this sub-manifold.*

Definition A.6.3 *A **foliation** $\{S_\alpha\}_{\alpha \in A}$ of X of dimension k is a partition*

$$X = \bigcup_{\alpha \in A} S_\alpha \tag{A.25}$$

*of X into disjoint connected sub-manifolds S_α, called **leaves**, which has the following property: for any $x \in X$ there exists a neighborhood U of x and a diffeomorphism $\Phi : U \to V \subset \mathbb{R}^N$ onto an open subset V such that*

$$\Phi((U \bigcap S_\alpha)_{cc}) = \{x = (x_1, \cdots, x_N) \in V | x_{k+1} = c_\alpha^{k+1}, \cdots, x_N = c_\alpha^N\}, \tag{A.26}$$

where P_{cc} denotes a connected component of the set P.

Proposition A.6.2 *Assume that a vector field g is tangent to a foliation $\{S_\alpha\}_{\alpha \in A}$, i.e., it is tangent to its leaves. Then, if the flow of another vector field f locally preserves this foliation, the Lie bracket $[f, g]$ is tangent to this foliation.*

By saying that the flow of f locally preserves the foliation $\{S_\alpha\}_{\alpha \in A}$, we mean that for any point $p \in S_\alpha$, there is a neighborhood U of p such that the image of a piece of a leaf $\gamma_t^f(S_\alpha \bigcap U)$ is contained in a leaf of the foliation, for any t sufficiently small.

A.7 Orbits of Families of Vector Fields

Consider a family of vector fields $\mathcal{F} = \{f_u\}_{u \in U}$ on X.

Definition A.7.1 *We define the **orbit of a point** $p \in X$ of this family as the set of the points of X reachable from p piecewise by trajectories of vector fields in the family, i.e.,*

$$Orb(p) = \{\gamma_{t_k}^{u_k} \circ \cdots \circ \gamma_{t_1}^{u_1} | k \geq 1, u_1, \cdots, u_k \in U, t_1, \cdots, t_k \in \mathbb{R}\}, \tag{A.27}$$

where γ_t^u denotes the flow of the vector field f_u.

Definition A.7.2 *Let Γ be the smallest distribution on X which contains the vector fields in the family $\mathcal{F}$ (i.e., $f_u(p) \in \Gamma(p)$ for all $u \in U$) and is invariant under any flow γ_t^u, $u \in U$, ie,*

$$D\gamma_t^u(p)\Gamma(p) \subset \Gamma(\gamma_t^u(p)) \tag{A.28}$$

for all $p \in X$, $u \in U$.

Equivalently, we have

$$g \in \Gamma \Longrightarrow \mathrm{Ad}_{\gamma_t^u} g \in \Gamma, \quad \forall u \in U, t \in \mathbb{R}. \tag{A.29}$$

Theorem A.7.1 (Orbit Theorem) *Each orbit $S = Orb(p)$ of a family of vector fields $\mathcal{F} = \{f_u\}_{u\in U}$ is an immersed sub-manifold. Moreover, the tangent space to this sub-manifold is given by the distribution Γ, i.e.,*

$$T_pS = \Gamma(p), \quad \forall p \in X. \tag{A.30}$$

Corollary A.7.2 *If the vector fields f_u are analytic, then the tangent space to the orbit can be computed as*

$$T_pX = L(p) = \{g(p) | g \in Lie\{f_u\}_{u\in U}\}, \tag{A.31}$$

where $Lie\{f_u\}_{u\in U}$ denotes smallest family of vector fields which contains the family $\mathcal{F}$ and is closed under taking linear combinations and Lie bracket. In the smooth case, the following inclusion holds

$$L(p) \subset \Gamma(p) \tag{A.32}$$

A.8 Integrability of Distributions and Foliations

Definition A.8.1 *A distribution of constant dimension $p \to \Delta(p)$ on X is **integrable** if there exists a foliation $\{S_\alpha\}_{\alpha\in A}$ on X such that for any $p \in X$,*

$$T_pS = \Delta(p), \tag{A.33}$$

where S is the leaf passing through p.

Theorem A.8.1 (Global Frobenius theorem) *A smooth distribution of constant dimension Δ is integrable if and only if it is involutive. The integral foliation of Δ is the partition of X into orbits of the family of vector fields $\{g | g \in \Delta\}$.*

Appendix B

Benchmark Examples

This chapter lists a few benchmark examples we have tried. Table B.1 summarizes the MATLAB scripts/codes for each benchmark example. The scripts are to be available on `http://www.eecs.berkeley.edu/~gcj`.

Table B.1: Scripts for benchmarks

Benchmark	Code/Script
NTL1	`hckt_nonlinear_transmission_line_tpwl.m`
NTL2	`hckt_nonlinear_transmission_line_quadratic_tpwl.m`
INVC	`hckt_inverter_chain.m`
NEURON_ML	`hckt_ml_neuron.m`
NEURON_FN	`hckt_fn_neuron.m`
LATCH1	`hckt_latch1.m`

B.1 Nonlinear Transmission Line Circuit 1 (NTL1)

The nonlinear transmission line circuit, shown in Fig. B.1 [2], is an example that has been tested in lots of MOR literature. All resistors and capacitors are set to 1 and the diode I-V characteristic is $i_D = e^{40v_D} - 1$. The input is the current source $i = u(t)$; the output is the voltage at node 1.

The modified nodal equations for this circuit are

$$\begin{aligned}
\dot{v_1} &= -2v_1 + v_2 + 2 - e^{40v_1} - e^{40(v_1 - v_2)} + u(t), \\
\dot{v_i} &= -2v_i + v_{i-1} + v_{i+1} + e^{40(v_{i-1} - v_i)} - e^{40(v_i - v_{i+1})}, \quad 2 \le i \le N-1, \\
\dot{v_N} &= -v_N + v_{N-1} - 1 + e^{40(v_{N-1} - v_N)}.
\end{aligned} \tag{B.1}$$

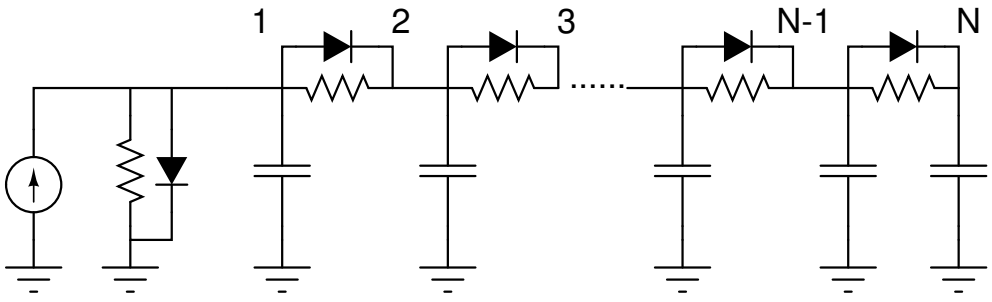


Figure B.1: Nonlinear transmission line circuit.

B.2 Nonlinear Transmission Line 2 (NTL2)

The circuit was tested in papers on the TPWL method [2] [142]. The circuit diagram is shown in Fig. B.2, where the resistances and capacitances are all set to 1, and the nonlinear resistor at each stage has the $I-V$ characteristic $i_n(v) = \text{sgn}(v)v^2$. The input is the current source $i = u(t)$; the output is the voltage at node 1.

The differential equations are

$$\begin{aligned} \dot{v_1} &= -2v_1 + v_2 - \text{sgn}(v_1)v_1^2 + u(t), \\ \dot{v_i} &= -2v_i + v_{i-1} + v_{i+1} - \text{sgn}(v_i)v_i^2, 2 \le i \le N-1, \\ \dot{v_N} &= -v_N + v_{N-1} - \text{sgn}(v_N)v_N^2. \end{aligned} \tag{B.2}$$

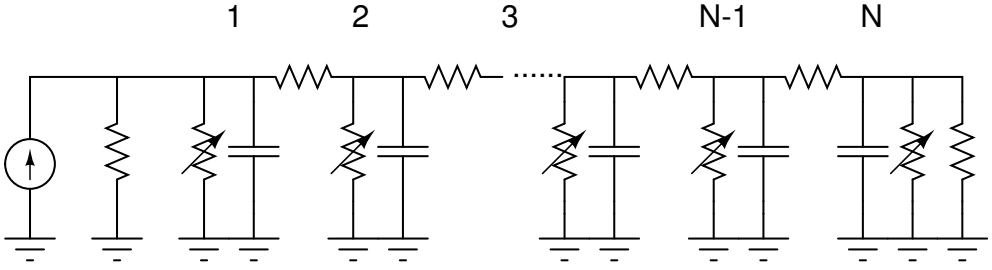


Figure B.2: Nonlinear transmission line (quadratic nonlinearity).

B.3 Inverter Chain Circuit (INVC)

The inverter chain circuit is shown in Fig. B.3. The resistances and capacitances are all set to 1. The inverter $V_{out} - V_{in}$ characteristic is

$$V_{out} = f_{inv}(V_{in}) = V_{dd} \tanh(-AV_{in}), \tag{B.3}$$

where V_{dd} is the supply voltage and A is a parameter. We choose $V_{dd} = 1$ and $A = 5$. The input is the voltage source $u(t)$ and the output is the node voltage at node N.

The differential equations are

$$\begin{aligned} &\dot{x_1} + x_1 - u(t) = 0 \\ &\dot{x_i} + x_i - f_{inv}(x_{i-1}) = 0, \quad 2 \le i \le N. \end{aligned} \tag{B.4}$$

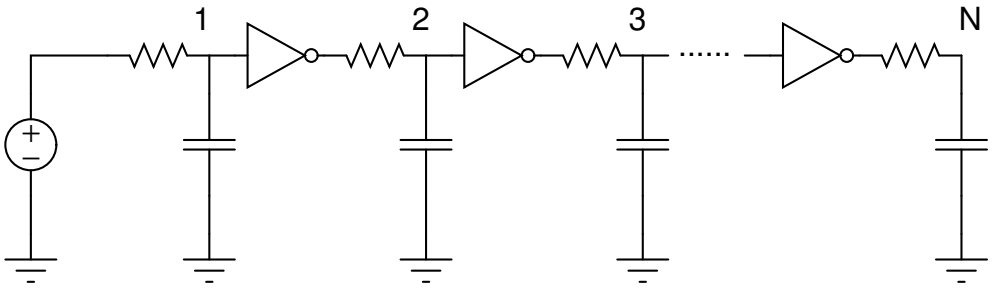


Figure B.3: Inverter chain.

B.4 Morris-Lecar Neuron Model (NEURON_ML)

The Morris-Lecar model [134] is one neuron model that describes potential dynamics of firing neurons. The input is the injection current I.

The differential equations are

$$\begin{aligned} \frac{d}{dt}V =& \frac{1}{C_M}\left(-g_L(V - V_L) - g_{Ca}M_\infty(V - V_{Ca}) - g_K N(V - V_K) + I\right) \\ \frac{d}{dt}N =& (N_\infty - N)\frac{1}{\tau_N}, \end{aligned} \tag{B.5}$$

where $M_\infty = 0.5(1+\tanh((V-V_1)/V_2), N_\infty = 0.5(1+\tanh((V-V_3)/V_4), \tau_N = 1/(\Phi\cosh((V-V_3)/(2V_4)))$. The input is the injection current I, and other parameters are chosen as $C_M = 20, g_K = 8, g_L = 2, V_{Ca} = 120, V_K = -80, V_L = -60, V_1 = -1.2, V_2 = 18, V_4 = 17.4, g_{Ca} = 4, \Phi = 1/15, V_3 = 12$, which are adapted from [134].

B.5 FitzHugh-Nagumo Model (NEURON_FN)

The FitzHugh-Nagumo model is a PDE neuron model. The input is the injection current $i_0(t)$ which is included in the boundary condition.

Let $x \in [0, L], t \geq 0$, the PDEs are

$$\begin{aligned} \frac{\partial}{\partial t}v(x,t) &= \epsilon\frac{\partial^2}{\partial x^2}v(x,t) + \frac{1}{\epsilon}\left(f(v(x,t)) - w(x,t) + c\right) \\ \frac{\partial}{\partial t}w(x,t) &= bv(x,t) - \gamma w(x,t) + c \end{aligned} \tag{B.6}$$

with

$$f(v) = v(v - 0.1)(1 - v), \tag{B.7}$$

boundary conditions

$$\frac{\partial}{\partial x}v(0,t) = -i_0(t), \quad \frac{\partial}{\partial x}v(L,t) = 0 \tag{B.8}$$

and initial conditions

$$v(x,0) = 0, \quad w(x,0) = 0, \tag{B.9}$$

where the parameters are

$$L = 1, \quad \epsilon = 0.015, \quad b = 0.5, \quad \gamma = 2, \quad c = 0.05. \tag{B.10}$$

Applying finite difference on x-dimension, let

$$v_i(t) = v(ih), \quad w_i(t) = w(ih), \quad i = 1, \cdots, N \tag{B.11}$$

where $h = 1/(N + 1)$.

We obtain

$$\begin{aligned} \frac{\partial}{\partial t}v_i(t) &= \epsilon\frac{v_{i+1} - 2v_i + v_{i-1}}{h^2} + \frac{1}{\epsilon}\left(f(v_i(t)) - w_i(t) + c)\right), \quad i = 2, \cdots, N-1, \\ \frac{\partial}{\partial t}v_1(t) &= \epsilon\frac{1}{h}\left(\frac{v_2 - v_1}{h} + i_0(t)\right) + \frac{1}{\epsilon}\left(f(v_1(t)) - w_1(t) + c)\right), \\ \frac{\partial}{\partial t}v_N(t) &= \epsilon\frac{1}{h}\left(-\frac{v_N - v_{N-1}}{h}\right) + \frac{1}{\epsilon}\left(f(v_N(t)) - w_N(t) + c)\right), \\ \frac{\partial}{\partial t}w_i(t) &= bv_i(t) - \gamma w_i(t) + c, \quad i = 1, \cdots, N. \end{aligned} \tag{B.12}$$

B.6 Latch Circuit 1 (LATCH1)

The circuit diagram of the latch is shown in Fig. B.4. The latch is composed of a sampler (consisting of a multiplexer) and a regenerator (consisting of two cross-coupled inverters). Each output of the multiplexer and inverters is followed by an RC circuit that is not shown in Fig. B.4. The latch samples the $DATA$ input when CLK is "1". When CLK is "0", two inverters are cross-coupled, and re-generate the output (QBB).

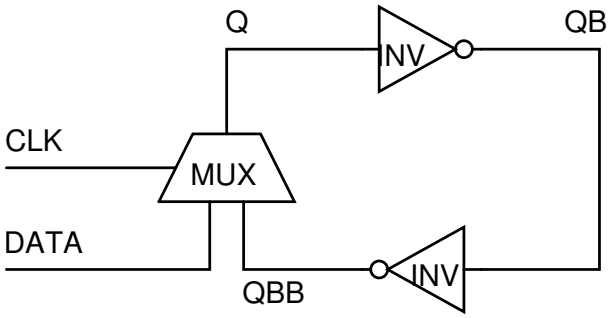


Figure B.4: A simplified latch model.

The differential equations are

$$\begin{aligned} RC\frac{dx_1}{dt} + (x_1 - f_{MUX}(u_1, u_2, x_3)) &= 0, \\ RC\frac{dx_2}{dt} + (x_2 - f_{INV}(x_1)) &= 0, \\ RC\frac{dx_3}{dt} + (x_3 - f_{INV}(x_2)) &= 0, \end{aligned} \tag{B.13}$$

where $C = 1$, $R = 1/3$,state variables x_1, x_2, x_3 are node voltages v_Q, v_{QB}, v_{QBB}, respectively; inputs u_1, u_2 are CLK and $DATA$, respectively; f_{INV} and f_{MUX} are I/O functions defined by

$$\begin{aligned} f_{INV}(x) &= -\tanh(10x), \\ f_{MUX}(C, A, B) &= \frac{A+B}{2} + \frac{B-A}{2}\tanh(10C), \end{aligned} \tag{B.14}$$

for the inverter and the multiplexer shown in Fig. B.5.

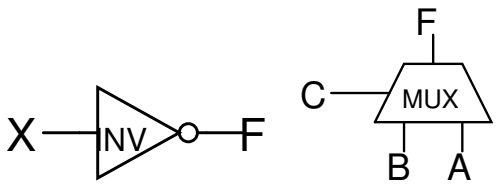


Figure B.5: An inverter and a multiplexer.