Foundations and Trends® in
Electronic Design Automation
Vol. 3, Nos. 2–3 (2008) 97–303

DOI: 10.1561/1000000009

now
the essence of knowledge

Numerical Simulation and Modelling of Electronic and Biochemical Systems

By Jaijeet Roychowdhury

Contents

Foundations and Trends® in
Electronic Design Automation
Vol. 3, Nos. 2–3 (2008) 97–303

DOI: 10.1561/1000000009

now
the essence of knowledge

Numerical Simulation and Modelling of Electronic and Biochemical Systems

Jaijeet Roychowdhury

University of California, Berkeley, CA 94720, USA, jr@eecs.berkeley.edu

Abstract

Numerical simulation and modelling are witnessing a resurgence. Designing systems with integrated wireless components, mixed-signal blocks and nanoscale, multi-GHz "digital" circuits is requiring extensive low-level modelling and simulation. Analysis and design in non-electronic domains, notably in systems biology, are also relying increasingly on numerical computation.

Sections 2–8 of this Monograph provide an introduction to the fundamentals of numerical simulation, and to the basics of modelling electronic circuits and biochemical reactions. The focus is on a minimal set of concepts that will enable the reader to further explore the field independently. Differential–algebraic equation models of electronic circuits and biochemical reactions, together with basic numerical techniques — quiescent, transient and linear frequency domain analyses, as well as sensitivity and noise analyses — for solving these differential equations are developed. Downloadable MATLAB implementations are provided.

The last two chapters provide an introduction to computational methods for nonlinear periodic steady states and multi-time partial

differential equation (PDE) formulations, followed by an overview of model order reduction (MOR) and, at the end, a glimpse of some applications of oscillator MOR — in circuits (PLLs), biochemical reaction–diffusion systems and nanoelectronics.

1

Introduction

1.1 Trends in Numerical Simulation and Applications

Since the advent of integrated circuits (ICs), numerical modelling and simulation have played an important rôle in analog, mixed-signal and RF design. Simulation is typically used to verify correctness and debug circuits during their design. Over the years, computer simulation has almost entirely replaced breadboarding and physical prototyping, especially for IC designs, where fabrication is expensive and time-consuming, and probing internal nodes difficult. Being able to "run" a circuit in simulation makes it much more likely to function correctly when it is actually built.

As a discipline, circuit simulation emerged in the late 1960s and early 1970s, with early programs like CANCER [70] maturing into design tools such as ASTAP [118] and SPICE [69, 89, 90]. These programs became popular during the chip design boom of the 1970s and enabled the design of new generations of ICs. After the early 1980s, simulation — the first electronic CAD discipline and arguably the progenitor of the electronic design automation (EDA) industry — became supplanted by other areas of EDA, such as physical design, combinatorial and sequential synthesis, formal verification, model

checking, *etc.* However, the past decade has witnessed a resurgence in simulation and related areas (such as model-order reduction).

This resurgence has been spurred by several factors. Ever-shrinking DSM technologies have resulted in analog and digital designs both becoming markedly less ideal. Indeed, at lower levels of design, the distinction has become blurred, with continuous/analog effects being pervasive and causing digital abstractions to break down. This trend started almost three decades ago, when delay and crosstalk started becoming important concerns in digital design, first at the gate level and then in interconnect; today, huge chip sizes and extensive system-level integration — e.g., in systems-on-chip (SoCs) and systems-in-packages (SiPs) — have made interference and noise crucial limiting factors in virtually all digital and mixed-signal designs. Moreover, as feature sizes have evolved toward the 22 nm node over the last decade, low-level non-ideality in transistor characteristics has emerged as another serious design issue. An important aspect of this non-ideality is greatly increased parameter variability. These effects are all continuous, or analog, in nature, hence call for low-level simulation and modeling tools during design and verification.

Integrated RF and communication system design, which since the mid-1990s has constituted an important part of the semiconductor industry's portfolio, has been another driver for numerical simulation and modelling. Such designs involve analog/mixed-signal, RF and digital components on the same substrate.

They also frequently involve micro/nanoelectromechanical system (MEMS/NEMS) elements and electromagnetic (EM) structures, such as antennas and transmission lines; and, to a lesser extent, optical and even biological elements. These trends have increased reliance on simulation technologies, not only during the design of individual circuits, but for the verification of larger systems. Indeed, interactions between analog/mixed-signal, RF, non-electronic and digital systems are a prolific source of design problems and functional failures; an important use of low-level simulation and modelling tools is to help debug such problems.

It is for these reasons that numerical simulation and modelling have been growing in importance and seeing steadily increasing practical

application. The proliferation of physical domains for which simulation technologies are now needed, compounded by generally increased complexity, has expanded the scope of numerical simulation and modelling within CAD and spurred new research directions. For example, interconnect issues drove research in EM simulation and extraction, and in linear model-order reduction (MOR), through the 1990s. Efficient algorithms for periodic and stochastic (noise) analysis of large nonlinear circuits and systems, developed in the mid to late 1990s, were driven by the integrated RF ICs which fueled the portable wireless revolution. Nonlinear stochastic simulation, currently a topic of active research, is motivated by the parameter variability problem. Automated nonlinear computational macromodelling, or nonlinear MOR, is similarly a topic of current interest because it enables system level verification and design. The rise of multi-core computational platforms has led to parallelization of numerical simulation algorithms becoming yet another topic of current interest.

Another exciting driver for research in the above areas is biological systems. The past decade has witnessed tremendous progress in mathematical biology; biology has been transforming into a precise, quantitative, bottom-up, predictive discipline, much as physics and engineering did over the last century. Even more than in electronics, biological systems feature many individual entities that interact extensively and are organized hierarchically, with logical functionality emerging from the hybrid interplay of discrete and continuous-time dynamics; and to a much greater extent than the human-engineered systems of today, understanding how most biological systems work — even qualitatively — is often simply not possible at all without computational tools. Similar circumstances led to CAD tools becoming indispensable in electronics. The same is happening in the biological arena; effective leveraging of computational techniques from electronics and engineering systems can catalyze progress in the field.

1.2 Scope and Organization of This Monograph

This Monograph provides an introduction to the fundamentals of numerical simulation, and to the basics of modelling electronic circuits

and biochemical reactions. The emphasis is on capturing a minimal set of important concepts succinctly, but concretely enough that the reader will be left with an adequate foundation for further independent exploration. Starting from mathematical models of basic electronic elements, circuits are modeled as nonlinear differential–algebraic equation (DAE) systems. Two basic techniques — quiescent steady state and transient — for solving these differential equations systems are then developed. It is then shown how biochemical reactions can also be modeled deterministically as DAEs (hence simulated using the various numerical simulation methods developed of this Monograph). Following this, frequency domain techniques for finding sinusoidal steady states of linear DAEs are developed, as are direct and adjoint techniques for computing parameter sensitivities and the effects of stationary random noise.

For readers interested in a glimpse of topics beyond these basics, an introduction to nonlinear periodic steady state methods (harmonic balance and shooting) and the multitime partial differential equation formulation is provided. Also provided is an overview of model order reduction, an important topic of current research that has roots in numerical simulation algorithms. Finally, sample applications of nonlinear oscillator macromodels — in circuits (PLLs), biochemical reaction–diffusion systems and nanoelectronics — are presented.

1.3 Website for MATLAB Scripts and Updates

The reader is encouraged to visit the following URL, which contains MATLAB implementations illustrating topics in this Monograph:

http://www.eecs.berkeley.edu/∼jr/NOW-FT-Monograph/.

The site also contains addenda and updates.

1.4 Acknowledgments

The basic material in the earlier chapters of this Monograph is strongly influenced by [7]. The author would like to acknowledge a number of distinguished colleagues for contributing materially to his knowledge of

the topics in this Monograph: Raj Raghuram, Bob Brayton, Peter Feldmann, Alper Demir, Amit Mehrotra, Joel Phillips, Onuttom Narayan, Hans-Georg Brachtendorf, David Long, Bob Melville and Jacob White. Much of the material on applications of nonlinear oscillator macromodels is the work of Xiaolue Lai; results from piecewise polynomial trajectory-based nonlinear macromodelling are the work of Ning Dong. The author thanks Sharad Malik and James Finlay for their patience and helpfulness during the development of this Monograph.

2

Circuit Element Equations ("Device Models")

To set up equations for a circuit, one needs to first understand the equations for the building blocks, i.e., the basic elements of circuits. These equations, which describe the behavior of each element quantitatively, are sometimes termed *Branch Constitutive Relationships* or BCRs.

2.1 Review: Linearity, Time-invariance, Memorylessness, Causality

Before looking at even the simplest circuit elements, it is helpful to review the important concepts of *linearity*, time invariance and *memorylessness*; and also *causality*.

The fundamental framework for any of the above concepts is the *input-output system*. That is to say, we have a system which accepts an *input* $x(t)$ and uses it to produce an *output* $y(t)$. [For our purposes, $x(t)$ and $y(t)$ will usually be real or complex numbers, or vectors of real/complex numbers. t, representing time, is a scalar real number.]

We express the system by means of the *operator* $\mathcal{L}\{\ \cdot\ \}$, i.e.,

$$y(t) = \mathcal{L}\big\{x(t)\big\}. \tag{2.1}$$

The use of the operator notation (as indicated by the curly braces as opposed to parentheses, and the calligraphic symbol $\mathcal{L}$) signifies that the system uses the *entire function* of time $x(t)$ to determine the entire output $y(t)$. Contrast this with the notation

$$y(t) = f\big(x(t)\big), \tag{2.2}$$

which denotes something quite different. What Equation (2.2) means is: "First, pick some value for t of your choice, such as $t = 0.6$. Then find the value of the input waveform $x(t)$ at that t, say $a = x(t = 0.6)$. The system represented by $f(\cdot)$ will produce an output that depends on the value a, i.e., $b = f(a)$. This value b is the value of the output waveform at the chosen time $t = 0.6$; hence $y(t = 0.6) = b$. Repeat this process for every t you are interested in." In other words, for the system (Equation (2.2)), the output $y(t)$ at any time t^* depends *only* on the value of the input waveform $x(t)$ at the *same* t^*; and not on the value of the input at any other times t. Such systems are called **memoryless**.

In contrast, the operator form (Equation (2.1)) implies that the value of $y(t^*)$ is determined, potentially, by the input waveform $x(t)$ at *every* t, not just by the value at $t = t^*$.

A system such as Equation (2.1) is called **linear** if it satisfies two conditions:

- Scaling the input $x(t)$ by any constant number α (i.e., α does not change with time) results in a scaling of the output $y(t)$ also by α. In other words, $\mathcal{L}\{\alpha x(t)\} = \alpha\mathcal{L}\{x(t)\} = \alpha y(t)$.
- The addition of two inputs produces the addition of the individual outputs. In other words, if $y_1(t) = \mathcal{L}\{x_1(t)\}$ and $y_2(t) = \mathcal{L}\{x_2(t)\}$, then $\mathcal{L}\{x_1(t) + x_2(t)\} = y_1(t) + y_2(t)$.

Simple though these conditions are, the concept of linearity is an extremely important and fruitful one in all of mathematics, science and engineering.

An input–output system is said to be **time invariant** if any (constant) time shift in the input results in the output being time-shifted identically. In other words, for *any* constant time shift τ,

the relationship:

$$y(t-\tau) = \mathcal{L}\{x(t-\tau)\} \tag{2.3}$$

must hold for the system to be time invariant. Note that memoryless systems are always time invariant, and that linearity and time invariance are independent concepts — neither one implies the other. Systems that are both linear and time invariant – i.e., linear time invariant, or LTI, systems — are a particularly common, useful and well-studied class, with many powerful results established that are leveraged for time-varying and nonlinear systems as well.

Finally, a system such as Equation (2.1) is said to be **causal** if, for any t^*, the value of the output $y(t^*)$ depends only on input values at $x(t \leq t^*)$. In other words, the output at any given time cannot depend on values of the input beyond that time, i.e., on future inputs.

2.2 Linear Resistor

A resistor is a two-terminal element which simply generates a current in response to a voltage applied across the terminals, according to *Ohm's Law*:

$$i(t) = \frac{v(t)}{R}, \quad \text{or } v(t) = i(t)R. \tag{2.4}$$

The fixed quantity R is called the *resistance*. When the resistance is a fixed constant, independent of v or i, then the resistance is *linear*.

The symbol for the resistor and a plot of the current through it, as a function of the voltage across, are shown in Figure 2.1. Observe that the characteristic curve is simply a straight line with slope equal to the conductance $G = \frac{1}{R}$.

Note that Equation (2.4) provides the instantaneous current through the resistor at any given time t; it says that the current of the resistor at a given time t depends only on the voltage *at the same time*. As we noted above, elements with this property are called *memoryless* or *algebraic*.

Note also that Equation (2.4) is a *mathematical idealization* of a real resistor. No physical resistor is likely to obey Equation (2.4) when, e.g., v is a billion volts. The process of coming up with more accurate

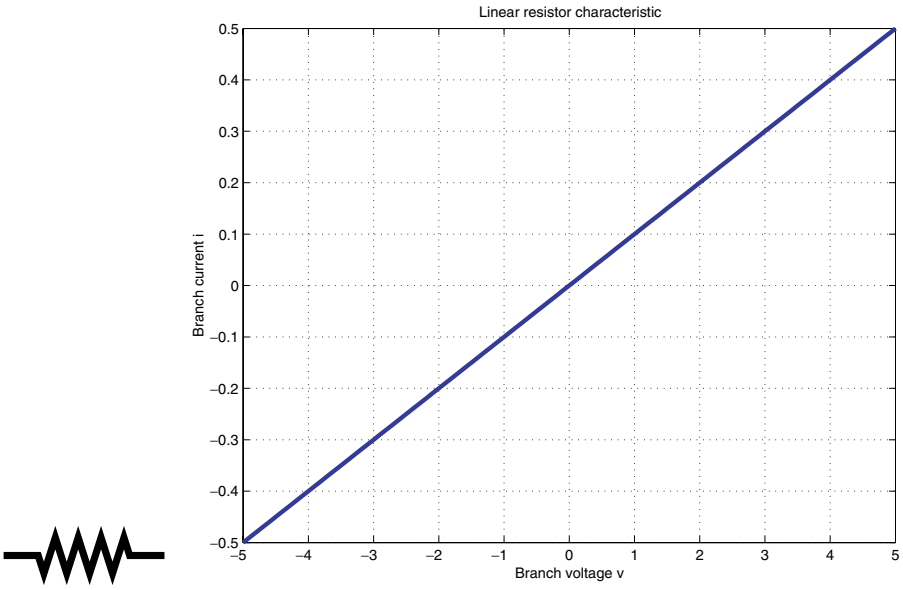


Fig. 2.1 Linear resistor: symbol and i–v characteristic.

equations that better represent reality is known as *modelling* — a field of activity in its own right.

2.3 Linear Capacitor

In contrast to a resistor, which generates an electric current when a voltage is applied, a capacitor builds a *charge* in response to applied voltage.

In a linear capacitor, the charge is linearly related to the voltage:

$$q(t) = Cv(t). \tag{2.5}$$

The fixed quantity C is called the *capacitance*.

As we will see later, we are usually interested in obtaining the current through an element, not just other quantities like charge. The current produced by the capacitor is simply the rate of change of the charge on it, i.e.,

$$i(t) = \frac{dq(t)}{dt}. \tag{2.6}$$

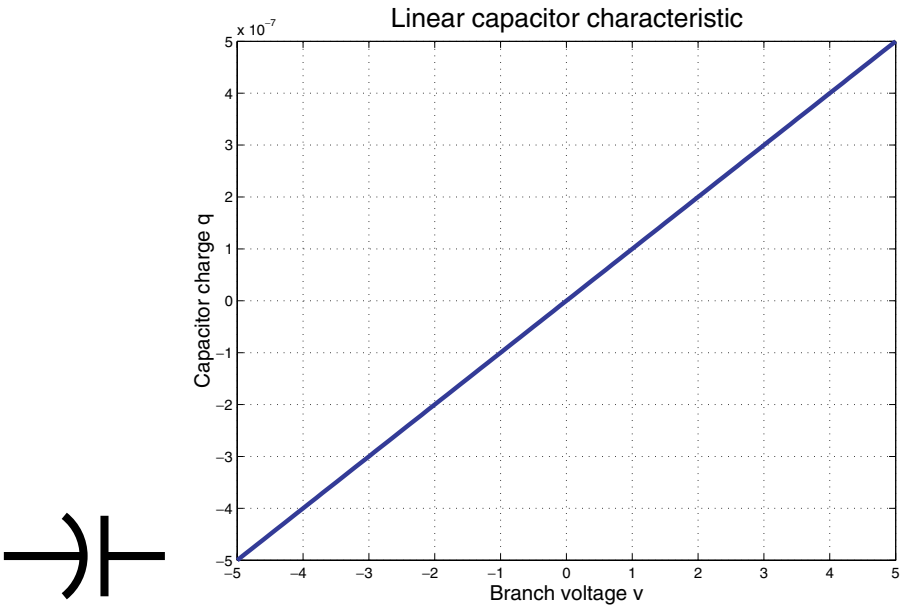


Fig. 2.2 Linear capacitor: symbol and q–v characteristic.

For a linear capacitor therefore, we have

$$i(t) = \frac{dCv(t)}{dt} = C\frac{dv(t)}{dt}. \tag{2.7}$$

The symbol for the capacitor and a plot of the charge through it, as a function of the voltage across, are shown in Figure 2.2. Observe that the charge-voltage characteristic curve is simply a straight line with slope equal to the capacitance C.

Note also that as far as the i–v relationship of a capacitor is concerned, it is not a memoryless element.

2.4 Linear Inductor

An inductor's equations are very similar to those of a capacitor, except that "input" and "output" are exchanged.

If a current i is sent through an inductor, it generates a *magnetic flux* ϕ in response.

In a linear inductor, the flux is linearly related to the current:

$$\phi(t) = Li(t). \tag{2.8}$$

The fixed quantity L is called the *inductance*.

We are interested primarily, of course, in relating the current to the voltage across the inductor's terminals. It is the rate of change of flux that determines the voltage (Faraday's Law):

$$v(t) = \frac{d\phi(t)}{dt}. \tag{2.9}$$

For a linear inductor, therefore, we have

$$v(t) = \frac{d}{dt}Li(t) = L\frac{di(t)}{dt}. \tag{2.10}$$

The symbol of an inductor and a plot of the flux in it, as a function of the current through it, are shown in Figure 2.3. Observe that the flux–current characteristic curve is simply a straight line with slope equal to the inductance L.

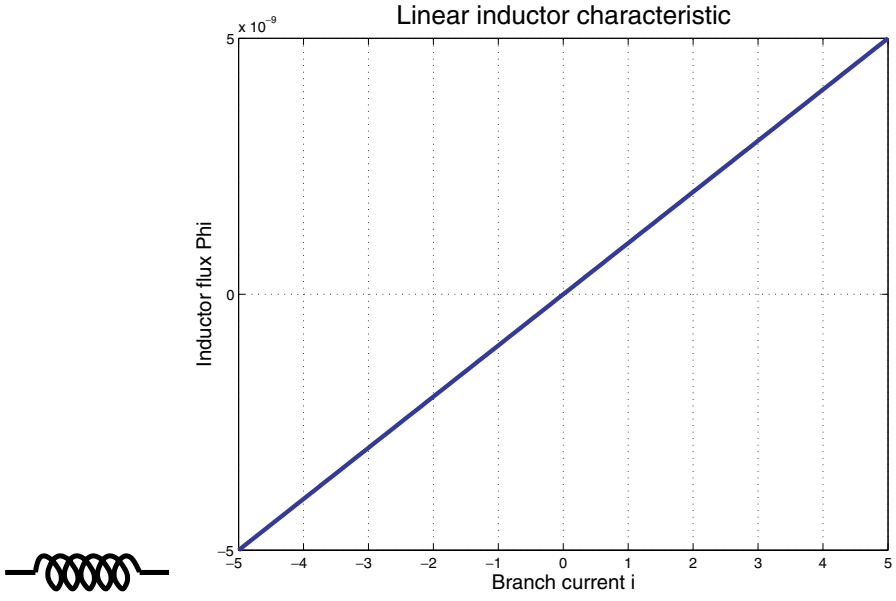


Fig. 2.3 Linear inductor: symbol and ϕ–i characteristic.

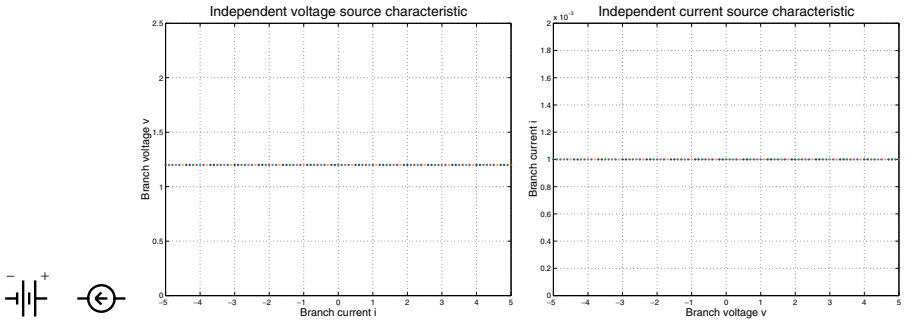


Fig. 2.4 Independent sources: symbols and characteristic curves.

2.5 Independent Voltage and Current Sources

Figure 2.4 depicts the symbols, and characteristic curves, of voltage and current sources, respectively. The constitutive relationship of a voltage source is given simply by

$$v(t) = E(t), \tag{2.11}$$

where $E(t)$ is a known, given waveform. For a DC voltage source of value 1V, for example, $E(t) \equiv 1$; for a sinusoidal source of amplitude 2 and frequency 1kHz, $E(t) = 2\sin(2\pi 1000t)$, and so on.

Observe that the current through the voltage source is not involved at all in its BCR. This simply means that the branch voltage $v(t)$ is independent of the current going through the voltage source; or, in other words, that an ideal voltage source can support any current. When plotting the characteristic curve of a voltage source, therefore, it becomes very important to correctly identify what quantity is the "input" of the element, and what is its output. (Recall that the input can be specified, whereas the output is determined by the input). Note that, for a voltage source, the *current* must be specified as the "input"; the voltage, determined by Equation (2.11) is the "output". Doing the opposite does not make very much sense, since it leads to a contradiction: you cannot specify any voltage input different from the voltage source's own; furthermore, you don't have any equation for the output current.

In other words, for a voltage source, the "output" $v(t)$ is always uniquely defined as a function (though trivial) of all values of the "input" $i(t)$; but one cannot express $i(t)$ as any well-defined output function of $v(t)$.

A current source is the dual of the voltage source — it can support any "input" voltage, but its "output" current is pre-determined, independent of the voltage applied. Its BCR is:

$$i(t) = I(t), \tag{2.12}$$

where $I(t)$ can be any given waveform.

2.6 Linear Controlled Sources

So far, we have been dealing with two-terminal elements, i.e., with two connecting nodes, and a current that flows from one to the other.

Another convenient and useful circuit abstraction is the controlled voltage or current source. These elements consist of two pairs of nodes: the *controlling nodes*, where an input voltage or current is applied; and *controlled or output nodes*, which develop a voltage or current across them in response to the the input.

For example, Figure 2.5 depicts the symbol of a *voltage-controlled voltage source*, or VCVS. The BCRs of this element are

$$\begin{aligned} v_o(t) &= K v_i(t), \\ i_i(t) &\equiv 0. \end{aligned} \tag{2.13}$$

K is a constant, called the *voltage gain*. Note that the VCVS is simply a voltage source across the output nodes, but that the voltage generated is not independent, depending instead on the voltage across two other nodes in the circuit (i.e., across the controlling nodes).

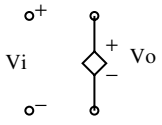


Fig. 2.5 Voltage-controlled voltage source: symbol.

2.7 Generic Two-Terminal Nonlinear Resistor

Any two-terminal "algebraic" element where the current–voltage relationship is *not* linear is termed a nonlinear resistor. In other words, the "output" current through the element as a function of the "input" voltage is given by

$$i(t) = f(v(t)), \tag{2.14}$$

where $f(\cdot)$ can be any well-defined nonlinear function. Equation (2.14) is a voltage-controlled nonlinear resistor; similarly, one can also have a current-controlled nonlinear resistor:

$$v(t) = f(i(t)), \tag{2.15}$$

where the input is $i(t)$ and the output is $v(t)$. Note that if $f(\cdot)$ is invertible, the nonlinear resistor is both voltage and current controlled.

2.8 Nonlinear Capacitors and Inductors

Nonlinear capacitors and inductors are similar in principle to nonlinear resistors, except that the nonlinearities are for the $q - v$ or $\phi - v$ characteristics, respectively:

$$\begin{aligned} q(v(t)) &= f\,(v(t)) \quad \text{(nonlinear voltage-controlled capacitor)}, \\ \phi(v(t)) &= f\,(i(t)) \quad \text{(nonlinear current-controlled inductor)}. \end{aligned} \tag{2.16}$$

Note that Equations (2.6) and (2.9), which relate q to i and ϕ to v, respectively, remain unchanged.

2.9 Diode

The basic diode, the simplest semiconductor device, can be modeled as a voltage-controlled nonlinear resistor:

$$I_D = I_s \left(e^{\frac{V_D}{V_T}} - 1 \right), \quad \text{where } V_T = \frac{kT}{q} \simeq 26mV, \quad I_s \simeq 10^{-12}. \tag{2.17}$$

In Equation (2.17), V_D and I_D are the voltage across and current through the diode, respectively; I_s is a constant called the *saturation*

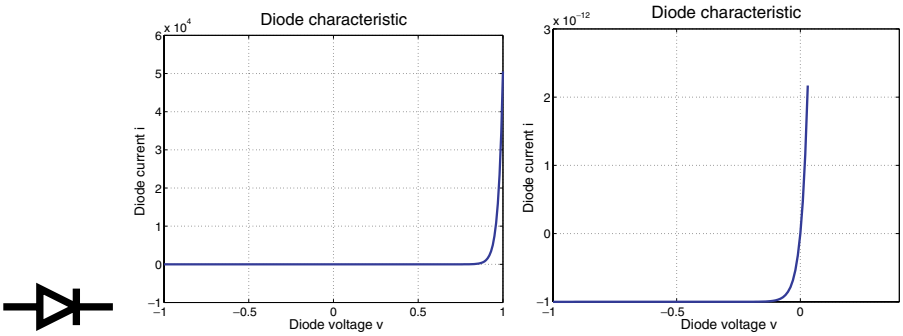


Fig. 2.6 Diode: symbol and i–v curves.

current; $V_T = \frac{kT}{q}$ is another (temperature-dependent) constant called the *thermal voltage*, where k is Boltzmann's constant, q the electronic charge, and T the absolute temperature. Figure 2.6 shows characteristic curves of the diode.

2.10 Ebers-Moll Bipolar Junction Transistor (BJT)

The bipolar junction transistor (BJT) is a semiconductor device widely used in analog and digital circuits.

A BJT has three terminals: emitter (E), base (B) and collector (C). The equivalent circuit of the Ebers–Moll model of a BJT is shown in Figure 2.7. As can be seen, it consists of two back-to-back diodes and two linear current-controlled current sources.

The currents through the two diodes are given by the diode equation (2.17); however, note that the parameters I_s and V_T are typically different for the two diodes. Let I_{bF} and I_{bR} denote the currents through the diodes (along the direction of forward conduction). The currents through the linear CCCSs are given by

$$\begin{aligned} I_{\mathrm{cF}} &= \alpha_F I_{\mathrm{bF}}, \\ I_{\mathrm{cR}} &= \alpha_R I_{\mathrm{bR}}, \end{aligned} \tag{2.18}$$

where $0 < \alpha_F < 1$ and $0 < \alpha_R < 1$ are constants. Typically, α_F is very close to 1 (e.g., $\alpha_F = 0.99$), while $\alpha_R \simeq 0.5$.

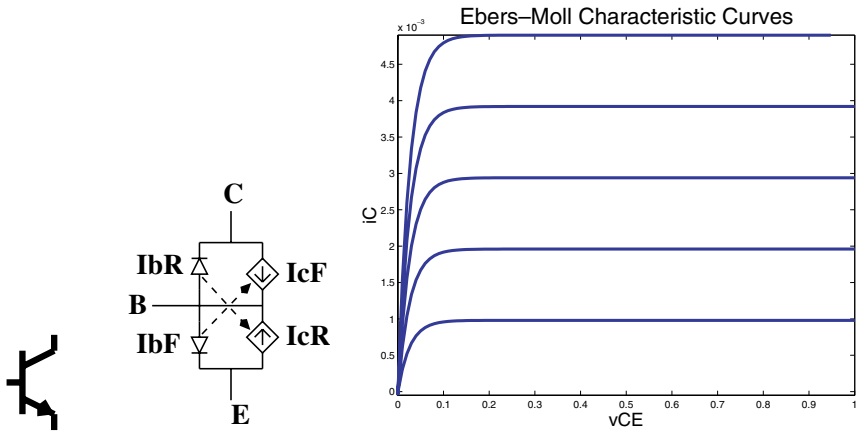


Fig. 2.7 BJT: symbol, Ebers–Moll equivalent circuit and i–v curves.

2.11 Schichman–Hodges MOSFET

Metal-oxide field-effect transistors (MOSFETs) are possibly the most important semiconductor device today, widely used in the entire gamut of digital, analog and RF applications. Figure 2.8 depicts the symbol of a four-terminal MOSFET and typical characteristic curves.

Given their importance, it is not surprising that an enormous amount of effort has been expended in devising accurate mathematical models of MOSFETs, with the result that many MOSFET models are available — MOS1, MOS2, MOS3, ASIM3, BSIM3, BSIM4, Philips MOS9, HISIM, and PSP are all models for MOSFETs, most comprising thousands of lines of code. A classic model for these devices (and possibly the simplest one) is the Schichman–Hodges (S–H) model; it forms the basis for the SPICE leve1 1 (MOS1) model. The S-H model is a 3-terminal model; i.e., voltage changes at the bulk/substrate node are not modeled. The equations for the S–H model are

$$I_D = \begin{cases} \beta\left[(V_{GS}-V_T)-\frac{V_{DS}}{2}\right]V_{DS}, & \text{if } 0 < V_{DS} < V_{GS}-V_T \\ & \quad \text{(triode region)}, \\ \frac{\beta}{2}(V_{GS}-V_T)^2, & \text{if } V_{DS} \geq V_{GS}-V_T > 0 \\ & \quad \text{(saturation or active region)}, \\ 0, & \text{if } V_{GS}-V_T <= 0 \text{ (cutoff)}. \end{cases} \tag{2.19}$$

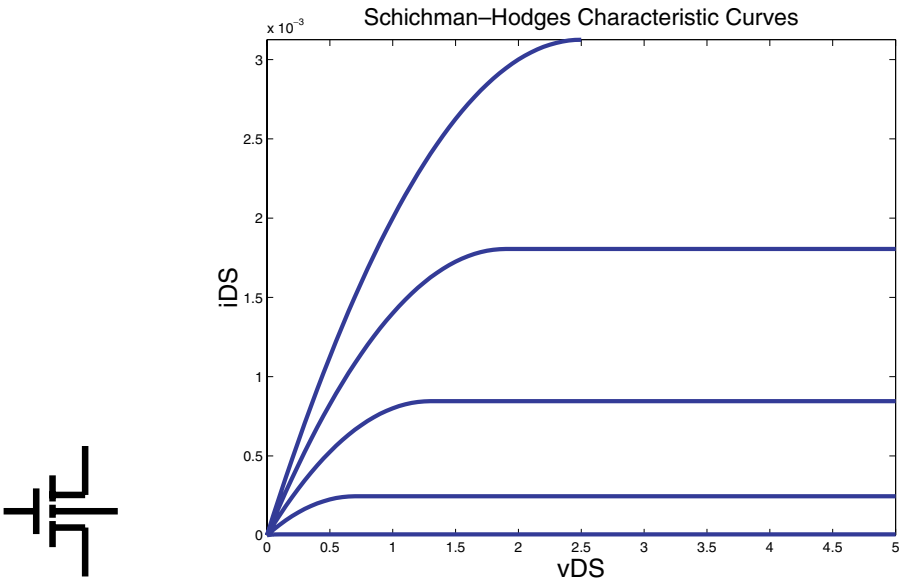


Fig. 2.8 MOSFET: symbol and i–v curves.

(In the above, it is assumed that $V_{DS} \geq 0$; otherwise, the source and drain nodes are interchanged.)

In Equation (2.19), V_T is the threshold voltage. β is a constant that depends on the MOSFET's dimensions:

$$\beta = k_p \frac{W}{L}, \tag{2.20}$$

where W and L are the width and length of the MOSFET, respectively, and k_p is a process-dependent constant.

Equation (2.19), as it stands above, is the most elementary model for the MOSFET that is useful — it is the basis of the MOS Level 1 model[1] available in many simulators, including Berkeley SPICE. This model is unable to capture many second-order effects important in design, such as short-channel effects, bulk effects, *etc.* Far more complex models (such as BSIM3, BSIM4, PSP, and HISIM) that do incorporate these effects are typically used in industrial simulation.

[1] e.g., the MOS Level 1 model incorporates channel resistance effects by multiplying Equation (2.19) with a term $1 + \lambda V_{DS}$, in addition to other modifications.

Observe that even though Equation (2.19) contains `if` conditions, it is still possible for the equations to be continuous, differentiable and smooth.

2.12 Continuity, Differentiability and Smoothness

From the standpoint of suitability for various numerical solution methods (such as Newton–Raphson, ODE/DAE initial value solution, *etc.*), as well as from fundamental physical considerations, it is crucial that element equations not only be well defined over a realistic domain of input values, but that they be continuous and at least once differentiable. It is also desirable (for, e.g., distortion analysis) that several higher-order derivatives exist — indeed, that they be infinitely differentiable.

In practice, however, many existing device models exhibit discontinuities in their equations. To be able to use them robustly in circuit simulation, it is necessary to smooth out these discontinuities mathematically.

2.12.1 Smoothing Discontinuous Equations

We illustrate a generic procedure for smoothing discontinuous equations, using a one-dimensional example. Consider the function:

$$f(x) = \begin{cases} x, & x \leq 1, \\ 2x, & x > 1, \end{cases} \tag{2.21}$$

which has a discontinuity at $x = 1$.

To better motivate the smoothing procedure, we first re-write $f(x)$ using the unit step function $u(t)$ to capture the discontinuity:

$$f(x) = x\ (1 - u(x-1)) + 2x\ u(x-1). \tag{2.22}$$

Equation (2.22) is identical to Equation (2.21) (except possibly at the exact point of discontinuity).

To smooth Equation (2.22), simply replace the unit step function $u(t)$ by a smooth approximation $s(t)$. For example, one possibility is to

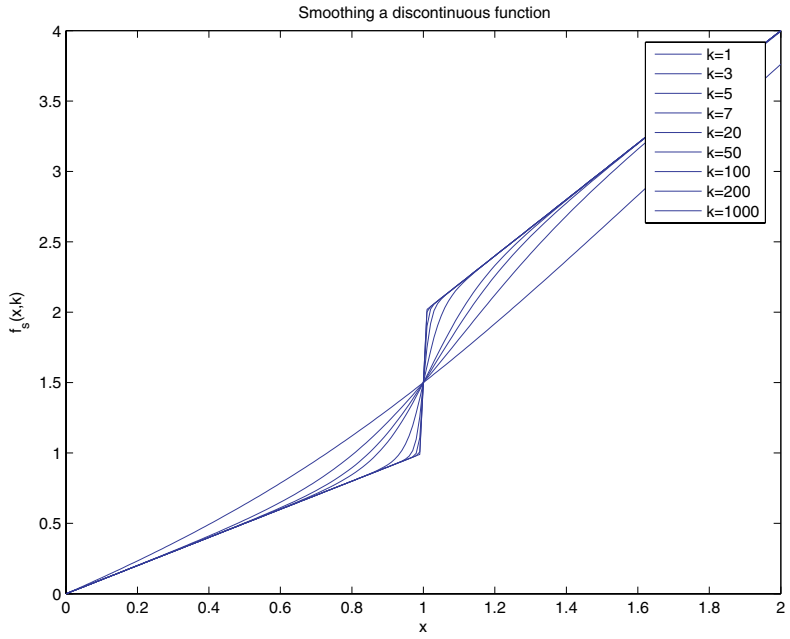


Fig. 2.9 Smoothing Equation (2.21).

use the smoothing function:

$$s(t,k) = \frac{1 + \tanh(kt)}{2}, \tag{2.23}$$

where k is a parameter that controls how closely $s(t,k)$ approximates $u(t)$ — for large k, the approximation is closer. There are many other possibilities for smoothing functions, of course.

Replacing $u(t)$ by $s(t,k)$ in Equation (2.22), we obtain the smoothed version:

$$f_s(x,k) \simeq f(x) = x\ (1 - s(x-1,k)) + 2x\ s(x-1,k). \tag{2.24}$$

Equation (2.24) is plotted in Figure 2.9, for various values of k.

3

Circuits as Nonlinear Differential–Algebraic Equations

The fundamental premise behind computer simulation is that appropriate mathematical representations can abstract a physical system's behavior usefully. Algorithms are then applied to solve these representations on a computer.

For most kinds of systems (including circuits), *systems of differential equations*, or more precisely, systems of nonlinear, differential–algebraic equations (DAEs), are an appropriate representation that is widely used. In particular, the following special DAE form is most often used:

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

In Equation (3.1), all quantities (except t, the time variable) are vectors of size n, which is (roughly speaking) the size of the circuit represented by the DAE. The nonlinear vector functions $\vec{q}(\cdot)$ and $\vec{f}(\cdot)$ represent the charge/flux and resistive parts of the circuit, respectively, while $b(t)$ is a forcing term representing external inputs from independed current and voltage sources. As we will see shortly, this DAE form is obtained by writing out the fundamental charge and potential conservation equations of the circuit — Kirchoff's Current Laws (KCL) and Kirchoff's

Voltage Law (KVL) — together with the Branch Constitutive Relationships (BCR) of the individual elements.

3.1 Writing Circuit Equations: A Simple Example

We refer the reader to e.g., [7, 48, 49], for more detailed expositions of how these equations can be obtained for a given circuit topology. It is not difficult, however, to understand how to obtain the quantities and functions in Equation (3.1) for any circuit of interest, either "by hand" or via a computer procedure. Consider the simple circuit shown in Figure 3.1, consisting of a voltage source driving a parallel-RC load through a nonlinear diode. The electrical quantities of interest in this circuit are the two node voltages $e_1(t)$ and $e_2(t)$, as well as the current through the voltage source, $i(t)$.

The resistor's action on the circuit is defined by its current–voltage relationship (Ohm's Law), $i_R(t) = \frac{e_2(t)}{R}$; similarly, the capacitor's current voltage relationship is given by $i_C(t) = \frac{d}{dt}(Ce_2(t))$. The current–voltage relationship of the diode element, which is nonlinear, is given by

$$i = d(e_1 - e_2) = I_s \left(e^{\frac{e_1 - e_2}{V_T}} - 1 \right), \tag{3.2}$$

where I_s and V_T are constants. Finally, the voltage source is captured simply by its setting the voltage of node 1 to its value: $e_1(t) = E$. These four relationships, one for each of the four circuit elements, are the BCRs relevant to this circuit.

By summing currents at each node of interest (this is *Kirchoff's Current Law*, or **KCL**), the circuit equations are obtained. At node 1,

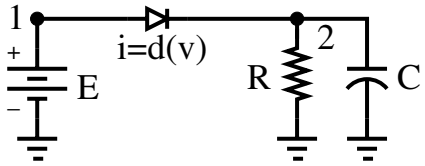


Fig. 3.1 Simple diode–resistor–capacitor circuit.

we obtain

$$i(t) + d(e_1 - e_2) = 0, \tag{3.3}$$

while at node 2, we have

$$\frac{d}{dt}(Ce_2) + \frac{e_2}{R} - d(e_1 - e_2) = 0. \tag{3.4}$$

Finally, we also have (for the voltage source)

$$e_1 = E. \tag{3.5}$$

These are known as the Modified Nodal Equations (MNA) of the circuit.

By writing Equation (3.3)–(3.5) in vector form, the form of Equation (3.1) can be obtained. First, the voltage–current unknowns are written in vector form as:

$$\vec{x} = \begin{bmatrix} e_1(t) \\ e_2(t) \\ i(t) \end{bmatrix}. \tag{3.6}$$

Define the vector function $q(\cdot)$ in Equation (3.1) to be

$$\vec{q}(\vec{x}) \equiv \begin{bmatrix} 0 & & \\ & C & \\ & & 0 \end{bmatrix} \vec{x} = \begin{bmatrix} 0 \\ Ce_2(t) \\ 0 \end{bmatrix}, \tag{3.7}$$

$f(\cdot)$ to be

$$\vec{f}(\vec{x}) \equiv \begin{bmatrix} i(t) + d(e_1 - e_2) \\ \frac{e_2}{R} - d(e_1 - e_2) \\ e_1 \end{bmatrix}, \tag{3.8}$$

and the vector $b(t)$ to be

$$\vec{b}(t) = \begin{bmatrix} 0 \\ 0 \\ -E(t) \end{bmatrix}. \tag{3.9}$$

With these definitions, Equation (3.1) simply represents the circuit Equations (3.3)–(3.5), but in vector DAE form. The utility of writing circuit equations in this canonical form is that a variety of mathematical

and computational techniques may be brought to bear on Equation (3.1), without having to worry about the details of $\vec{f}(\cdot)$, $\vec{q}(\cdot)$ and $\vec{b}(t)$, so long as they can be realistically assumed to have some basic properties like continuity, smoothness, *etc.* Indeed, the DAE form (Equation (3.1)) is not restricted to circuits; it subsumes virtually all physical systems, including those from mechanical, optical, chemical, *etc.* applications.

It will be readily understood, of course, that DAEs for typical industrial circuits are usually much larger and more complex than the one for the simple example of Figure 3.1. Much of the complexity of these equations stems from the complicated equations describing the *semiconductor devices*, such as MOS and bipolar transistors, that populate circuits in large numbers.

3.2 Kirchoff's Current and Voltage Laws

The example above showed how to write circuit equations from inspection of a circuit diagram. It is, in fact, possible to derive these circuit equations systematically from a few basic rules:

- **Kirchoff's Current Law (KCL)**, which states that the sum of all currents into a node is always zero.
- **Kirchoff's Voltage Law (KVL)**, which states that the sum of branch voltages around all closed loops is zero.
- **Branch Constitutive Relationships (BCRs)**, these equations relate branch currents and branch voltages. The element and device equations encountered earlier are all BCRs.

A useful concept for capturing the topological connectivity of a circuit is the *incidence matrix*. This is a rectangular matrix that indicates which branches are connected to which node. Each branch in circuit graph is given a direction (indicating positive current flow). Each row of the incidence matrix corresponds to a node of the graph; each column to a branch in the graph. If the current flows out of a node n via

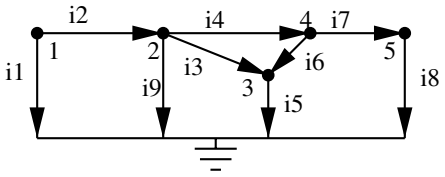


Fig. 3.2 Circuit connectivity graph.

branch b, the corresponding matrix entry (n, b) has a 1 entry; if the current flows into the node, the corresponding entry is -1.

For example, for the graph shown in Figure 3.2, the incidence matrix is

$$\begin{array}{r|ccccccccc} & i_1 & i_2 & i_3 & i_4 & i_5 & i_6 & i_7 & i_8 & i_9 \\ \text{node 1:} & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{node 2:} & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ \text{node 3:} & 0 & 0 & -1 & 0 & 1 & -1 & 0 & 0 & 0 \\ \text{node 4:} & 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 \\ \text{node 5:} & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ \hline \text{ground:} & -1 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & -1 \end{array} . \tag{3.10}$$

In the above, we have included a row for the ground node; in which case, the matrix is called an *augmented* incidence matrix. Conventionally, the ground row is not included in an incidence matrix; hence, the incidence matrix is

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \end{pmatrix} . \tag{3.11}$$

Now, define a vector $\vec{i}_b$ of branch currents, in the order of columns of the incidence matrix. From the manner in which the incidence matrix has been defined, it is easy to see that KCL can simply be expressed

as $A\vec{i}_b = \vec{0}$ — or, in expanded form:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \\ i_7 \\ i_8 \\ i_9 \end{pmatrix} = \vec{0}. \tag{3.12}$$

Another useful feature of the incidence matrix is that it can be easily used to express branch voltages in terms of the node voltages. Let $\vec{e}$ represent a vector of all the node voltages (in the same order as the rows of the incidence matrix); then, the branch voltages of the graph ($\vec{v}_b$, in the same order the branch currents $\vec{i}_b$) can be expressed as:

$$\vec{v}_b = A^T \vec{e}. \tag{3.13}$$

This is essentially a re-statement of KVL. If there are n nodes (excluding ground) and b branches, we have $\vec{e} \in \mathcal{R}^n$, $\vec{v}_b \in \mathcal{R}^b$, $\vec{i}_b \in \mathcal{R}^b$, and $A \in \mathcal{R}^{n \times b}$.

3.3 Sparse Tableau Formulation

If we put KCL and KVL together with all the equations representing the elements' branch constitutive relationships (BCRs, from Section 2), we obtain a complete system of equations for the circuit:

$$\begin{aligned} A\vec{i}_b &= \vec{0} \\ A^T\vec{e} - \vec{v}_b &= \vec{0} \\ \frac{d}{dt}\vec{q}_b\big(\vec{v}_b, \vec{i}_b\big) + \vec{f}_b\big(\vec{v}_b, \vec{i}_b\big) + \vec{b}(t) &= \vec{0}. \end{aligned} \tag{3.14}$$

This is known as the *Sparse Tableau* formulation. Defining

$$\begin{aligned} \vec{x}_{ST} &\triangleq \begin{bmatrix} \vec{e} \\ \vec{v}_b \\ \vec{i}_b \end{bmatrix}, \quad \vec{f}_{ST}(\vec{x}_{ST}) \triangleq \begin{bmatrix} A\vec{i}_b \\ A^T\vec{e} - \vec{v}_b \\ \vec{f}_b(\vec{v}_b, \vec{i}_b \end{bmatrix}, \\ \vec{q}_{ST}(\vec{x}_{ST}) &\triangleq \begin{bmatrix} \vec{0} \\ \vec{0} \\ \vec{q}_b(\vec{v}_b, \vec{i}_b) \end{bmatrix}, \quad \text{and } \vec{b}_{ST}(t) \triangleq \begin{bmatrix} \vec{0} \\ \vec{0} \\ \vec{b}(t) \end{bmatrix}, \end{aligned} \tag{3.15}$$

we observe that Equation (3.14) can be written as:

$$\frac{d}{dt}\vec{q}_{ST}(\vec{x}_{ST}) + \vec{f}_{ST}(\vec{x}_{ST}) + \vec{b}_{ST}(t) = \vec{0}, \tag{3.16}$$

i.e., in the form of Equation (3.1). Note that Equations (3.16) and (3.14) are "square" systems, with $2b + n$ unknowns and as many equations.

3.4 Nodal Formulation

Suppose we are given the branch constitutive relationships (BCRs) of all the elements/devices in the circuit in the following special (*voltage controlled*) form:

$$\vec{i}_b = \vec{f}_b(\vec{v}_b) + \frac{d}{dt}\vec{q}_b(\vec{v}_b) + \vec{b}_v(t). \tag{3.17}$$

By applying KCL to the BCR equations and using KVL to eliminate the branch voltages, it becomes possible to eliminate all the branch quantities (i.e., $\vec{v}_b$ and $\vec{i}_b$), to obtain a small set of equations for the circuit in which the only unknowns are the n node voltages:

$$A\left[\vec{f}_b\left(A^T\vec{e}\right) + \frac{d}{dt}\vec{q}_b\left(A^T\vec{e}\right) + \vec{b}_v(t)\right] = \vec{0}. \tag{3.18}$$

The above procedure, in which we are able to express the circuit equations entirely in terms of node voltage unknowns, is known as *Nodal Analysis (NA)*.[1] Observe, again, that this is a "square" system of equations — i.e., there are as exactly as many equations as unknowns — a feature we will find essential when we solve equations.

[1] *Nodal Formulation* (NF) would perhaps be a better name, but the term Nodal Analysis is established.

3.5 Modified Nodal Formulation

It is not always possible to express the BCRs of *all* the elements/devices in the circuit in the voltage-controlled form (Equation (3.17)). For example, independent voltage sources cannot be written in this form, since the current through a voltage source cannot be expressed in terms of the voltage across it. NA does not, therefore, apply to circuits which have elements that are not voltage controlled.

However, it is still possible to write a system of equations of much smaller size than STA. The basic idea behind the resulting *Modified Nodal Analysis (MNA)* procedure is to split up the BCRs into two groups: voltage controlled and non-voltage controlled. For the voltage-controlled group of BCRs, we apply the same procedure as for NA. For the remaining BCRs — for which the branch currents cannot be represented explicitly as functions of branch voltages — the branch currents are included as *unknowns* that need to be solved for. The extra unknowns are compensated for with extra equations added to the system.

As before, let there be n nodes in the circuit; and let there be b_v branch currents that can be expressed using voltage-controlled BCRs; and another b_o branch currents that cannot be expressed explicitly in terms of branch voltages. Denote by $\vec{i}_{bv}$ the vector of the voltage-controlled branch currents, and by $\vec{i}_{bo}$ the remaining ones. Then, Equation (3.12) can be written as:

$$\begin{pmatrix} A_v & A_o \end{pmatrix} \begin{pmatrix} \vec{i}_{bv} \\ \vec{i}_{bo} \end{pmatrix} = \vec{0}. \tag{3.19}$$

The branch voltages can still expressed in terms of the node voltages as:

$$\vec{v}_b = \begin{pmatrix} A_v^T \\ A_o^T \end{pmatrix} \vec{e} = A^T \vec{e}. \tag{3.20}$$

The BCRs can also be split into two groups. The first set of b_v BCR equations express $\vec{i}_{bv}$ in terms of the branch voltages:

$$\vec{i}_{bv} = \vec{f}_{bv}(\vec{v}_b) + \frac{d}{dt}\vec{q}_{bv}(\vec{v}_b) + \vec{b}_v(t). \tag{3.21}$$

The second set of b_o BCR equations are, in general, *implicit* in $\vec{v}_{bo}$, as well as in $\vec{v}_b$:

$$\vec{0} = \vec{f}_{bo}\left(\vec{v}_b, \vec{i}_{bo}\right) + \frac{d}{dt}\vec{q}_{bo}\left(\vec{v}_b, \vec{i}_{bo}\right) + \vec{b}_o(t). \tag{3.22}$$

Next, we eliminate $\vec{i}_{bv}$ and $\vec{v}_b$ from the BCRs. Applying Equations (3.19) and (3.20) to Equation (3.21), we obtain

$$A_v\left[\vec{f}_{bv}\left(A^T\vec{e}\right) + \frac{d}{dt}\vec{q}_{bv}\left(A^T\vec{e}\right) + \vec{b}_v(t)\right] + A_o\vec{i}_{bo} = \vec{0}. \tag{3.23}$$

Note that the above represents n equations in $n + b_o$ unknowns. We now substitute Equation (3.20) in Equation (3.22) to obtain b_o more equations in the same unknowns:

$$\vec{f}_{bo}\big(A^T\vec{e}, \vec{i}_{bo}\big) + \frac{d}{dt}\vec{q}_{bo}\big(A^T\vec{e}, \vec{i}_{bo}\big) + \vec{b}_o(t) = \vec{0}. \tag{3.24}$$

Putting Equations (3.23) and (3.24) together, we obtain the complete equations for the circuit:

$$\begin{aligned} A_v\left[\vec{f}_{bv}(A^T\vec{e}) + \frac{d}{dt}\vec{q}_{bv}(A^T\vec{e}) + \vec{b}_v(t)\right] + A_o\vec{i}_{bo} = \vec{0}, \\ \vec{f}_{bo}(A^T\vec{e}, \vec{i}_{bo}) + \frac{d}{dt}\vec{q}_{bo}(A^T\vec{e}, \vec{i}_{bo}) + \vec{b}_o(t) = \vec{0}. \end{aligned} \tag{3.25}$$

This is a square system of $n + b_o$ equations in the same number of unknowns.

Finally, we can easily rewrite Equation (3.25) in our standard circuit DAE form $\dot{q}(x) + f(x) + b(t) = 0$. Define the vector of unknowns in Equation (3.25):

$$\vec{x} = \begin{pmatrix} \vec{e} \\ \vec{i}_o \end{pmatrix}, \tag{3.26}$$

and the functions

$$\vec{f}(\vec{x}) = \begin{pmatrix} A_v\vec{f}_{bv}(A^T\vec{e}) + A_o\vec{i}_{bo} \\ \vec{f}_{bo}(A^T\vec{e}, \vec{i}_{bo}) \end{pmatrix}, \tag{3.27}$$

$$\vec{q}(\vec{x}) = \begin{pmatrix} A_v\vec{q}_{bv}(A^T\vec{e}) \\ \vec{q}_{bo}(A^T\vec{e}, \vec{i}_{bo}) \end{pmatrix}, \tag{3.28}$$

and

$$\vec{b}(t) = \begin{pmatrix} A_v \vec{b}_v(t) \\ \vec{b}_o(t) \end{pmatrix}. \tag{3.29}$$

With these definitions, Equation (3.25) can be rewritten in the form of Equation (3.1), reproduced here again:

$$\frac{d}{dt}\vec{q}(\vec{x}) + \vec{f}(\vec{x}) + \vec{b}(t) = \vec{0}. \tag{3.30}$$

3.6 Derivative ("Jacobian") Matrices and Device "Stamps"

As we will see later, solving Equation (3.1) numerically will require that we be able to evaluate $\vec{q}(\cdot)$, $\vec{f}(\cdot)$, and $\vec{b}(\cdot)$, as well as the *derivatives* of $\vec{q}(\vec{x})$ and $\vec{f}(\vec{x})$. Since these are vector functions of vector arguments, their derivatives are *matrices*; these derivative matrices are often termed *Jacobian matrices*.

To illustrate the structure of the Jacobian matrix, consider again the example of Figure 3.1, together with its $\vec{f}(\vec{x})$ defined in Equation (3.8). The Jacobian matrix of $\vec{f}(\vec{x})$ (i.e., $\frac{d}{d\vec{x}}\vec{f}(\vec{x})$) in Equation (3.8) is

$$J(\vec{x}) = \begin{pmatrix} d'(v_1 - v_2) & -d'(v_1 - v_2) & 1 \\ -d'(v_1 - v_2) & \frac{1}{R} + d'(v_1 - v_2) & 0 \\ 1 & 0 & 0 \end{pmatrix}, \tag{3.31}$$

where $d'(\cdot)$ is the derivative of the diode's current–voltage relationship, defined in Equation (3.2). Observe, firstly, that each element in the circuit has a characteristic pattern of entries that it contributes to the Jacobian matrix. For example, the resistor R from node 2 to ground contributes the pattern:

$$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \frac{1}{R} & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}, \tag{3.32}$$

while the diode between nodes 1 and 2 contributes the pattern:

$$\begin{pmatrix} d'(v_1 - v_2) & -d'(v_1 - v_2) & \cdot \\ -d'(v_1 - v_2) & d'(v_1 - v_2) & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}. \tag{3.33}$$

The pattern of Jacobian entries contributed by each element is called its *matrix stamp*. The Jacobian matrix $J(\vec{x})$ thus consists of the addition of the stamps of all the elements in the circuit.

3.6.1 Jacobian Sparsity and Its Importance

Observe also that several entries in the Jacobian matrix in Equation (3.31) are zero. As the size of the circuit grows larger, the number of zero entries in the matrix predominates. The reason for this is easy to see — if the circuit has roughly n nodes and each node is connected to only two or three circuit elements (as is typical), the total number of nonzero matrix stamps is of the order of $2n$ or $3n$. As a result, the remaining entries of the matrix (which has a total of n^2 entries) must be zero or "unstamped". Such matrices, where there are relatively few nonzero entries and many zero entries, are called *sparse matrices*. When a matrix has no zero entries, it is called *dense*. We have just noted that because each node in typical circuits is connected only to a few elements, Jacobian matrices of circuits tend to be sparse. Note that the sparse structure of the circuit is first evident in its incidence matrix. As we will see later, the fact that circuit matrices are usually sparse is of enormous importance for circuit simulation.

4

Quiescent Steady-State Analysis via Newton–Raphson

In Section 3, we saw how any circuit an be written as a set of nonlinear differential equations in the form (Equation (3.1)). We now look at solving Equation (3.1) numerically for a basic type of solution: the *quiescent steady state* (QSS), i.e., a solution in which none of the quantities in Equation (3.1) change with time.

For circuits, the QSS is known as the *DC operating point*. Finding the DC operating point is a fundamental and basic requirement in circuit design. The DC operating point is essential not only as a first basic check of circuit operation, but is also a prerequisite for further analyses. For example, transient and sinusoidal periodic steady-state analyses rely on a prior DC operating point having been calculated; the same is usually the case for noise, sensitivity, harmonic balance, shooting, envelope, *etc.* , analyses.

In quiescent steady state analysis, the input $\vec{b}(t)$ is constant with time, i.e., $\vec{b}(t) = \vec{b}^*$; additionally, an important *assumption* is made about the solution of the circuit, namely that $\vec{x}(t) = \vec{x}^*$, i.e., that the solution is also constant with time. Since $\vec{x}(t)$ (hence $\vec{q}(\vec{x}(t))$) is constant, the derivative term in Equation (3.1) vanishes, hence

$$\vec{f}(\vec{x}^*) + \vec{b}^* = 0, \quad \text{or} \quad \vec{g}(\vec{x}^*) \triangleq \vec{f}(\vec{x}^*) + \vec{b}^* = \vec{0}. \tag{4.1}$$

This is, in general, a *nonlinear equation system*, where the unknown to be solved for is the vector $\vec{x}^*$. Solving such systems of equations is usually a non-trivial numerical task. In most applications, a numerical method known as the *Newton–Raphson algorithm* is most commonly used to solve nonlinear systems of equations.

4.1 Solving Nonlinear Equations: The Newton–Raphson Algorithm

The Newton–Raphson (NR) algorithm (e.g., [86]) is a technique that is widely used for finding solutions of nonlinear equations. It is an *iterative* method — that is to say, it starts from a guess for the solution $\vec{x}_0$ of Equation (4.1), and keeps refining the guess until it finds a solution. NR performs this refinement by approximating the nonlinear system locally around the current guess by a linearized system of the form $A\vec{x} = \vec{b}$, the solution of which is used to update the current guess. In other words, NR leverages our ability to solve algebraic linear systems of equations in each refinement step; as such, it is guaranteed to solve any already-linear system in a single iteration.

NR relies on having available two callable functions:

(1) a function to evaluate $\vec{g}(\vec{x})$, given any $\vec{x}$; and
(2) a function to evaluate the derivative (or Jacobian matrix) of $\vec{g}(\cdot)$, i.e., $J(\vec{x}) = \frac{d\vec{g}(\vec{x})}{d\vec{x}}$, given any $\vec{x}$. Note that $\vec{x}$ and $\vec{g}(\vec{x})$ are vectors of size n, while J is an $n \times n$ matrix.

The following is an outline of the basic NR algorithm, in MATLAB-like pseudo-code[1]:

```
1  function solution = NR(xguess)
2    x = xguess;
3    finished = 0;
4    epsilon = 1e-10;
5    while (~finished)
6       gx = g(x);
```

[1] A simple implementation of NR is available at http://www.eecs.berkeley.edu/∼jr/NOW-FT-Monograph/.

```
 7       if (norm(gx) < epsilon)
 8         finished = 1;
 9         solution = x;
10         break;
11       end
12       Jx = dgdx(x);
13       delta_x = - inverse(Jx)*gx;
14       x = x + delta_x;
15    end
```

Observe that a key step is line 13, where the current guess for the solution is updated by an amount $\Delta\vec{x} = \vec{x}_{i+1} - \vec{x}_i$, calculated by solving

$$J(\vec{x}) \cdot \Delta\vec{x} = -\vec{g}(\vec{x}), \quad \text{or} \quad \vec{x}_{i+1} - \vec{x}_i = -J^{-1}(\vec{x}_i) \cdot \vec{g}(\vec{x}_i). \tag{4.2}$$

The above involves solving a linear matrix equation (using the Jacobian matrix $J(\vec{x})$) at every iteration of the NR algorithm.

4.1.1 Graphical Interpretation for Scalar $g(x) = 0$

For scalar $g(x) = 0$, Newton–Raphson has a pleasing graphical interpretation, depicted in Figure 4.1, which offers insights into the operation of the algorithm. The Newton update step (Equation (4.2)), when interpreted graphically, corresponds to drawing a straight line of slope $\frac{dg(x_i)}{dx}$ through the point $(x_i, g(x_i))$, and finding its intersection with the horizontal axis; the point of intersection is the next iterate x_{i+1}.

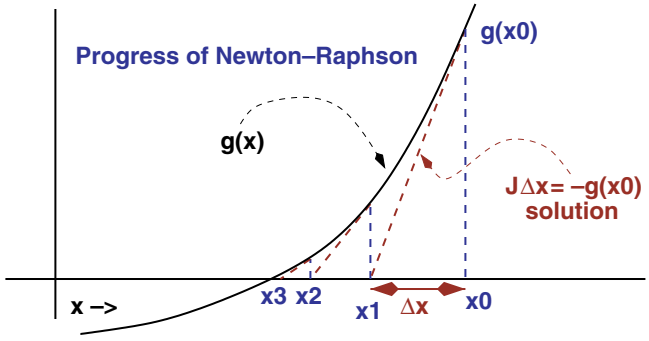


Fig. 4.1 Newton–Raphson interpreted graphically.

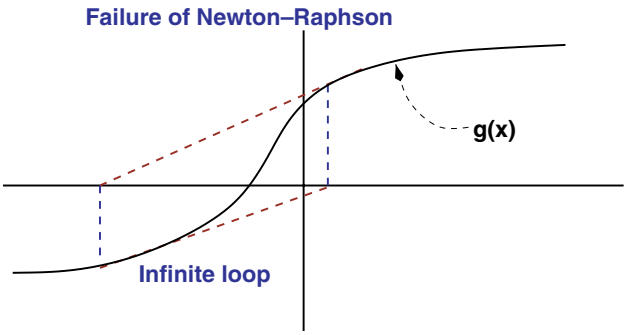


Fig. 4.2 Non-convergent Newton–Raphson iteration.

For the example $g(x)$ shown in Figure 4.1, it is visually apparent that a few such updates lead rapidly to the point where $g(x)$ interects the horizontal axis, i.e., to the solution of $g(x) = 0$. Indeed, the rapid convergence of Newton–Raphson in regions close to the solution is one of its greatest strengths; in most situations, this convergence is quadratic, as we shall see below.

It is important to realize, however, that there is *no guarantee* that Newton–Raphson will succeed in finding a solution for a given $g(x)$ and an initial guess x_0. Figure 4.2 illustrates graphically how Newton–Raphson can fail. For the initial guess shown, it is apparent that the NR algorithm will go into an infinite loop. It is also visually apparent that if the initial guess is even further from the solution, the algorithm will diverge; i.e., subsequent iterations will take x_i further and further away from the solution.

Therefore, the algorithm is typically implemented to announce failure after a number of iterations (e.g., 100 iterations). In practice, heuristics and modifications of the basic algorithm are often used to help it converge better to a solution.

4.1.2 Convergence Criteria: `reltol` and `abstol`

Note that lines 7–11 in the NR algorithm above set up the variable `finished`, which controls (line 5) when the algorithm exits or *converges* to a solution. The convergence criterion shown in lines 7–11, that:

$$\|\vec{g}(\vec{x})\| < \epsilon_g \tag{4.3}$$

for some small value of ϵ_g, is meant to test whether $\vec{g}(\vec{x})$ is acceptably close to $\vec{0}$, i.e., whether a solution $\vec{g}(\vec{x}) \simeq \vec{0}$ has been reached. Since the value of $\vec{g}(\vec{x})$ at any point in the NR algorithm is known as the *residual*, this convergence criterion is an example of a *residual-based convergence criterion*.

Other convergence criteria are also possible. For example, a criterion based on the Newton update step $\Delta\vec{x}$ is widely used in circuit simulation. This criterion checks if

$$\|\Delta\vec{x}\| < \epsilon_\Delta, \tag{4.4}$$

motivated by the fact that $\Delta\vec{x} = \vec{0}$ at the exact solution. A reason why the update step criterion is often preferred over the residual condition is that $\Delta\vec{x}$ consists of node voltage and current variables, hence it is more convenient to select physically reasonable values for ϵ_Δ than for ϵ_g. Indeed, it is typical to change ϵ_Δ during the progress of the NR iteration, based on the magnitude of the solution iterate $\vec{x}$, as follows:

$$\epsilon_\Delta = \epsilon_\Delta(\vec{x}) = \texttt{reltol} \times \|\vec{x}\| + \texttt{abstol}. \tag{4.5}$$

`reltol` is a relative tolerance parameter, while `abstol` is an absolute tolerance number. In electronics, typical values of `reltol` range from 10^{-3} to 10^{-6}; values for abstol are typically in the range 10^{-9} to 10^{-12}.

Often, it is useful to ensure that *each component* of $\Delta\vec{x}$, not simply its aggregate norm, is below some small number; i.e.,

$$|\Delta\vec{x}| < R \times |\vec{x}| + \vec{a}, \tag{4.6}$$

where R is a diagonal matrix of individual `reltol`s and $\vec{a}$ a vector of individual `abstol`s. This convergence criterion allows each component of the unknown vector to have its own, tailored, convergence parameters. It is typical for voltage variables and current variables to have different values of `reltol` and (especially) `abstol`.

The choice of convergence criterion is an important heuristic for making Newton–Raphson converge robustly across a wide range of circuits. Practical simulators often use a combination of residual and Newton update convergence criteria.

4.2 Linear Solution: Importance of Jacobian Sparsity

We noted earlier (Section 3) that the fact that circuit matrices are typically sparse is of great importance in simulation. The reason is that, while the Newton update step in Equation (4.2) is in general computationally expensive for arbitrary matrices J (e.g., if all entries of J are non-zero; i.e., J is *dense*), efficient techniques, collectively dubbed "sparse matrix technology", exist for solving for Δx in Equation (4.2), when J is sparse. More specifically, the computational complexity of general linear equation solution (e.g., when J is dense) is $O(n^3)$, while that for typical sparse J is $O(n)$. When circuit sizes n reach the 1,000s or 10s of 1,000s, this difference in computational complexity becomes very significant. Thus, sparsity is important for enabling efficient, practically effective, linear solution for circuits — not only for QSS analysis, but in virtually all algorithms for numerical simulation.

4.3 Quadratic Local Convergence of Newton–Raphson

One of the strengths of Newton–Raphson is that for initial guesses sufficiently close to the solution, it can be shown to converge very rapidly. Indeed, in most cases, the error at a given iteration falls as the square of the error at the previous iteration, a property known as *quadratic local convergence*. To see this, we express Newton–Raphson as a recurrence in the error at each iteration (focussing on scalar $g(x)$ for simplicity).

Assume that x^* is an exact solution of $g(x) = 0$. For small ϵ, we can expand $g(x + \epsilon)$ in Taylor series as:

$$g(x^* + \epsilon) = g(x^*) + J^*\epsilon + Q^*\epsilon^2 + R_1(\epsilon), \tag{4.7}$$

where $J^* = J|_{x^*} = \triangleq \left.\frac{dg}{dx}\right|_{x=x^*} = g'(x^*)$, $Q^* \triangleq \frac{1}{2}\left.\frac{d^2g}{dx^2}\right|_{x=x^*} = \frac{1}{2}g''(x^*)$ and $R_1(\epsilon)$ represents higher-order terms in ϵ, i.e.,

$$R_1(\epsilon) \triangleq \frac{1}{3!}g'''(x^*)\epsilon^3 + \frac{1}{4!}g''''(x^*)\epsilon^3 + \cdots. \tag{4.8}$$

It is useful to rewrite Equation (4.7) in terms of $J|_{x^*+\epsilon}$, since this quantity arises in the Newton–Raphson update step. Toward this end,

note that:

$$\begin{aligned} J|_{x^*+\epsilon} \triangleq g'(x^*+\epsilon) =& g'(x^*) + g''(x^*)\epsilon + R_2(\epsilon) \\ =& J^* + 2Q^*\epsilon + R_2(\epsilon), \end{aligned} \tag{4.9}$$

where

$$R_2(\epsilon) \triangleq \frac{1}{2}g'''(x^*)\epsilon^2 + \frac{1}{3!}g''''(x^*)\epsilon^3 + \cdots \tag{4.10}$$

consists of higher-order terms in ϵ.

Using Equation (4.9) and the fact that $g(x^*) = 0$, Equation (4.7) can be rewritten as:

$$\begin{aligned} g(x^*+\epsilon) &= \left(J|_{x^*+\epsilon} - 2Q^*\epsilon - R_2(\epsilon)\right)\epsilon + Q^*\epsilon^2 + R_1(\epsilon) \\ &= J|_{x^*+\epsilon}\epsilon - Q^*\epsilon^2 + R_1(\epsilon) - R_2(\epsilon)\epsilon. \end{aligned} \tag{4.11}$$

Defining the error at the i^{th} Newton iterate to be $\epsilon_i \triangleq x_i - x^*$, and denoting $J_i = J|_{x^*+\epsilon_i}$, we can express the update step (Equation (4.2)) as:

$$J_i(\epsilon_{i+1} - \epsilon_i) = -g(x^*+\epsilon_i) = -J_i\epsilon_i + Q^*\epsilon_i^2 - R_1(\epsilon_i) + R_2(\epsilon_i)\epsilon_i, \tag{4.12}$$

which can be rewritten as:

$$\begin{aligned} \epsilon_{i+1} &= \frac{Q^*\epsilon_i^2 - R_1(\epsilon_i) + R_2(\epsilon_i)\epsilon_i}{J_i} \\ &= \frac{Q^*\epsilon_i^2 - R_1(\epsilon_i) + R_2(\epsilon_i)\epsilon_i}{J^* + 2Q^*\epsilon_i + R_2(\epsilon_i)}. \end{aligned} \tag{4.13}$$

Observe that if $J^* \neq 0$ and $Q^* \neq 0$ (as is usually the case), Equation (4.13) approximates to

$$\epsilon_{i+1} \simeq \frac{Q^*}{J^*}\epsilon_i^2. \tag{4.14}$$

Equation (4.14) states that a small error at the ith Newton iterate becomes much smaller, due to squaring, at the $(i+1)$th iterate; i.e., that Newton–Raphson is quadratically convergent in regions close to the solution, provided the first and second derivatives of $g(x)$ are both nonzero at the solution. Numerically, this translates to a doubling of the number of digits of accuracy at each iteration. Figure 4.3 illustrates how the error diminishes quadratically for the function $g(x) \triangleq x^2 - 4$.

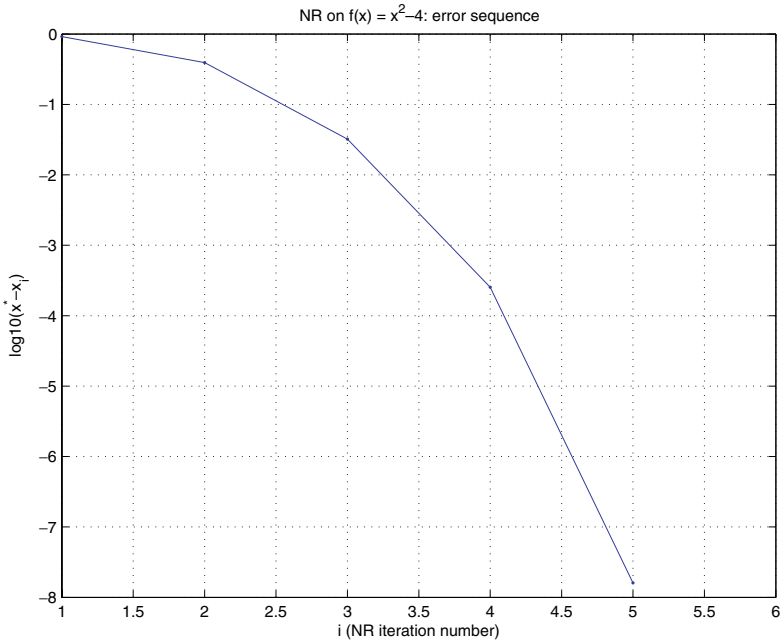


Fig. 4.3 Quadratic local convergence of Newton–Raphson.

4.4 Convergence Heuristics: Device Initialization and Limiting

In spite of its local quadratic convergence properties, Newton–Raphson frequently does not converge if the initial guess is not close to the solution. Convergence problems are especially exacerbated by functions that grow very rapidly, and also by functions that feature regions with very low slopes, leading to singular Jacobian problems. In particular, exponential functions, which abound in electronics and many other domains, can break Newton–Raphson by a combination of these mechanisms.

4.4.1 Newton–Raphson Failure Mechanisms: A Diode-Resistor Circuit

Running Newton–Raphson on the scalar nonlinear equation for the simple diode–resistor circuit of Figure 3.1 offers insights into common

failure mechanisms.[2] Ignoring the capacitor (which contributes nothing in QSS analysis), the KCL equation for e_2 is

$$g(e_2) \equiv d(e_2) - \frac{E - e_2}{R} = 0, \tag{4.15}$$

where $d(e_2) \triangleq I_s(e^{\frac{e_2}{V_T}} - 1)$ is the diode's exponential BCR. For concreteness, assume $E = 10\text{V}$, $V_T = 0.025\text{V}$, $I_s = 10^{-12}$ and $R = 10\Omega$.

Observe that for "reasonable values" of e_2, i.e., of the order of 1V, the magnitude of the term $\frac{E-e_2}{R}$ is of the order of 1. However, the magnitude of $d(e_2)$ varies very widely: for $e_2 < -1$, $d(e_2) \sim -I_s = -10^{-12}$; at $e_2 = 0.5$ V, $d(e_2) \sim 10^{-3}$; at $e_2 = 0.7$ V, $d(e_2) \sim 1$; at $e_2 = 1$ V, $d(e_2) \sim 10^5$; at $e_2 = 2$ V, $d(e_2) \sim 10^{23}$; at $e_2 = 3$ V, $d(e_2) \sim 10^{40}$; and at $e_2 = 5$ V, $d(e_2) \sim 10^{75}$!

When the magnitudes of $d(e_2)$ and $\frac{E-e_2}{R}$ differ by a factor of about 10^{16} or more, double-precision computer arithmetic, which can represent only about 16 decimal digits, loses all accuracy when adding or subtracting the two terms. In other words, for $e_2 > 1.6$ V or so, Equation (4.15) becomes *numerically equivalent* to simply

$$g(e_2) = d(e_2) = I_s e^{\frac{e_2}{V_T}} = 0. \tag{4.16}$$

Equation (4.16) is simple enough that its Newton update can be expressed analytically in closed form: it is $\Delta e_2 = -\frac{g(e_2)}{g'(e_2)} = -V_T$.

In other words, if e_2 exceeds about 1.6 V, the Newton–Raphson update step will be $-V_T \simeq -0.025$ V. For example, if $e_2 = 10$ V, it will take about $\frac{9}{0.025} = 360$ iterations for Newton to refine e_2 to 1 V. Since the maximum number of iterations typically used by Newton–Raphson implementations is usually much smaller (from a few 10s up to about 100), the algorithm would announce failure.

Furthermore, for $e_2 > 17.5$ or so, the exponential in Equation (4.16) exceeds the largest number that can be represented in double precision arithmetic (about 10^{308}), resulting in a *floating point exception* during its computation. In such situations, the algorithm breaks down immediately because $g(e_2)$ cannot even be computed.

It is evident from the above that convergence considerations dictate that the initial guess for e_2 should not be very much more than 1 V.

[2] MATLAB scripts are available at http://www.eecs.berkeley.edu/~jr/NOW-FT-Monograph/.

However, it is equally important that the initial guess not be too low. Consider, for example, the choice of initial guess $e_2 = 0$, at which $d(e_2) = 0$ and $d'(e_2) = \frac{I_s}{V_T} \simeq 10^{-10}$, which is small compared to $\frac{1}{R} = 0.1$. As a result, the Newton update for Equation (4.15) is dominated by the term $\frac{E-e_2}{R}$. This results in a large update $\Delta e_2 = 10$ V, leading to $e_2 = 10$ V at the next Newton iterate, with subsequent iterations following the failure pattern for $e_2 > 1.6$ V noted above. Indeed, numerical experiments[3] show that for an implementation that allows up to 100 iterations before announcing failure, the initial guess needs to be approximately in the range 0.5–3 V for Newton–Raphson to converge to the solution of 0.68899 V.

The above analysis motivates two types of heuristics for improving the convergence of Newton–Raphson:

(1) *Initialization heuristics* for choosing "good" initial guesses, and
(2) *Limiting heuristics*, that modify the update step of Newton–Raphson to disallow large jumps in the arguments of rapidly growing functions (like exponentials).

4.4.2 Initialization Heuristics for Circuits

The basic idea behind choosing good initial guesses for Newton–Raphson is to ensure that each nonlinear element receives branch voltage–current arguments that lead to outputs that are representative of typical operation; additionally, good initial guesses try to ensure that each nonlinear element contributes non-vanishing Jacobian terms.

For example, in a diode, this would mean that the branch voltage for the first Newton iteration should be about $v_b = 0.7$ V, at which $i_b = d(v_b) \sim 1$ and $d'(v_b) \sim 40$. Smaller branch voltages lead to vanishing d', while larger branch voltages to $i_b = d(v_b)$ being many orders of magnitude larger than currents typically encountered in electronic applications. An appropriate branch voltage for initialization can be chosen by inverting the diode BCR for a branch current value chosen to represent typical operation. For example, for $i_b = 10$ mA, we

[3] Downloadable from http://www.eecs.berkeley.edu/∼jr/NOW-FT-Monograph/.

obtain $v_b \simeq V_T \ln\left(\frac{10^{-2}}{I_s}\right) = 0.58$ V. It is common practice to use branch voltage/current initializations along similar lines for other nonlinear elements, especially BJTs and MOSFETs.

Depending on the equation formulation used, however, it may or not be possible to arrive at a consistent initial guess for the circuit equation system that respects the branch initializations of all the elements. For example, consider the case where NA or MNA is used for a circuit with two different diodes connected in parallel (i.e., between the same two nodes). For NA or MNA, the unknowns include only node voltages, not the branch voltages of the diodes; the branch voltage supplied to each of the two diodes is the same, obtained by subtracting node voltage unknowns. As a result, different branch voltage initializations for the two diodes cannot be respected. Note, however, that the Sparse Tableau formulation does not face this problem, since it retains the branch voltages of the two diodes as separate unknowns.

For NA/MNA, this inconsistency can be worked around at the software level, e.g., by passing an additional flag to the routines implementing $d(v_b)$ and $d'(v_b)$ that indicates the first step of Newton–Raphson. When the flag is raised, the routines ignore their input, instead using internally available branch voltages corresponding to initialization.

4.4.3 Limiting Heuristics for Circuits

Limiting heuristics for Newton–Raphson modify the update step (Equation (4.2)), replacing it with

$$\vec{y}_{i+1} = \vec{x}_i - J^{-1}(\vec{x}_i) \cdot \vec{g}(\vec{x}_i), \tag{4.17}$$

$$\vec{x}_{i+1} = \vec{l}(\vec{y}_{i+1}, \vec{x}_i). \tag{4.18}$$

$\vec{l}(\vec{y}_{i+1}, \vec{x}_i)$ is a *limiting function* that tries to ensure that large updates $\Delta\vec{x} = \vec{x}_{i+1} - \vec{x}_i$ are avoided. The intent is to limit large changes, like the $\Delta e_2 = 10$ V update encountered above for (Equation (4.15)).

As with initialization, limiting heuristics typically start at the level of individual circuit elements. For example, diodes (and other elements with PN junctions) use a technique called PNJLIM, which limits updates

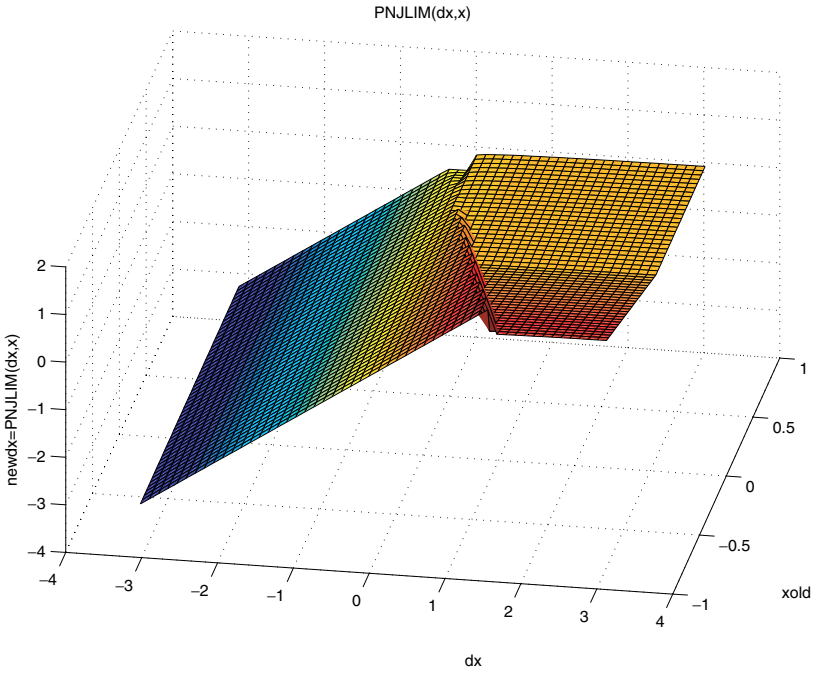


Fig. 4.4 Plot of PNJLIM.

to the branch voltage of the diode, is frequently used. PNJLIM[4] cuts the branch voltage update if it is too large ($> 2V_T$) and would result in a new branch voltage greater than a critical threshold of about 0.6 V. Figure 4.4 depicts how PNJLIM cuts the branch voltage update, using the value of the branch voltage at the current Newton–Raphson iterate (xold) and the unmodified update (dx) to arrive at a potentially limited update (newdx).

Since PNJLIM operates on branch voltages, it can lead to inconsistencies similar to those for initializations in the NA and MNA equation formulations. Unlike for initialization, where the inconsistency is only at the first iteration, limiting occurs at every Newton–Raphson iteration.

One way to arrive at a consistent limiting scheme for NA/MNA is to compute the ratio $\alpha = \frac{\text{newdx}}{\text{dx}}$ for every diode limited by PNJLIM, find

[4] Code implementing PNJLIM is available at http://www.eecs.berkeley.edu/∼jr/NOW-FT-Monograph/.

the minimum α over all elements, and use it to define the limited Newton step to be $\vec{x} \leftarrow \vec{x} + \alpha_{\min}\delta\vec{x}$, where $\delta\vec{x}$ is the original (unmodified) Newton update. This ensures a conservative update, that does not lead to branch voltage updates greater than those computed by PNJLIM for any of the elements in the circuit.

Limiting heuristics can, in practice, significantly improve the convergence characteristics of Newton–Raphson. For example, if $e_2 = 0$ in (4.15), PNJLIM prevents the next Newton iterate from reaching 10 V, limiting it to about 0.6 V instead; this leads to convergence to the solution (0.68899 V) in a few iterations. Other limiting routines, such as FETLIM for MOSFETs, are used for other nonlinear elements as aids for convergence.

5

Transient Simulation

In the previous section, we saw how the Newton–Raphson method can be used to solve Equation (3.1) for quiescent steady states. Quiescent steady states are, of course, only a very restricted type of solution of Equation (3.1). We now consider solving Equation (3.1) for the most general type of solution: where the inputs $\vec{b}(t)$ can be any specified function of time, leading to the solution $\vec{x}(t)$ also being a function of time. Finding a numerical approximation to $\vec{x}(t)$, termed *initial value* or *transient* simulation, is possibly the most widely used numerical analysis technique in applications.

5.1 Existence and Uniqueness of ODE Solutions

Before looking into numerical techniques for achieving the above, an important preliminary question needs to be considered. Does Equation (3.1) have a solution; and if it does, is the solution unique? Since Equation (3.1) typically represents a physical system, it seems reasonable to hope that, given a specific input $\vec{b}(t)$, a single, well-defined output $\vec{x}(t)$ should result. Before we attempt to find a numerical approximation of the solution, it is as well if we can assure ourselves

that a given differential equation in the form of Equation (3.1) does indeed have a well-defined solution. One reason why this is a good idea is that numerical approximations always introduce inaccuracies and artifacts; being assured that the DAE being solved has a well-defined solution is useful for understanding and debugging these numerical problems.

It turns out that, in order to be assured of a *unique* solution, we will also need an "initial condition" in addition to all the quantities in Equation (3.1). However, even if an initial condition is specified, not all equations of the form (Equation (3.1)) have well-defined or unique solutions. To appreciate this, consider the following simple, analytically solvable, differential equations:

(1) The simple linear differential equation:

$$\dot{x}(t) = \lambda x + b(t) \tag{5.1}$$

can be shown to have a unique solution if, in addition to λ and $b(t)$, a value for $x(t)$ is provided at any (single) time-point t_0. This value, $x_0 = x(t_0)$, is called an *initial condition* (IC) for the differential equation. Given the initial condition, the unique solution of Equation (5.1) is:

$$x(t) = x(t_0)e^{\lambda(t-t_0)} + \int_{t_0}^{t} e^{\lambda(t-\tau)}b(\tau)\,d\tau. \tag{5.2}$$

This solution is unique and well-defined for all t, for all initial conditions.

(2) The scalar differential equation

$$\dot{x}(t) = -\frac{1}{2x}, \tag{5.3}$$

with initial condition $x(0) = x_0$, has the solution

$$x(t) = \sqrt{x_0^2 - t}. \tag{5.4}$$

Observe that this solution is not defined for all t: for $t \geq x_0^2$, there is no solution in the domain of real numbers. This seems a departure from physical reality, where one expects *something* well defined to happen at all times. Hence, we are led

to conclude that Equation (5.3) is perhaps not a good representation of physical reality.
Some insight into this results from the observation that as $t \to x_0^2,\ x(t) \to 0$, resulting in $\frac{1}{x}$ blowing up, which seems physically unreasonable. Thus, we might surmise that physically relevant differential equations should satisfy some well-posedness conditions (including not blowing up for any x) to ensure that their solutions are physically reasonable (i.e., that they exist and are unique).

(3) The differential equation:

$$\dot{x}(t) = \frac{3}{2} x^{\frac{1}{3}}, \tag{5.5}$$

with initial condition $x(0) = 0$, has a solution

$$x(t) = \begin{cases} 0, & 0 <= t <= k \\ (t-k)^{\frac{3}{2}}, & t > k \end{cases} \tag{5.6}$$

for every choice of the parameter k. In other words, it has an infinite number of different solutions for the same initial condition! Such behavior certainly does not seem physical or natural. And yet, unlike the previous example, Equation (5.5) does not blow up for any value of x. This indicates that well-posedness of Equation (3.1) is not guaranteed simply by avoiding blow-ups, as in the previous example.

The above examples motivate us to look for conditions on Equation (3.1) that do guarantee that a unique, well-defined solution exists for all time. Before we do so, we will limit our attention to the restricted case where $\vec{q}(\vec{x}) \equiv \vec{x}$. In other words, Equation (3.1) becomes:

$$\frac{d}{dt}\vec{x}(t) + \vec{f}(\vec{x}) + \vec{b}(t) = \vec{0}. \tag{5.7}$$

This simpler form is known as an *Ordinary Differential Equation* (or ODE). Because theory and numerical techniques for ODEs are considerably more accessible, and better developed, than for DAEs like Equation (3.1), we will focus mostly on ODEs in this section. Numerical methods for ODEs do provide considerable insight into the more general case of DAEs — indeed, numerical techniques developed for ODEs can often be adapted successfully for use with DAEs.

One of the key benefits of limiting ourselves to ODEs is that well-established conditions for their solvability are available. In particular, the *Picard–Lindelöf theorem* establishes simple conditions for the existence and uniqueness of global solutions of ODEs. We state a simplified version here:

Theorem 5.1. *Picard–Lindelöf*: If $\vec{g}(\vec{x},t) \equiv -(\vec{f}(\vec{x}) + \vec{b}(t))$ is everywhere Lipschitz in $\vec{x}$, then Equation (5.7) has a unique global solution for any initial condition $\vec{x}(t_0) = \vec{x}_0$.

The Picard–Lindelöf existence and uniqueness theorem depends crucially on the concept of *Lipschitz continuity* of a function:

Definition 5.2. A function $\vec{g}(\vec{x},t)$ is said to be *Lipschitz* (or Lipschitz continuous) in $\vec{x}$ if there exists some finite number L such that:

$$\|\vec{g}(\vec{x}_1,t) - \vec{g}(\vec{x}_2,t)\| < L\|\vec{x}_1 - \vec{x}_2\|, \quad \forall \vec{x}_1, \vec{x}_2.$$

As depicted in Figure 5.1, Lipschitzness limits how quickly a function can change: for a scalar function, the quantity L must be greater than the maximum absolute value of the derivative of the function $g(x)$, anywhere. A function can be Lipschitz even if it is not differentiable, though it can easily be shown that a discontinuous function cannot be Lipschitz.

Re-examining examples (Equations (5.3) and (5.5)), we see that neither $\frac{1}{x}$ nor $\sqrt[3]{x}$ is Lipschitz, since the slopes of both tend to infinity as $x \to 0$. Therefore, the fact that neither equation has a globally unique solution is consistent with Theorem 5.1.

5.2 Basic Notions: Discretization and Time-Stepping

From now on, we will assume that Equation (5.7) (or, as necessary, Equation (3.1)) satisfies conditions for existence and uniqueness. We now consider the question of finding a numerical approximation to the unique solution of Equation (5.7), given an initial condition.

Intuitively, this is achieved by "discretizing time", and by using discrete time samples to approximate the derivative term $\frac{d}{dt}\vec{x}(t)$. This

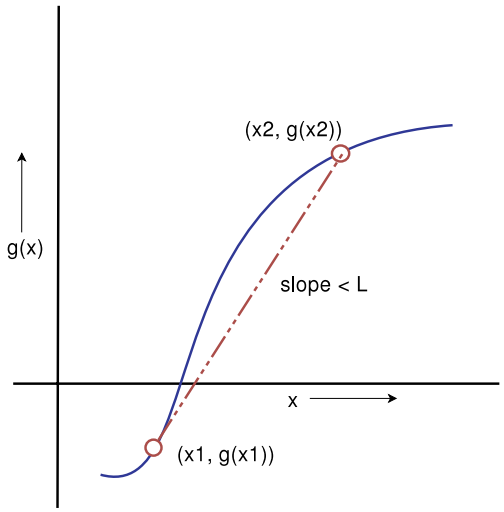


Fig. 5.1 Lipschitz continuity.

process approximates the ODE (Equation (5.7)) by a sequence of nonlinear equations in which the unknowns are the values of the time samples of $\vec{x}(t)$. These nonlinear equations are solved numerically using, e.g., the Newton–Raphson method of the previous section. The process can be summarized as follows:

(1) Suppose we are given an initial condition at $t_0 = 0$ and are interested in solving Equation (5.7) over some interval $[0,T]$. First, we partition $[0,T]$ using discrete time samples: $\{t_0 = 0, t_1, t_2, \ldots, t_{N-1}, t_N = T\}$. Our goal is to find numerical approximations to $\{\vec{x}(t_i), \quad i = 1, \ldots, N\}$.

(2) At each timepoint t_i, we approximate the derivative $\frac{d}{dt}\vec{x}(t)$ using the above samples of $\vec{x}(t)$. For example, one natural choice, motivated by first-principles definitions of a derivative, might be

$$\frac{d}{dt}\vec{x}(t_i) \simeq \frac{\vec{x}_i - \vec{x}_{i-1}}{t_i - t_{i-1}}, \tag{5.8}$$

where we have used the notation $\vec{x}_i \triangleq \vec{x}(t_i)$.

(3) For each t_i , we substitute Equation (5.8) in Equation (5.7), resulting in a sequence of *nonlinear algebraic equations*

for $i = 1, \ldots, N$:

$$\vec{g}_i(\vec{x}_i) \triangleq \frac{\vec{x}_i - \vec{x}_{i-1}}{t_i - t_{i-1}} + \vec{f}(\vec{x}_i) + \vec{b}(t_i) = \vec{0}. \tag{5.9}$$

(4) Next, we solve $\vec{g}_i(\vec{x}_i) = 0$ numerically (e.g., using Newton–Raphson), in the sequence $i = 1, \ldots, N$. Each solution yields $\vec{x}_i$ as a result.
Observe that for each i, $\vec{g}_i(\vec{x}_i)$ involves not only $\vec{x}_i$, but also $\vec{x}_{i-1}$; i.e., $\vec{x}(t)$ at the *previous* timepoint t_{i-1}. In particular, when we solve the very first equation $\vec{g}_1(\vec{x}_1) = \vec{0}$, we need x_0, the initial condition. It is in this manner that the initial condition (which we need to specify for existence and uniqueness via Picard–Lindelöf) concretely enters our numerical procedure and influences the results we obtain.
Next, for solving $\vec{g}_2(\vec{x}_2) = \vec{0}$ to find $\vec{x}_2$, we need $\vec{x}_1$, involved in the definition of $\vec{g}_2(\cdot)$; we have this available from solving the first equation. Similarly, for each succeeding i, when we solve $\vec{g}_i(\vec{x}_i) = \vec{0}$, we have $\vec{x}_{i-1}$ available from the solution of the previous equation. This process, in which $\vec{x}_i,\ i = 1, \ldots, N$ are solved for successively, is called *time-stepping*.

Step 2 of the time-stepping process above incorporated a specific way to approximate $\frac{d}{dt}\vec{x}(t)$ (for illustration), but this approximation step can be performed in a more general, systematic way, resulting in different numerical solution techniques with different computational characteristics.

5.3 Numerical Integration by Piecewise Polynomial Approximation

A general technique for approximating the derivative using time samples starts with a basic assumption: that the solution of Equation (5.7) can be approximated as a *piecewise polynomial* function. The concept is illustrated in Figure 5.2, where the black waveform represents the exact solution of Equation (5.7), and the blue and green waveforms represent piecewise linear and quadratic approximations, respectively. Each piecewise section can be expressed using an appropriate polynomial

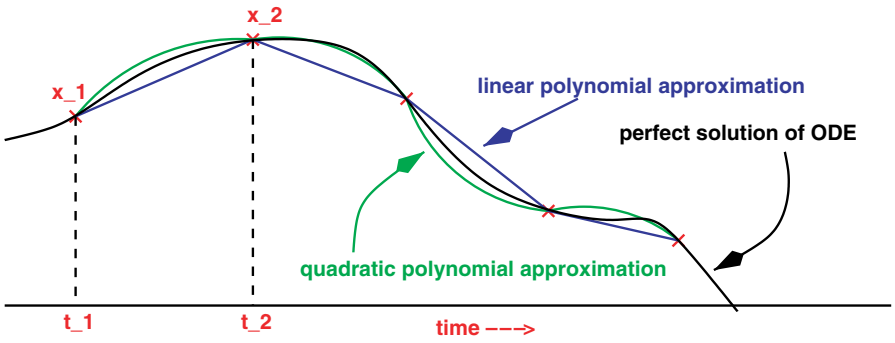


Fig. 5.2 Approximating $\vec{x}(t)$ using piecewise polynomials.

formula; for example, the formula for the first piecewise linear segment in Figure 5.2, between t_1 and t_2, is:

$$\vec{x}(t) \simeq \vec{x}_1 + \frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1}(t - t_1), \quad t \in [t_1, t_2]. \tag{5.10}$$

Such an assumption can be used to derive numerical integration formulas for solving Equation (5.7) (i.e., akin to $\vec{g}_i(\vec{x}_i)$ in Equation (5.9)), in two steps. First, an expression for $\frac{d}{dt}\vec{x}(t)$ that is consistent with the piecewise polynomial assumption is obtained. Second, the polynomial expressions for $\vec{x}(t)$ and $\frac{d}{dt}\vec{x}(t)$ are substituted into Equation (5.7), and a timepoint (or combination of timepoints) at which this equation is enforced is chosen.

5.3.1 Forward Euler, Backward Euler and Trapezoidal Methods

For example, in the case of the piecewise linear assumption of Equation (5.10), we have

$$\frac{d}{dt}\vec{x}(t) \simeq \frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1}, \quad t \in [t_1, t_2]; \tag{5.11}$$

substituting this in Equation (5.7), we obtain

$$\frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1} + \vec{f}(\vec{x}(t)) + \vec{b}(t) \simeq \vec{0}, \quad t \in [t_1, t_2]. \tag{5.12}$$

Assume that $\vec{x}_1$ is known (e.g., it is an initial condition, or has been previously solved for while time-stepping). In order to use Equation (5.12)

to obtain an equation for x_2, we can choose values of t to enforce Equation (5.12) at such that only $\vec{x}_2$ and $\vec{x}_1$ are involved. The only two choices are $t = t_1$ and $t = t_2$. For the former choice, we obtain

$$\vec{g}_{FE,2}(\vec{x}_2) \triangleq \frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1} + \vec{f}(\vec{x}_1) + \vec{b}(t_1) = \vec{0}. \tag{5.13}$$

The time-stepping technique that results from using Equation (5.13) in place of Equation (5.9) (observe that the two are not identical) is known as the **Forward Euler** (FE) method.

If, on the other hand, we choose to enforce Equation (5.12) at $t = t_2$, we obtain

$$\vec{g}_{BE,2}(\vec{x}_2) \triangleq \frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1} + \vec{f}(\vec{x}_2) + \vec{b}(t_2) = \vec{0}; \tag{5.14}$$

this is known as the **Backward Euler** (BE) method. Note that this is identical to Equation (5.9).

Comparing Equations (5.13) and (5.14), we immediately note a qualitative difference between the two. Equation (5.13) expresses the unknown $\vec{x}_2$ explicitly in terms of known quantities, i.e.,

$$\vec{x}_2 = \vec{x}_1 - (\vec{f}(\vec{x}_1) + \vec{b}(t_1))(t_2 - t_1), \tag{5.15}$$

whereas Equation (5.14) does not: it is an implicit equation in the unknown $\vec{x}_2$. Indeed, $\vec{g}_{BE,i}(\vec{x}_i)$ is nonlinear if $\vec{f}(\cdot)$ is nonlinear, hence solving for $\vec{x}_i$ requires a numerical technique such as the Newton–Raphson method. Therefore, Forward Euler is termed an **explicit method**, whereas Backward Euler is an **implicit method**.

FE therefore has the considerable computational advantage over BE that the unknown $\vec{x}_i$ is available immediately using Equation (5.13), without the need for solving nonlinear equations using Newton–Raphson. Unfortunately, this advantage is insignificant compared to two great disadvantages of FE (and explicit methods in general): numerical errors during time-stepping can accumulate to such an extent that they render the method useless in practice (a phenomenon termed *numerical instability*); and it does not generalize usefully to DAEs such as Equation (3.1). Backward Euler and many other implicit methods, although more computationally expensive than FE and other explicit methods, do not suffer from these problems. Indeed, BE is a good first

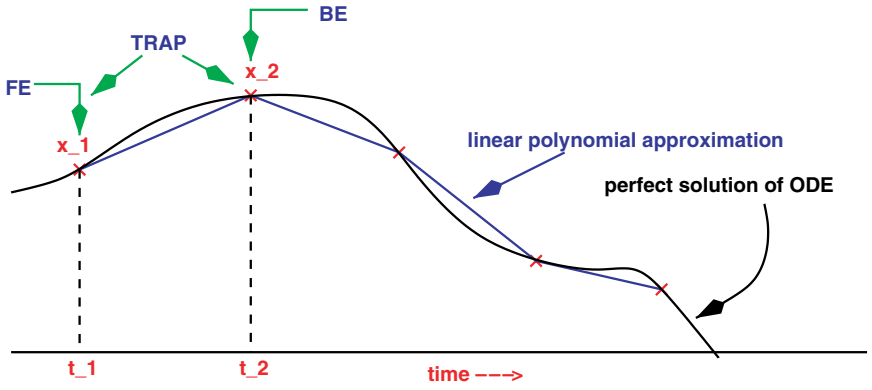


Fig. 5.3 FE, BE and TRAP methods.

choice of a simple-to-implement, numerically robust general purpose method.

It is also possible to combine Equations (5.13) and (5.14) to derive other numerical integration formulas. For example, by averaging Equations (5.13) and (5.14), we obtain

$$\vec{g}_{TRAP,2}(\vec{x}_2) \triangleq \frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1} + \frac{\vec{f}(\vec{x}_1) + \vec{f}(\vec{x}_2) + \vec{b}(t_1) + \vec{b}(t_2)}{2} = \vec{0}, \quad (5.16)$$

a numerical integration formula known as the **Trapezoidal method** (TRAP). Figure 5.3 summarizes how FE, BE and TRAP are all obtained from the piecewise linear assumption (Equation (5.10)) on $\vec{x}(t)$.

5.3.2 Linear Multistep Methods

Similarly, it is possible to derive many other numerical integration formulas, using piecewise quadratic, piecewise cubic, *etc.* assumptions on $\vec{x}(t)$, and enforcing them at different combinations of timepoints. In such higher-order formulas, more than two timepoints and time-samples can be involved.

In general, if we use a pth-order polynomial, $p+1$ timepoints are involved in a formula of the form:

$$\vec{g}_{LMS,n}(\vec{x}_n) \triangleq \sum_{i=0}^{p} \alpha_i \vec{x}_{n-i} + \sum_{i=0}^{p} \beta_i \left(\vec{f}(x_{n-i}) + \vec{b}(t_{n-i})\right) = \vec{0}, \quad (5.17)$$

where the unknown being solved for at timepoint n is $\vec{x}_n$, and p prior samples are used. Numerical integration formulas of this form are known as **Linear Multistep** (LMS) methods.

5.4 Accuracy and Numerical Stability

All numerical integration formulas inevitably involve approximations, which lead to some error or inaccuracy in the results they predict. To develop confidence in the usefulness of these results, it is important to obtain estimates of the error of a given numerical integration method; one of the reasons different numerical integration formulas are of interest is that they have different accuracy properties.

The investigation of error properties of a given numerical integration formula is typically split into two steps. The first is to estimate *local error*, i.e., the error incurred in a single timestep, assuming that the solution at the prior (known) timepoint was exact. The second step, assessing *global error* or *numerical stability* properties of the integration formula, explores the mechanism by which local errors accumulate over a long series of time steps. Although a general, systematic analysis of local and global error can be involved [11], it is not difficult to obtain an understanding of the basic ideas.

5.4.1 Local Truncation Error

The basic technique for estimating local error is to expand the exact solution of Equation (5.7) as a Taylor series in the timestep, do the same for the solution predicted by the numerical integration formula, and compare terms to determine which terms agree and which do not. Suppose our numerical integration's timestepping process is currently at time step $t_{i+1} = t_i + h$, and assume further that we know the exact value of the solution at time t_i — call this value $\vec{x}_i^*$. The exact solution at t_{i+1}, $\vec{x}_{i+1}^*$, can be written in Taylor series as:

$$\vec{x}_{i+1}^* = \vec{x}_i^* + h\frac{d\vec{x}^*(t_i)}{dt} + \frac{h^2}{2}\frac{d^2\vec{x}^*(t_i)}{dt^2} + \frac{h^3}{6}\frac{d^3\vec{x}^*(t_i)}{dt^3} + \cdots . \qquad (5.18)$$

Using Equation (5.7) to substitute for $\frac{d\vec{x}^*(t_i)}{dt}$, this can be rewritten as:

$$\vec{x}_{i+1}^* = \vec{x}_i^* - h\big(\vec{f}(\vec{x}_i^*) + \vec{b}_i\big) + \frac{h^2}{2}\frac{d^2\vec{x}^*(t_i)}{dt^2} + \frac{h^3}{6}\frac{d^3\vec{x}^*(t_i)}{dt^3} + \cdots . \qquad (5.19)$$

Equation (5.19) can be compared with the numerical integration formula, suitably rewritten, to understand the dominant terms of error. This is particularly easy for explicit integration methods[1]; for example, the FE formula (5.15) matches the first two terms of Equation (5.19) exactly, but differs in the higher-order terms. For small time steps, the error between the two is dominated by the first non-identical Taylor term, i.e.,

$$\epsilon_{LTE,FE} = \frac{h^2}{2}\frac{d^2\vec{x}^*(t_i)}{dt^2}. \tag{5.20}$$

This dominant error is termed the **Local Truncation Error** (LTE) of the numerical integration method. Observe that for FE, it is quadratic in the size of the time step, and depends also on the second derivative of the exact solution. If a given numerical integration formula has a higher order of dependence on h, then it is more accurate for small time steps than a method with a lower order of dependence. It can be shown that a p^{th}-order LTE term will involve the p^{th} derivative $\frac{d^p\vec{x}^*(t_i)}{dt^p}$. This also lowers the error in practice, because higher derivatives of smooth solutions tend to drop rapidly in magnitude. Moreover, it can also be shown that a properly set up p^{th}-order LMS formula Equation (5.17) will have an LTE expression where the dominant term is of at least $(p+1)^{\text{th}}$ order.

5.4.2 Numerical Stability and Global Error Accumulation

As we noted above, analysis of local error proceeds under the assumption that an exact solution was available at the previous timepoint. Other than for the very first step in a timestepping simulation (at which point an exact initial condition can be provided), this assumption is not valid, since each prior solution using the numerical method has, in fact, introduced errors. As the timestepping simulation proceeds timepoint by timepoint, these errors can compound and grow, sometimes so catastrophically that the procedure cannot provide any useful numerical approximation at all. How errors compound over many time

[1] It is also possible for implicit methods like FE and TRAP, but involves significantly more manipulation.

steps depends crucially on the choice of numerical integration method, in particular on its *numerical stability* properties.

5.4.2.1 Standard Linear Test Problem

Numerical stability analysis of a numerical integration method starts with the choice of a *test problem* whose exact solution is known analytically. The numerical integration method under consideration is applied to the test problem and expressions are obtained for its numerical solution. These expressions are compared against the known exact solution to assess its global error properties.

The standard test problem that is typically used for numerical stability analysis is the scalar, linear ODE

$$\dot{x} = \lambda x, \tag{5.21}$$

with initial condition x_0 at $t = 0$. This has the exact solution

$$x(t) = x_0 e^{\lambda t}. \tag{5.22}$$

If λ is real and negative, the solution is a decaying exponential, depicted in Figure 5.4. This is a qualitative feature for most physical applications: if a system is not excited externally, it tends to move to a rest state. The test problem (Equation (5.21)), in fact, captures the essential dynamics of all linear systems, since vector systems of linear differential equations can be represented (via eigendecomposition) as an interacting collection of such test problems with different values of λ. Equation (5.21) is also representative of the local dynamics of smooth nonlinear systems, since linearization can be used to represent their local dynamics approximately.

5.4.2.2 Numerical Stability Properties of FE, BE and TRAP

When applied to the test problem (Equation (5.21)), the FE method with uniform time steps $h = t_i - t_{i-1}$ reduces to

$$x_i = x_{i-1}(1 + h\lambda), \quad i = 1, \ldots, N. \tag{5.23}$$

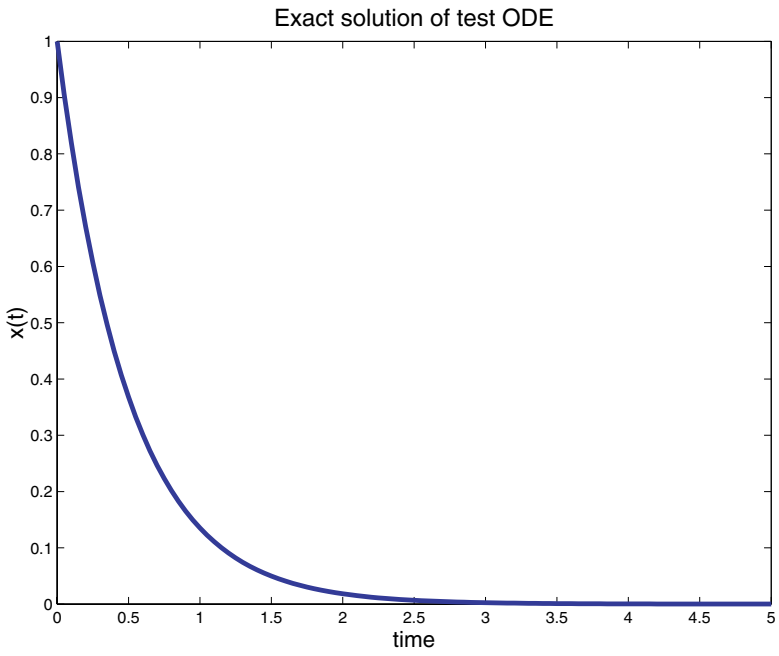


Fig. 5.4 Exact solution of linear test ODE for $\lambda < 0$.

This is a simple recurrence relation (difference equation) that can be solved immediately for x_i:

$$x_i = x_0(1 + h\lambda)^i, \quad i \geq 0. \tag{5.24}$$

Observe that the solution obtained by FE, given by Equation (5.24), decays to zero if, and only if, $|1 + h\lambda| < 1$. Even if $\Re(\lambda) < 0$ and the exact solution of the test problem, given by Equation (5.22), decays, it is possible for FE's solution to blow up, i.e., increase without limit with increasing i. This happens, for example, if $h > \frac{2}{|\lambda|}$, as depicted in Figure 5.5 with $\lambda = -1$. When this happens, the error of FE's solution grows rapidly, and indefinitely, without bound.

Thus, we see that even qualitatively, FE has a fundamental limitation: its accuracy degenerates catastrophically for step sizes larger than a critical value related to the reciprocal of λ. To better compare FE's qualitative behavior against that of the exact solution, allow λ to be complex (imaginary components of λ can result from

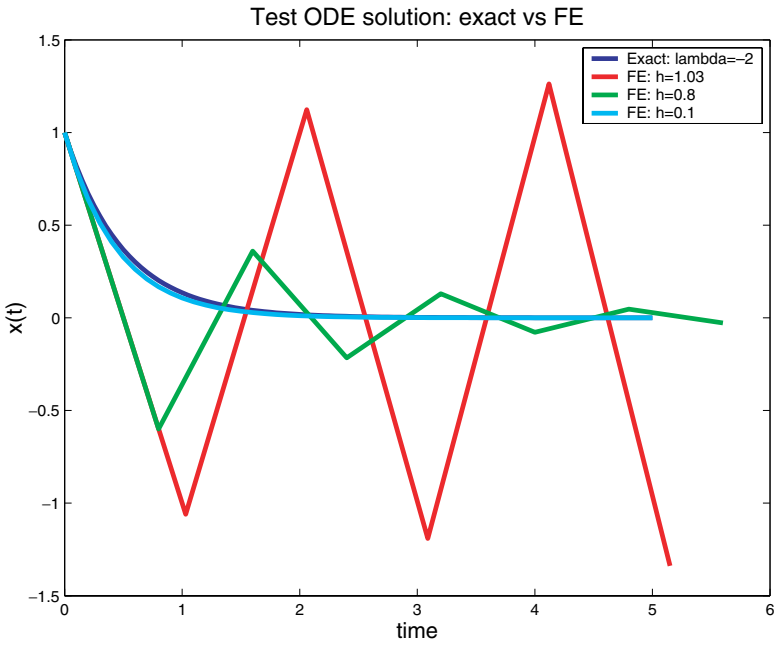


Fig. 5.5 Numerical instability of FE for $h > \frac{2}{|\lambda|}$.

eigendecomposition of vector linear ODEs). Figure 5.6 depicts how both FE and the exact solution behave, for different choices of h and λ conveniently represented as points in the $h\lambda$ plane. Observe that the qualitative behavior of FE matches that of the exact solution only in two regions: with the circle of radius 1 centered at -1, and on the entire right half plane. For points within the aforementioned circle, exact and FE solutions both decay; i.e., their qualitative behaviors match. For points on right half plane, with $\Re(\lambda) > 0$, both exact and FE solutions blow up; again, their qualitative behaviors match. However, for all points on the left half plane *not* in the aforementioned circle, FE's solution blows up, while the exact solution dies down — an egregiously inaccurate situation, in which FE is of no utility. This region is particularly important in applications, which tend to have $\Re(\lambda) < 0$.

Figure 5.6 establishes that the step sizes one can take with FE are limited to of the order of $\frac{1}{\lambda}$. Although this limitation is not an overwhelming one for *scalar* systems such as the test problem (reasonable

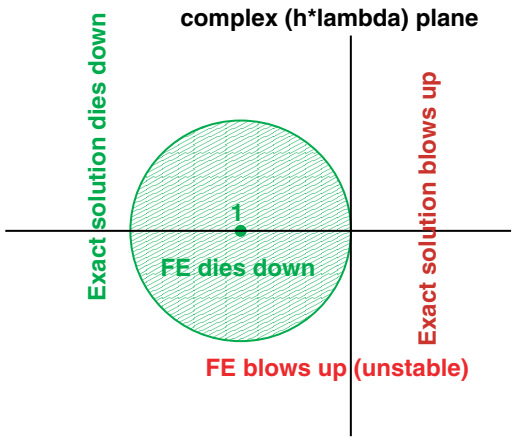


Fig. 5.6 Stability region of FE.

choices for step size are typically a tenth or a hundredth of $\frac{1}{\lambda}$), the situation is quite different for *vector* systems of differential equations, because solving a typical system of vector differential equations is effectively equivalent to simultaneously solving many copies of the scalar test problem, with the *same step size* for all copies. The step size must, therefore, be chosen to be in stability region of *all* copies of the test equation — i.e., $h < \frac{2}{|\lambda_{\max}|}$.

Typically, the values of λ over these copies of the test problem differ very widely; i.e., $\frac{|\lambda_{\max}|}{|\lambda_{\min}|}$ can be many orders of magnitude in size, with 10^6-10^{12} being typical for large circuits.[2] Because typical simulations need t to run at least as long the slowest dynamics in the system (characterized by time scales of the order of $\frac{1}{|\lambda_{\min}|}$), and FE's stability properties limit time steps to the order of $\frac{1}{|\lambda_{\max}|}$, millions or billions of steps are typically needed for a single simulation of a stiff system. This is so expensive computationally that FE becomes impractical in practice.

Having explored the stability properties of FE, an explicit method, it is instructive to do the same for BE, an implicit method. BE applied to the test problem yields the recurrence

$$x_i(1 - h\lambda) = x_{i-1}, \quad i = 1, \ldots, N, \tag{5.25}$$

[2] Such systems are known as *stiff* differential equations.

the solution of which is

$$x_i = x_0 \frac{1}{(1 - h\lambda)^i}, \quad i \geq 0. \tag{5.26}$$

Stability regions corresponding to Equation (5.26) are shown in Figure 5.7. Observe that the solution obtained by BE matches the exact solution qualitatively (i.e., both solutions decay) in the entire left half plane. In other words, if $\Re(\lambda) < 0$, BE can take time steps of any size without running into blow-ups. This is a very desirable feature for most applications. The only region where BE's solution *does not* match the exact solution qualitatively is the interior of the circle of radius 1, centered at 1; this is in right half plane. In this region, the exact solution blows up, while BE's solution decays; i.e., the error grows without bound. Since right half plane points ($\Re(\lambda) > 0$) do not arise very often in applications, this does not significantly limit BE's applicability.

Finally, if TRAP Equation (5.16), also an implicit method, is applied to the linear test problem, we obtain

$$x_i \frac{1 - h\lambda}{2} = x_{i-1} \frac{1 + h\lambda}{2}, \quad i = 1, \ldots, N, \tag{5.27}$$

or

$$x_i = x_0 \left(\frac{1 + h\lambda}{1 - h\lambda} \right)^i, \quad i \geq 0. \tag{5.28}$$

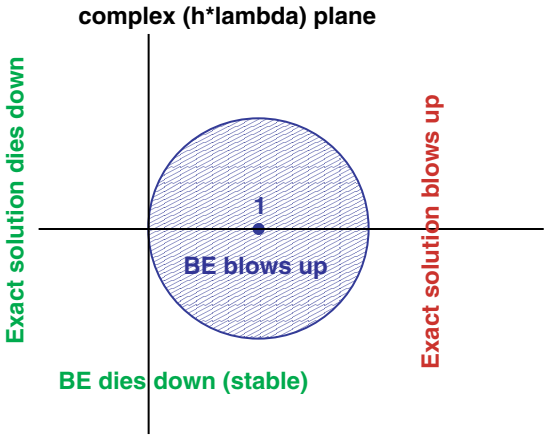


Fig. 5.7 Stability regions for BE.

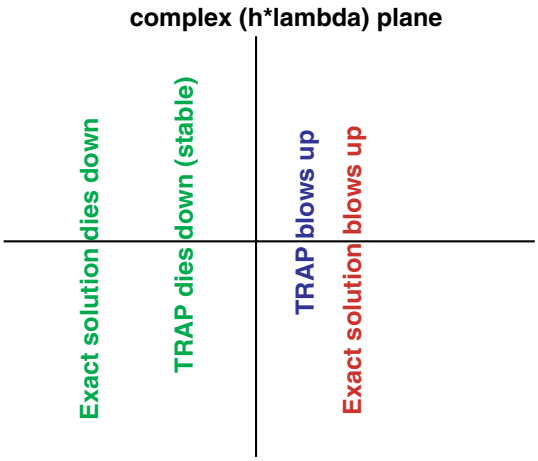


Fig. 5.8 Stability regions for TRAP.

As might perhaps be surmised from the fact that TRAP was obtained by averaging FE and BE, its stability properties are also, loosely speaking, an average of those of FE and FE. TRAP's stability regions are shown in Figure 5.8. We notice immediately that TRAP features *perfect stability* — i.e., the solution predicted by TRAP always matches that of the exact solution qualitatively, regardless of the value of λ and the choice of timestep h. This is possibly even better than BE's stability, though as we have already noted, errors limited to the right half plane do not constitute a significant limitation in practice.

5.4.2.3 Global Error for the Linear Test Problem

The above exploration of stability focussed on qualitative comparisons (i.e., blow-up versus decay) between the exact and numerical solutions of the test problem. It is instructive to also compare them in detail.

Expanding the exact solution (Equation (5.22)) at timepoint t_n in Taylor series, we obtain

$$x_n^* = x^*(t_n) = x^*(nh) = x_0 e^{\lambda nh} = x_0 \left(1 + nh\lambda + \frac{n^2h^2\lambda^2}{2!} + \cdots\right); \tag{5.29}$$

doing the same for FE (Equation (5.24)), BE (Equation (5.26)), and TRAP (Equation (5.28)), we obtain

$$x_{FE}(nh) = x_0\left(1 + nh\lambda + \frac{n(n-1)}{2}h^2\lambda^2 + \cdots\right), \tag{5.30}$$

$$x_{BE}(nh) = x_0\left(1 + nh\lambda + \frac{n(n+1)}{2}h^2\lambda^2 + \cdots\right), \quad \text{and} \tag{5.31}$$

$$x_{TRAP}(nh) = x_0\left(1 + nh\lambda + \frac{n^2}{2}h^2\lambda^2 + \cdots\right), \tag{5.32}$$

respectively. Note that the FE and BE expansions match the exact solution's expansion up to first order, but differ in the quadratic term. analysis of their LTE. TRAP, however, matches both linear and quadratic terms of the exact solutions expansion; it differs starting only from the cubic term. Therefore, we see in addition to excellent stability properties, TRAP has a higher order of accuracy than BE. It is not surprising, therefore, that TRAP is typically the method of choice for if a low-order ($p = 1$ in Equation (5.17)) LMS method is desired.

5.5 Adapting ODE Methods to Solve DAEs

Despite the fact that theory and numerical methods for DAEs (Equation (3.1)) are far more involved than for ODEs (Equation (5.7)), it is possible to adapt many numerical integration formulas for ODEs to solve DAEs successfully in practice. The key idea is to substitute $\frac{d}{dt}\vec{q}(\vec{x}(t))$ in place of $\frac{d}{dt}\vec{x}(t)$ in all the derivations of the previous sections. As a result, the general LMS formula for ODEs (Equation (5.17)) changes to

$$\vec{g}_{LMS,n}(\vec{x}_n) \triangleq \sum_{i=0}^{p} \alpha_i \vec{q}(\vec{x}_{n-i}) + \sum_{i=0}^{p} \beta_i \big(\vec{f}(x_{n-i}) + \vec{b}(t_{n-i})\big) = \vec{0}. \tag{5.33}$$

FE, BE and TRAP, in particular, become

$$\vec{g}_{FE,n}(\vec{x}_n) \triangleq \frac{\vec{q}(\vec{x}_n) - \vec{q}(\vec{x}_{n-1})}{t_n - t_{n-1}} + \vec{f}(\vec{x}_{n-1}) + \vec{b}(t_{n-1}) = \vec{0}, \tag{5.34}$$

$$\vec{g}_{BE,n}(\vec{x}_n) \triangleq \frac{\vec{q}(\vec{x}_n) - \vec{q}(\vec{x}_{n-1})}{t_n - t_{n-1}} + \vec{f}(\vec{x}_n) + \vec{b}(t_n) = \vec{0}, \quad \text{and} \quad (5.35)$$

$$\vec{g}_{TRAP,n}(\vec{x}_n) \triangleq \frac{\vec{q}(\vec{x}_n) - \vec{q}(\vec{x}_{n-1})}{t_n - t_{n-1}} + \frac{\vec{f}(\vec{x}_{n-1}) + \vec{b}(t_{n-1}) + \vec{f}(\vec{x}_n) + \vec{b}(t_n)}{2} = \vec{0}, \quad (5.36)$$

respectively. Observe that the appearance of $\vec{q}(\vec{x})$ in place of $\vec{x}$ in the derivative approximation terms does not fundamentally change the time-stepping process for BE and TRAP, which are both implicit methods that require Newton–Raphson solution at each timestep.

However, for FE, a numerical integration method in which the solution $\vec{x}_n$ was explicitly available in the ODE case, substitution by $\vec{q}(\vec{x}_n)$ makes an enormous difference. The unknown $\vec{x}_n$ is no longer explicitly available; only $\vec{q}(\vec{x}_n)$ is, hence $\vec{q}(\cdot)$ needs to be inverted to obtain $\vec{x}_n$. However, for any "true" DAE, $\vec{q}(\cdot)$ is not invertible, essentially because some equations in the DAE are algebraic and involve no derivatives, hence those components of $\vec{q}(\cdot)$ are identically zero. As a result, FE cannot be used *at all* for non-trivial DAEs; this is another crucial limitation of the method. Note that this limitation does not apply to BE or TRAP since they involve finding inverses of linear combinations of $\vec{q}(\cdot)$ and $\vec{f}(\cdot)$, which are typically invertible even when $\vec{q}(\cdot)$ is not.

6

Modelling Biochemical Reaction Kinetics Deterministically

In Sections 2 and 3, we formulated electronic circuits as nonlinear differential–algebraic equations; following which, in Chapters 4 and 5, we examined techniques for solving DAEs numerically for quiescent and transient solutions, respectively. Differential–algebraic equation formulations are, of course, not limited to electronic circuit applications — indeed, they are used in many domains, including mechanics, optics, chemistry and biology.

Biochemical reactions can be modelled at varying levels of detail. *Molecular dynamics* (MD) models keep track of the location/movement of individual molecules or atoms within a three-dimensional volume. Reactions take place when collisions, between the right combinations of molecules in the right orientations and velocities,[1] occur. MD simulations, which are extremely expensive computationally, are indispensable for understanding and predicting certain crucial biological phenomena (such as the detailed dynamics of ion channels in cell membranes). Many reactions can be modeled usefully at a coarser level

[1] In addition, inherent quantum-mechanical randomness modulates whether a reaction takes place or not.

than MD, keeping track only of the total numbers (populations) of molecules/atoms of different reactants, while abstracting away details of 3D position, velocities, momenta, *etc.* These simplifications rely on the so-called *well-mixed* assumption, i.e., that reactants/reactions are distributed in space in a homogeneous manner. The rôle of positions/velocities/orientations in determining the occurrence of reactions is captured using simple probabilistic models, in which the probability that a given reaction will occur depends only on environment-dependent constants characterizing the reaction, as well as the instantaneous populations of the reactants involved. Such *stochastic reaction models* — which keep track of discrete populations and discrete reaction events in time — involve far less computation than MD models, yet provide statistical/probabilistic information about reaction progress. This is a strength of stochastic reaction models, especially when reactant populations are small.

When large numbers of reactant molecules are involved, it is possible to model reactions usefully at an even coarser level, by keeping track only of *average reactant concentrations* as they change continuously with time. Such *reaction rate equation* (RRE) models use deterministic DAEs/ODEs to abstract away the probabilistic details of reactions completely; as such, they are far less computationally demanding than stochastic or MD models. RRE models, which were among the first mathematical models of reaction kinetics to be established (using an intuitive empirical principle known as the *law of mass action*), are the subject of this chapter.

It is worth noting at the outset that although RRE models of reactions have the same canonical form (Equation (3.1)) as electronic circuit equations, there are notable qualitative differences in internal structure that affect the applicability/performance of numerical algorithms for their solution. In particular, though RRE models are typically written in vector ODE form, some of their ODEs are in a sense redundant due to a fundamental property of reactions known as *conservation*. Because of conservation, ODE RRE models are really DAEs with special structure: they are composed of some "essential" nonlinear ODE components, augmented by algebraic linear equations (called conservation laws). Because of this feature, standard methods for computing

quiescent steady states like Newton–Raphson (Section 4) typically fail on RRE models, unless special steps are taken.

In this section, we will model reactions using RREs, starting from simple unimolecular reactions and progressing to general chains of reactions. We will then examine and analyse a special two-reaction chain, the *enzyme-catalyzed reaction*, that is important in biological applications.[2]

6.1 Unimolecular Reactions: $A \rightleftharpoons B$

Possibly, the simplest type of reaction is one in which a single reactant (molecule type) A changes spontaneously to another reactant B, and vice versa. The reaction is typically written as:

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B. \tag{6.1}$$

The constants k_1 and k_2, known as the *forward and reverse reaction rate constants*, play key roles in determining how quickly A converts to B, and vice versa.

The dynamics of reactions are quantified using a principle known as the *law of mass action*. For reactions such as Equation (6.1), which involve only one molecule of each reactant, the law of mass action stipulates that the rate at which a reaction proceeds is proportional to the concentration of each reactant involved. In other words, the rate of the forward reaction $A \rightharpoonup B$ is proportional to the concentration of A, i.e.,

$$R_f([A]) = k_1[A], \tag{6.2}$$

where $[A]$ denotes the concentration of A (typically in units of moles/litre). k_1, the forward rate constant, serves as the proportionality constant. Similarly, the reverse reaction $A \leftharpoonup B$ has the rate:

$$R_r([B]) = k_2[B]. \tag{6.3}$$

Since A is consumed by the forward reaction with rate $R_f([A])$, and generated by the reverse reaction with rate $R_r([B])$, its concentration

[2] Enzyme-catalyzed reactions are fundamental building blocks ubiquitous in metabolic and signalling pathways; see, e.g., the KEGG database [40].

changes as:

$$\frac{d}{dt}[A] = -R_f([A]) + R_r([B]) = -k_1[A] + k_2[B]. \tag{6.4}$$

Likewise, the rate of change of $[B]$ is given by

$$\frac{d}{dt}[B] = R_f([A]) - R_r([B]) = k_1[A] - k_2[B]. \tag{6.5}$$

Equations (6.4) and (6.5) are the *reaction rate equations* (RRE) for the reaction Equation (6.1). They can be expressed in vector form as:

$$\frac{d}{dt}\vec{x} = \vec{g}(\vec{x}) \triangleq G\vec{x}, \tag{6.6}$$

where

$$\vec{x}(t) = \begin{pmatrix} [A](t) \\ [B](t) \end{pmatrix} \quad \text{and } G = \begin{bmatrix} -k_1 & k_2 \\ k_1 & -k_2 \end{bmatrix}. \tag{6.7}$$

Observe that Equation (6.6) is a *linear* system of ODEs.

6.1.1 Conservation: Reducing RRE Size

Equation (6.1) indicates that when a forward reaction occurs, one molecule of A turns into a molecule of B; and vice versa when the reverse reaction occurs. In other words, the total number of molecules does not change, i.e., is *conserved*. Conservation can be expressed using concentrations as:

$$[A](t) + [B](t) = \text{constant} = [A]_0 + [B]_0, \tag{6.8}$$

where $[A]_0$ and $[B]_0$ denote the concentrations of A and B, respectively, at time $t = 0$.

Conservation can be inferred directly from the reaction rate equations. Adding Equations (6.4) and (6.5) results in

$$\frac{d}{dt}\left([A] + [B]\right) = 0, \tag{6.9}$$

which implies Equation (6.8).

The conservation equation (6.8) can be used to eliminate an unknown and hence reduce the size of the RREs. For example, expressing $[B](t)$ in terms of $[A](t)$ using Equation (6.8), and substituting in Equation (6.4), results in

$$\frac{d}{dt}[A] = -(k_1 + k_2)[A] + k_2([A]_0 + [B]_0), \tag{6.10}$$

a *scalar* linear differential equation in $[A](t)$.

6.1.2 Equilibrium Solution

The quiescent steady state of Equation (6.10), also termed the *equilibrium state* of A and denoted by $[A]_\infty$, can be obtained by setting $\frac{d}{dt}[A] = 0$. It is

$$[A]_\infty = \frac{k_2}{k_1 + k_2}([A]_0 + [B]_0). \tag{6.11}$$

$[B]_\infty$ can be obtained from Equations (6.11) and (6.8); it is

$$[B]_\infty = \frac{k_1}{k_1 + k_2}([A]_0 + [B]_0). \tag{6.12}$$

Observe that Equation (6.12) implies that if $k_1 \gg k_2$, then $[A]_\infty \simeq 0$, i.e., that A gets converted to B almost entirely. This matches the intuitive expectation that if the forward rate constant is is much larger than the reverse, the forward reaction dominates and A is consumed. Similarly, using Equation (6.11), if $k_2 \gg k_1$, the reverse reaction dominates and B gets consumed.

Note that the equilibrium concentrations of A and B *cannot* immediately be obtained by setting $\frac{d}{dt}\vec{x}(t) = 0$ in Equation (6.6); this results in $G\vec{x}_\infty = \vec{0}$, which has the trivial solution $\vec{x}_\infty = \vec{0}$. However, G is singular (the second row being the negation of the first); as a result, nontrivial solutions respecting Equation (6.8) also exist. The existence of such non-trivial solutions is of fundamental importance in reaction kinetics.

6.1.3 Transient Solution

Equation (6.10) is a scalar linear ODE that is readily solved analytically; its solution is

$$[A](t) = ([A]_0 - [A]_\infty)e^{-(k_1+k_2)t} + [A]_\infty. \tag{6.13}$$

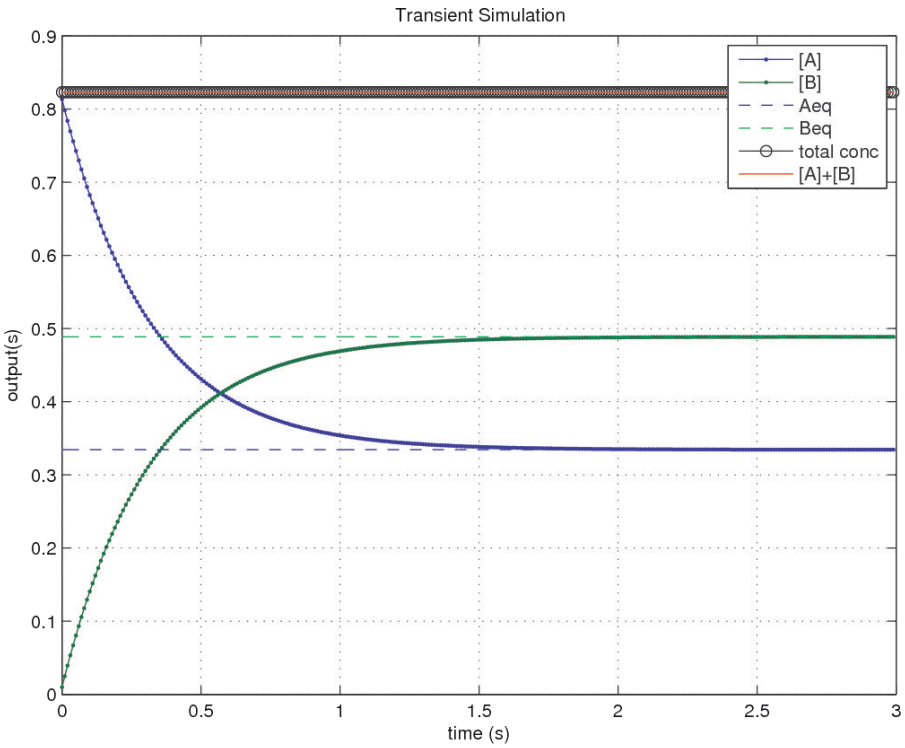


Fig. 6.1 Transient solution of unimolecular reaction rate equations.

Similarly, $[B](t)$ can be obtained (e.g., using Equations (6.13) and (6.8)) to be

$$[B](t) = ([B]_0 - [B]_\infty)e^{-(k_1+k_2)t} + [B]_\infty. \tag{6.14}$$

These solutions can also be obtained numerically, of course, using the techniques of Section 5; results of a transient simulation of Equation (6.6) are shown in Figure 6.1. Note that the higher equilibrium level of B indicates that $k_1 > k_2$ in this simulation.

6.2 Bimolecular Reactions: $A + B \rightleftharpoons C$

We observed above that the unimolecular reaction rate equation Equation (6.6) features a $\vec{g}(\cdot)$ with a singular Jacobian matrix G;

furthermore, $\vec{g}(\cdot)$ is linear. It turns out that the first property holds generically for reactions, but the second does not.

Consider a simple reversible bimolecular reaction, in which one molecule each of A and B react to form one molecule of C in the forward reaction, and vice versa in the reverse reaction:

$$A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C. \tag{6.15}$$

The law of mass action for Equation (6.15) manifests itself as:

$$R_f([A],[B]) = k_1[A][B], \quad R_r([C]) = k_2[C], \tag{6.16}$$

resulting in the reaction rate equations:

$$\begin{aligned}
\frac{d}{dt}[A](t) &= -R_f([A],[B]) + R_r([C]) = -k_1[A][B] + k_2[C], \\
\frac{d}{dt}[B](t) &= -R_f([A],[B]) + R_r([C]) = -k_1[A][B] + k_2[C], \\
\frac{d}{dt}[C](t) &= -R_r([C]) + R_f([A],[B]) = -k_2[C] + k_1[A][B].
\end{aligned} \tag{6.17}$$

Note that Equation (6.17), a vector ODE system of size 3, is *polynomial*, involving products of $[A]$ and $[B]$.

Conservation for Equation (6.15) is not as simple as that for the unimolecular case (Equation (6.1)). The total number of molecules of A, B and C is not conserved, since in the forward reaction, two molecules are consumed to produce only one; and vice versa for the reverse reaction. However, one molecule each of A and B must be consumed (or created) simultaneously; at the same time, a molecule of C must be created (or consumed). These observations lead to two conservation laws for Equation (6.15). Indeed, the conservation laws can be identified from Equation (6.17), by noting that:

$$\begin{aligned}
\frac{d}{dt}([A] - [B]) &= 0, \\
\frac{d}{dt}([A] + [C]) &= 0,
\end{aligned} \tag{6.18}$$

implying that

$$[A](t) - [B](t) = \text{constant} = [A]_0 - [B]_0, \tag{6.19}$$

$$[A](t) + [C](t) = \text{constant} = [A]_0 + [C]_0. \tag{6.20}$$

Equations (6.19) and (6.20) can be used to eliminate two of the three unknowns in Equation (6.17). For example, eliminating $[B]$ and $[C]$ leads to

$$\begin{aligned}\frac{d}{dt}[A](t) &= -k_1[A]([A] - [A]_0 + [B]_0) + k_2([A]_0 + [C]_0 - [A]) \\ &= -k_1[A]^2 - (k_2 + k_1([B]_0 - [A]_0))[A] + k_2([A]_0 + [C]_0),\end{aligned} \tag{6.21}$$

i.e., a scalar, nonlinear ODE in $[A](t)$.

The equilibrium state $[A]_\infty$ can be solved for by setting $\frac{d}{dt}[A](t) = 0$ in Equation (6.21) and solving a quadratic equation analytically. Figure 6.2 shows a numerical transient solution of Equation (6.17). Note that since Equation (6.21) is a Riccati equation with constant

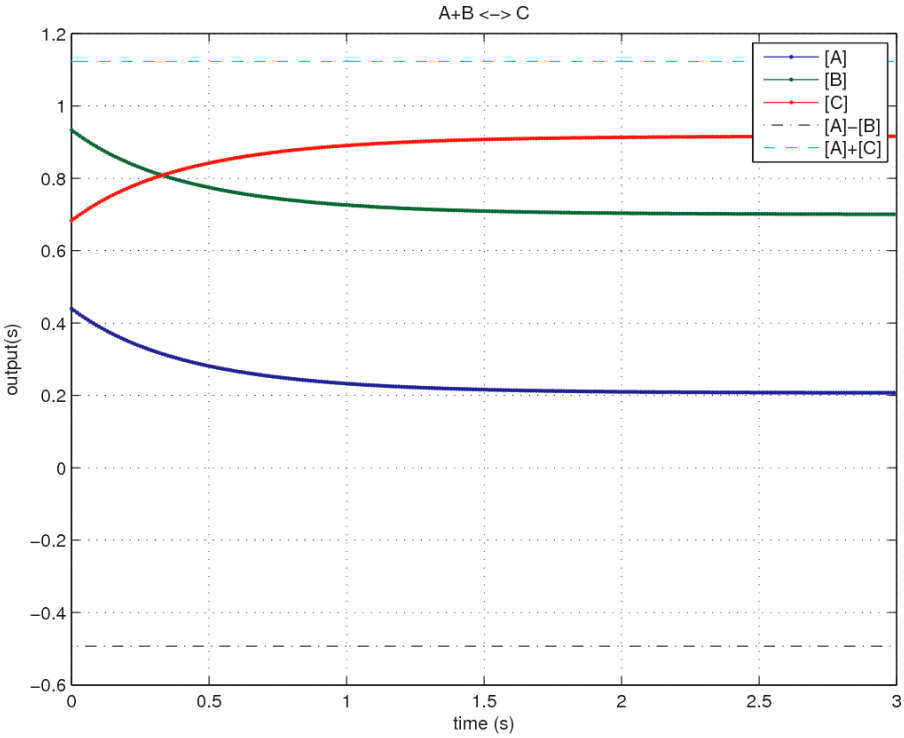


Fig. 6.2 RREs of $A + B \rightleftharpoons C$: transient solution.

coefficients [38], an analytical solution for the transient dynamics of Equation (6.15) is also available.

6.3 Nontrivial Stoichiometries: $s_A A + s_B B \rightleftharpoons s_C C + s_D D$

Reactions may, of course, involve more than one molecule of each reactant, for example

$$2A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C + 3D, \tag{6.22}$$

or

$$s_A A + s_B B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} s_C C + s_D D, \tag{6.23}$$

where s_A, s_B s_C and s_D, termed *stoichiometric coefficients*, are positive integers. The law of mass action applied to Equation (6.23) leads to the reaction rates[3]

$$\begin{aligned} R_f([A],[B]) &= k_1[A]^{s_A}[B]^{s_B}, \\ R_r([C],[D]) &= k_2[C]^{s_C}[D]^{s_D}. \end{aligned} \tag{6.24}$$

Note that when a forward (reverse) reaction occurs, s_A molecules of A and s_B molecules of B are consumed (created), while s_C molecules of C and s_D molecules of D are created (consumed). This leads to the rate equations

$$\begin{aligned} \frac{d}{dt}[A](t) &= s_A\left[-R_f([A],[B]) + R_r([C],[D])\right], \\ \frac{d}{dt}[B](t) &= s_B\left[-R_f([A],[B]) + R_r([C],[D])\right], \\ \frac{d}{dt}[C](t) &= s_C\left[+R_f([A],[B]) - R_r([C],[D])\right], \\ \frac{d}{dt}[D](t) &= s_D\left[+R_f([A],[B]) - R_r([C],[D])\right]. \end{aligned} \tag{6.25}$$

[3] Note that s_A copies of A, s_B copies of B, *etc.* are involved in the reaction.

Conservation laws for Equation (6.23) can be inferred from Equation (6.25) by noting that

$$\begin{aligned}
\frac{d}{dt}\left(\frac{[A]}{s_A}-\frac{[B]}{s_B}\right)=0 &\Rightarrow \frac{[A]}{s_A}-\frac{[B]}{s_B}=\frac{[A]_0}{s_A}-\frac{[B]_0}{s_B},\\
\frac{d}{dt}\left(\frac{[A]}{s_A}+\frac{[C]}{s_C}\right)=0 &\Rightarrow \frac{[A]}{s_A}+\frac{[C]}{s_C}=\frac{[A]_0}{s_A}+\frac{[C]_0}{s_C},\\
\frac{d}{dt}\left(\frac{[A]}{s_A}+\frac{[D]}{s_D}\right)=0 &\Rightarrow \frac{[A]}{s_A}+\frac{[D]}{s_D}=\frac{[A]_0}{s_A}+\frac{[D]_0}{s_D}.
\end{aligned} \tag{6.26}$$

The conservation equation (6.26) can be used to eliminate three of the four unknowns in Equation (6.25) to obtain, e.g., a scalar nonlinear ODE for $[A](t)$. However, in general, analytical solutions are not available, since higher-degree polynomials in $[A]$ are involved; hence numerical integration techniques, such as those in Section 5, are needed. Figure 6.3 depicts a transient solution of the RREs of the reaction Equation (6.22).

6.3.1 Stoichiometry Matrices: Net Reaction Rates

Note that Equation (6.25) can be written as:

$$\frac{d}{dt}\underbrace{\begin{bmatrix}[A]\\[B]\\[C]\\[D]\end{bmatrix}}_{\vec{x}(t)}=\underbrace{\begin{bmatrix}-s_A\\-s_B\\s_C\\s_D\end{bmatrix}}_{S}\underbrace{\left(R_f([A],[B])-R_r([C],[D])\right)}_{r(\vec{x})}. \tag{6.27}$$

$r(\vec{x})$, a nonlinear function of the components of $\vec{x}$, is the *net reaction rate*, i.e., the difference between the forward and reverse rates of the reaction Equation (6.23). S is known as the *stoichiometry matrix*[4] of the reaction. Writing reaction rate equations in the form Equation (6.27), i.e., using a stoichiometry matrix multiplying a vector of net reaction rates, makes it convenient to obtain a general procedure for identifying conservation laws, as we shall see later.

[4] for single reactions like Equation (6.23), S reduces to a column vector.

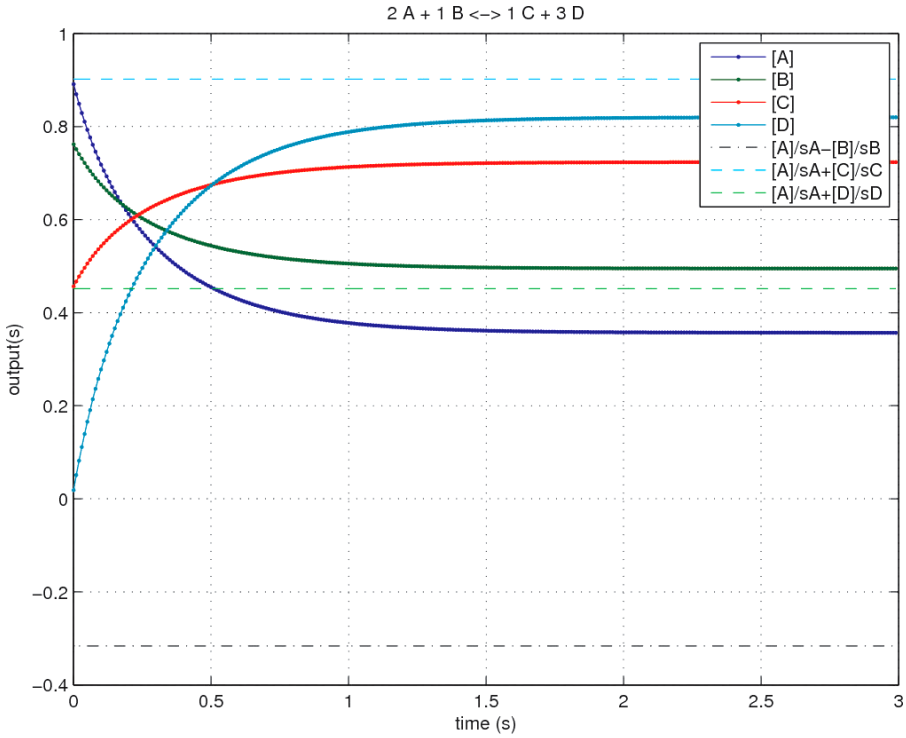


Fig. 6.3 $2A + B \rightleftharpoons C + 3D$: transient solution.

6.3.2 Kinetics of Scaled Reactions are Not Identical

From the reaction-rate expressions Equation (6.24) for the reaction Equation (6.23), note that the kinetics of, e.g.,

$$A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C + D \tag{6.28}$$

are *not* identical to those of the "scaled" reaction (with doubled quantities)

$$2A + 2B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} 2C + 2D. \tag{6.29}$$

Physically, the difference stems from the fact that in the reaction Equation (6.29), *two molecules each* of A and B need to come together

to initiate a forward reaction; a situation quite different from Equation (6.28), where only one of each needs to come together. As a result, the two are fundamentally different reactions, even though they involve the same reactants in scaled quantities. Hence, it is only to be expected that their kinetics differ.

6.4 Reaction Chains: $A + B \rightleftharpoons C, C + D \rightleftharpoons E + A$

So far, we have been concerned with only one reaction. In practice, of course, different reactions can take place simultaneously and involve common reactants.

Consider, for example, the system (or chain) of two reactions

$$\text{Reaction 1:} \quad A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C, \tag{6.30}$$

$$\text{Reaction 2:} \quad C + D \underset{k_4}{\overset{k_3}{\rightleftharpoons}} E + A, \tag{6.31}$$

with five reactants A, B, C, D and E. Applying the law of mass action, the net reaction rates of Equations (6.30) and (6.31) are:

$$\vec{r}(\vec{x}) \triangleq \begin{bmatrix} r_1(\vec{x}) \\ r_2(\vec{x}) \end{bmatrix} = \begin{bmatrix} k_1[A][B] - k_2[C] \\ k_2[C][D] - k_4[E][A] \end{bmatrix}, \tag{6.32}$$

where $\vec{x} = [[A], [B], [C], [D], [E]]^T$ is the vector of reactant concentrations.

Observe that A and C are involved in both reactions; hence the rates of change of their concentrations depend on the net reaction rates of both reactions, i.e.,

$$\frac{d}{dt}[A] = -r_1(\vec{x}) + r_2(\vec{x}), \quad \frac{d}{dt}[C] = r_1(\vec{x}) - r_2(\vec{x}). \tag{6.33}$$

The rates of change of the remaining reactants (B, D and E) are governed by the net reaction rate of only one reaction:

$$\frac{d}{dt}[B] = -r_1(\vec{x}), \quad \frac{d}{dt}[D] = -r_2(\vec{x}), \quad \frac{d}{dt}[E] = r_2(\vec{x}). \tag{6.34}$$

Putting Equations (6.33) and (6.34) together, we obtain the reaction rate equations for the reaction chain (Equations (6.30)–(6.31)) to be

$$\frac{d}{dt}\underbrace{\begin{bmatrix}[A]\\[B]\\[C]\\[D]\\[E]\end{bmatrix}}_{\vec{x}(t)} = \vec{g}(\vec{x}) \triangleq \underbrace{\begin{bmatrix}-1 & 1\\ -1 & \\ 1 & -1\\ & -1\\ & 1\end{bmatrix}}_{S}\underbrace{\begin{bmatrix}r_1(\vec{x})\\ r_2(\vec{x})\end{bmatrix}}_{\vec{r}(\vec{x})}. \tag{6.35}$$

Denoting by $n = 5$ the number of reactants, and by $m = 2$ the number of reactions, observe that S, the stoichiometry matrix, is of size $n \times m$, i.e., it is *"tall/thin" rectangular*. The vector of net reaction rates, $\vec{r}(\vec{x})$, has dimension m, while the vector of reactant concentrations, $\vec{x}$, has dimension n.

The structure of Equation (6.35) makes it clear why the Jacobian matrix of $\vec{g}(.)$ is necessarily singular. The Jacobian of $\vec{g}(.)$ can be expressed as:

$$J_g(\vec{x}) \triangleq \underbrace{\frac{d\vec{g}(\vec{x})}{d\vec{x}}}_{5\times 5} = \underbrace{S}_{5\times 2}\underbrace{\frac{d\vec{r}(\vec{x})}{d\vec{x}}}_{2\times 5}, \tag{6.36}$$

i.e., it is the product of a 5×2-matrix and a 2×5-matrix. Hence, its maximum rank is $m = 2$. Because the Jacobian is rank deficient (hence singular), attempting to solve Equation (6.35) for a quiescent steady state using Newton–Raphson (Section 4) is guaranteed to fail right at the start. One remedy, discussed in the next section, is to identify conservation laws and apply them to reduce Equation (6.35) to a size-m system which features a nonsingular QSS Jacobian matrix.

Transient simulation does not suffer from this Jacobian singularity limitation, since the Jacobian matrix of Equation (5.17) is $\beta_0 J_g(\vec{x}) + \alpha_0 I_{5\times 5}$; the identity matrix component, which dominates for small time steps, makes the transient Jacobian nonsingular. Figure 6.4 shows the results of a transient simulation of Equation (6.35). Observe how — unlike for the single-reaction cases examined previously — some of the concentrations change non-monotonically with time.

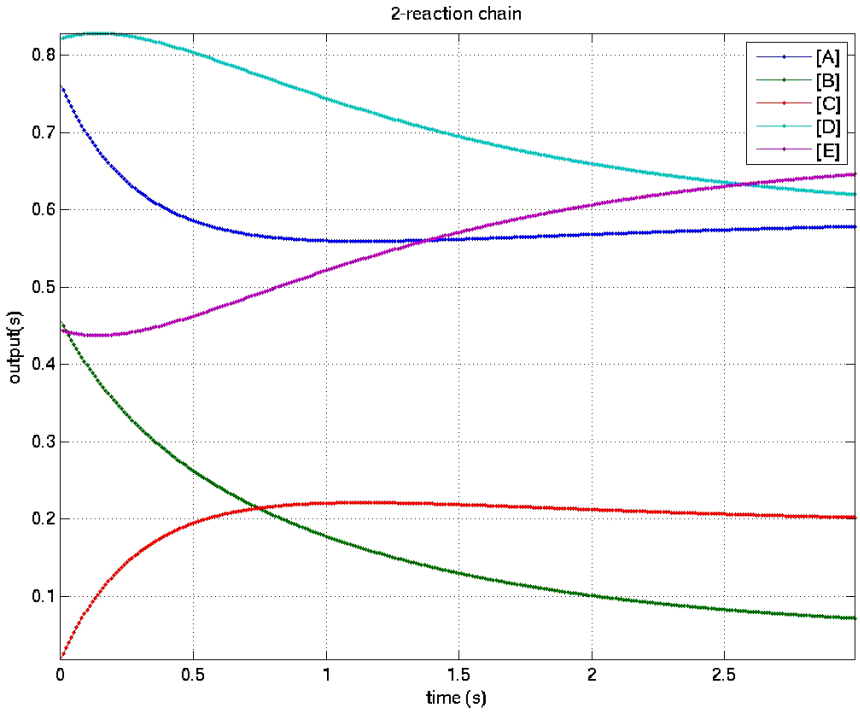


Fig. 6.4 Two-reaction chain $A + B \rightleftharpoons C, C + D \rightleftharpoons E + A$: transient simulation.

Observe also that conservation laws can be identified from examination of the stoichiometry matrix S alone. For example, adding rows 1 and 3 of S results in $\frac{d}{dt}([A] + [C]) = 0$; adding rows 4 and 5 in $\frac{d}{dt}([D] + [E]) = 0$; subtracting row 2 from row 1, and adding row 4, results in $\frac{d}{dt}([A] - [B] + [D]) = 0$.

6.5 Identifying Conservation Laws Numerically Using SVDs

While conservation laws can often be identified by inspection for simple reaction systems, as we have seen above, it is useful to set up a systematic procedure by which they can be obtained for any system of m reactions involving n reactants. Possibly, the most straightforward way involves applying the *singular value decomposition* (SVD) to the stoichiometry matrix, though other techniques are also available.[5]

[5] e.g., a more computationally efficient scheme uses rectangular LU factorization.

Singular value decomposition (e.g., [86]) can be applied to any matrix S, square or rectangular, to decompose it into the form:

$$\underbrace{S}_{n\times m} = \underbrace{U}_{n\times n}\,\underbrace{\Sigma}_{n\times m}\,\underbrace{V^T}_{m\times m}, \tag{6.37}$$

where U and V are square, *unitary* matrices (i.e., $UU^T = U^TU = I_{n\times n}$ and $VV^T = V^TV = I_{m\times m}$) and Σ is a *diagonal* $n \times m$ *matrix*. Moreover, the diagonal entries of Σ, called *singular values*, are always real and non-negative; they are usually arranged in decreasing order.

For example, the SVD of the 5×2 stoichiometry matrix S in Equation (6.35) can be written as:

$$\underbrace{\begin{bmatrix} -1 & 1 \\ -1 & \\ 1 & -1 \\ & -1 \\ & 1 \end{bmatrix}}_{S} = \underbrace{\begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\ u_{21} & u_{22} & u_{23} & u_{24} & u_{25} \\ u_{31} & u_{32} & u_{33} & u_{34} & u_{35} \\ u_{41} & u_{42} & u_{43} & u_{44} & u_{45} \\ u_{51} & u_{52} & u_{53} & u_{54} & u_{55} \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} \sigma_1 & \\ & \sigma_2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}}_{\Sigma} \underbrace{\begin{pmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{pmatrix}}_{V^T}. \tag{6.38}$$

Values of $\{u_{ij}\}$, $\{v_{ij}\}$ and $\{\sigma_i\}$ can be obtained using a numerical SVD routine such as MATLAB's `svd`; for example, for S above, the singular values are $\sigma_1 \simeq 2.3583$ and $\sigma_2 \simeq 1.1994$.

Applying Equation (6.38), Equation (6.35) becomes

$$\frac{d}{dt}\vec{x} = U\Sigma V^T\ \ \vec{r}(\vec{x}), \tag{6.39}$$

or (pre-multiplying by U^T and invoking unitarity of U)

$$\frac{d}{dt}U^T\vec{x} = \Sigma V^T\ \ \vec{r}(\vec{x}). \tag{6.40}$$

Denoting

$$\vec{y} = U^T\vec{x} \quad \text{(equivalently, } \vec{x} = U\vec{y}\text{)}, \tag{6.41}$$

Equation (6.40) can be written as:

$$\frac{d}{dt}\vec{y} = \Sigma V^T\ \ \vec{r}(U\vec{y}). \tag{6.42}$$

Since $\Sigma \in \mathbb{R}^{m\times n}$ with $m < n$, (i.e., Σ is "tall/thin" rectangular) and diagonal, we can write:

$$\Sigma = \begin{pmatrix} \Sigma_R \\ 0_{(n-m)\times m} \end{pmatrix}, \quad \vec{y} = \begin{pmatrix} \vec{y}_R \\ \vec{y}_c \end{pmatrix}, \quad U = \begin{pmatrix} U_{n\times m} & | & U_{n\times(n-m)} \end{pmatrix}, \tag{6.43}$$

where

$$U_{n\times m} = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{pmatrix}, \quad U_{n\times(n-m)} = \begin{pmatrix} u_{13} & u_{14} & u_{15} \\ u_{23} & u_{24} & u_{25} \\ u_{33} & u_{34} & u_{35} \\ u_{43} & u_{44} & u_{45} \\ u_{53} & u_{54} & u_{55} \end{pmatrix}, \tag{6.44}$$

and

$$\Sigma_R = \begin{bmatrix} \sigma_1 & \\ & \sigma_2 \end{bmatrix}, \quad \vec{y}_R = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \quad \vec{y}_c = \begin{pmatrix} y_3 \\ y_4 \\ y_5 \end{pmatrix}. \tag{6.45}$$

Substituting Equation (6.43) into Equation (6.42), we obtain

$$\frac{d}{dt}\vec{y}_R = \Sigma_R V^T \ \vec{r}(U\vec{y}), \tag{6.46}$$

$$\frac{d}{dt}\vec{y}_c = \vec{0}. \tag{6.47}$$

From Equation (6.47), $\vec{y}_c =$ constant; i.e., (using Equations (6.41) and (6.43))

$$\vec{y}_c = U^T_{n\times(n-m)}\vec{x} = \text{constant } = U^T_{n\times(n-m)}\vec{x}_0, \tag{6.48}$$

where $\vec{x}_0 = \vec{x}(0)$ are the initial concentrations of the reactants. *Equation (6.48) are the $n - m$ conservation equations of the reaction system.*

To use the conservation equations Equation (6.48) to eliminate unknowns and reduce the size of Equation (6.35), first note that (from Equations (6.41) and (6.43))

$$\vec{y}_R = U^T_{n\times m}\vec{x}. \tag{6.49}$$

Equation (6.46) can be re-expressed as:

$$\frac{d}{dt}\vec{y}_R = \vec{g}_R(\vec{y}_R) \triangleq \Sigma_R V^T \ \vec{r}\left(U \begin{bmatrix} \vec{y}_R \\ \vec{y}_c \end{bmatrix}\right), \tag{6.50}$$

with initial condition (using Equation (6.49)) $y_{R0} = U_{n\times m}^T \vec{x}_0$. $\vec{y}_c$ is a constant, given by Equation (6.48), representing conserved quantities.

Equation (6.50), an ODE of size m, is the reduced system. Solving it numerically yields $\vec{y}_R(t)$, from which $\vec{x}(t)$ can be recovered using Equations (6.43) and (6.41).

6.6 Enzyme-catalyzed Reactions: Michaelis-Menten Kinetics

A particular two-reaction chain, the *enzyme-catalyzed reaction*, is an important building block in biochemistry. Enzyme-catalyzed reactions convert a *substrate* A into a *product* D. The conversion process involves a *catalyst* B, which reacts with A to produce an intermediate *complex* C. The complex C changes into the product D via a one-way reaction, at the same time regenerating the catalyst B.

The enzyme-catalyzed reaction chain is:

$$\text{Reaction 1:} \quad A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C, \tag{6.51}$$

$$\text{Reaction 2:} \quad C \xrightarrow{k_3} D + B. \tag{6.52}$$

Note that Reaction 2 is a one-way reaction, i.e., the reverse reaction rate is zero.

The (nonlinear) rate equations of the above reactions can be set up in the same manner as for the two-reaction chain encountered previously in this chapter, with $n = 4$ and $m = 2$, and solved numerically. It is, however, possible to simplify the rate equations using judicious approximations, resulting in a smaller system of equations that relate the concentrations of the product D and the substrate A directly. This simplified system, known as the *Michaelis–Menten approximation*, is widely used in biochemical modelling.

The key approximation in deriving Michaelis–Menten kinetics is that the intermediate complex $[C]$ reaches *quasistatic steady state*, i.e., changes very slowly with time, for most of the reaction. Although this approximation is inaccurate at the beginning of the reaction, it tends to be roughly valid during most of the reaction. The rate of change

of $[C]$ is

$$\frac{d[C]}{dt} = k_1[A][B] - (k_2 + k_3)[C]. \tag{6.53}$$

Under the quasi-static approximation,

$$\frac{d[C]}{dt} \simeq 0 = k_1[A][B] - (k_2 + k_3)[C] \quad \Rightarrow \quad [C] \simeq \frac{k_1}{k_2 + k_3}[A][B]. \tag{6.54}$$

When the approximation (Equation (6.54)) is applied to the rates of change of $[A]$ and $[D]$,

$$\frac{d[A]}{dt} = -k_1[A][B] + k_2[C] \quad \text{and} \quad \frac{d[D]}{dt} = k_3[C], \tag{6.55}$$

they become equal in magnitude:

$$\frac{d[A]}{dt} \simeq -\frac{k_1 k_3}{k_2 + k_3}[A][B] \simeq -\frac{d[D]}{dt}. \tag{6.56}$$

The next step in deriving Michaelis–Menten kinetics is to apply conservation. Observe that each of the reactions Equations (6.51) and (6.52) conserves $[B] + [C]$ individually, hence $[B] + [C]$ is conserved overall. Since enzyme-catalyzed reactions are typically started with no intermediate complex, i.e., $[C]_0 = 0$, we have

$$[B] + [C] = [B]_0 \quad \Rightarrow \quad [B] = [B]_0 - [C], \tag{6.57}$$

where $[B]_0$ is the initial concentration of the catalyst.

Eliminating $[C]$ in Equation (6.57) by substituting Equation (6.54) leads to

$$[B] \simeq [B]_0 - \frac{k_1}{k_2 + k3}[A][B] \quad \Rightarrow \quad [B] \simeq \frac{k_2 + k_3}{k_2 + k_3 + k_1[A]}[B]_0. \tag{6.58}$$

Finally, substituting Equation (6.58) in Equation (6.56), we obtain

$$\frac{d[D]}{dt} \simeq -\frac{d[A]}{dt} \simeq \frac{k_1 k_3}{k_2 + k_3 + k_1[A]}[B]_0[A] = \frac{k_3}{\underbrace{\frac{k_2 + k_3}{k_1}}_{K_M} + [A]}[B]_0[A]. \tag{6.59}$$

Equation (6.59) is the Michaelis–Menten rate equation for enzyme-catalyzed reactions. It implies that Equations (6.51) and (6.52) may be viewed as a *single, one-way reaction*:

$$A \rightharpoonup D, \tag{6.60}$$

that converts A to D with forward reaction rate

$$R_f([A]) = \frac{k_3}{K_M + [A]}[B]_0[A], \quad K_M \triangleq \frac{k_2 + k_3}{k_1}. \tag{6.61}$$

Observe that this reaction rate is a nonlinear function of $[A]$, of the form $\frac{x}{1+x}$. For small concentrations of A (i.e., $[A] \ll K_M$), the rate is linear in $[A]$; but as $[A]$ becomes large (i.e., $A \gg K_M$), the rate *saturates* to the constant value $k_3[B]_0$. In other words, increasing the concentration of the substrate $[A]$, beyond a point, does not increase the rate of the reaction.

Note also that the reaction rate is directly proportional to the initial concentration of the catalyst, $[B]_0$; in other words, the catalyst B serves

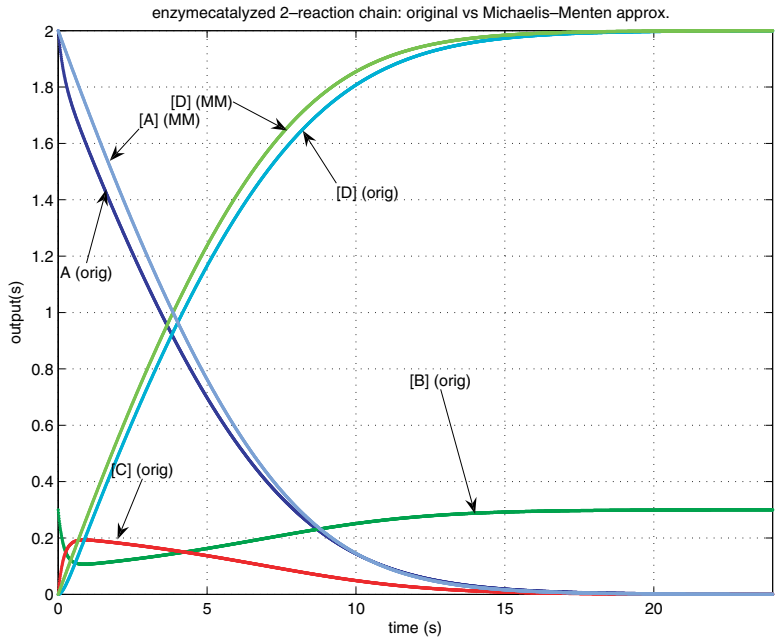


Fig. 6.5 Enzyme-catalyzed two-reaction chain: original system versus Michaelis–Menten approximation.

as a control that speeds up, or slows down, the rate at which A converts to D.

Figure 6.5 compares transient simulations[6] of the original reaction system Equation (6.51) and Equation (6.52) against that of the Michaelis-Menten approximation (Equation (6.59)). Observe that the concentrations of $[A]$ and $[D]$ from the Michaelis-Menten approximation match those from the original relatively well. Observe also how the Michaelis–Menten approximation is most inaccurate at the start of the simulation. This is consistent with the fact that the key assumption for Michaelis–Menten, namely the quasi-static assumption for $[C]$ in Equation (6.54), breaks down most at the start of the reaction, as is readily observed in the trace of $[C](t)$.

[6] MATLAB code for these simulations is available at http://www.eecs.berkeley.edu/∼jr/NOW-FT-Monograph/.

7

Sinusoidal Steady States of LTI Systems

We now concentrate on a special case of Equation (3.1): *linear time invariant* (LTI) DAEs, which arise in a variety of contexts. For example, circuits consisting only of linear resistors, capacitors and inductors, together with independent sources, are LTI; such circuits are important as models of electrical interconnect networks. Equally importantly, nonlinear DAEs can often be approximated usefully by LTI DAEs that are locally valid around a quiescent steady state ("operating point"). Such approximations, or *linearizations*, find many uses in simulation, analysis and design.

A computational advantage of specializing to LTI DAEs is that an especially efficient technique is available for finding their steady state responses to inputs that vary sinusoidally with time. This technique can also be generalized for nonlinear DAEs (see Section 9).

7.1 Linearizing DAEs Around a Quiescent Steady State

Recall that an input/output system $\vec{y}(t) = \mathcal{L}\{\vec{x}(t)\}$ is linear time-invariant (LTI) if the following conditions hold:

(1) (linearity) $\vec{y}_1(t) = \mathcal{L}\{\vec{x}_1(t)\}$ and $\vec{y}_2(t) = \mathcal{L}\{\vec{x}_2(t)\}$, then $\alpha\vec{y}_1(t) + \beta\vec{y}_2(t) = \mathcal{L}\{\alpha\vec{x}_1(t) + \beta\vec{x}_2(t)\}$, for all choices of

scalars α and β, and all choices of inputs $\vec{x}_1(t)$ and $\vec{x}_2(t)$; and

(2) (time invariance) if $\vec{y}(t) = \mathcal{L}\{\vec{x}(t)\}$, then $\vec{y}(t-\tau) = \mathcal{L}\{\vec{x}(t-\tau)\}$ for all choices of time-shift τ and input $\vec{x}(t)$.

The DAE Equation (3.1) can be interpreted as an input–output system with input $\vec{b}(t)$ and output $\vec{x}(t)$, i.e., $\vec{x}(t) = \mathcal{L}_{\rm DAE}\{\vec{b}(t)\}$. It is easy to verify that $\mathcal{L}_{\rm DAE}$ is time invariant. If the DAE has the special form:

$$C\frac{d}{dt}\vec{x}(t) + G\vec{x}(t) + \vec{b}(t) = \vec{0}, \tag{7.1}$$

where C and G are constant matrices, then it is easy to verify that it is linear as well; i.e., it is LTI.

Suppose we have a quiescent steady-state solution $\vec{x}^*$ of Equation (3.1), corresponding to a constant input $\vec{b}^*$. ($\vec{x}^*$ could be obtained, for example, using the Newton–Raphson method of Section 4.) Consider the question of the effect of a *small input perturbation* on the system. In other words, if the quiescent input $\vec{b}^*$ is changed by a small amount $\delta\vec{b}(t)$, how much does the operating point $\vec{x}^*$ change? Denote the amount it changes by $\delta\vec{x}(t)$. Then we have $\vec{b}(t) = \vec{b}^* + \delta\vec{b}(t)$, $\vec{x}(t) = \vec{x}^* + \delta\vec{x}(t)$ and Equation (3.1) can be rewritten as:

$$\frac{d}{dt}\vec{q}(\vec{x}^* + \delta\vec{x}(t)) + \vec{f}(\vec{x}(t) = \vec{x}^* + \delta\vec{x}(t)) + (\vec{b}^* + \delta\vec{b}(t)) = \vec{0}. \tag{7.2}$$

Assuming that the input perturbation $\delta\vec{b}(t)$ is small, we now make a crucial assumption: that the response $\delta\vec{x}(t)$ is also small. It seems natural that a small disturbance to a system should result in only a small change to its response. However, it is possible for (mathematical models of) systems to be at quiescent operating points that are "input–output unstable" — i.e., even tiny changes to the input can result in large changes to the output. A vertical pencil that is perfectly balanced on its tip is an example of such a situation; similarly, quiescent steady states of nonlinear oscillators are typically unstable, as are metastable states of flip-flops and latches. Nevertheless, most practical systems are in fact stable; our assumption of small $\delta\vec{x}(t)$, in response to small $\delta\vec{b}(t)$, is valid for such situations.

The utility of assuming $\delta\vec{x}(t)$ small is that it allows us to approximate the functions $\vec{q}(\cdot)$ and $\vec{f}(\cdot)$ using Taylor series, keeping only the linear terms:

$$\vec{f}(\vec{x}^* + \delta\vec{x}) \simeq \vec{f}(\vec{x}^*) + \underbrace{\left.\frac{d\vec{f}}{d\vec{x}}\right|_{\vec{x}^*}}_{G} \delta\vec{x}, \tag{7.3}$$

$$\vec{q}(\vec{x}^* + \delta\vec{x}) \simeq \vec{q}(\vec{x}^*) + \underbrace{\left.\frac{d\vec{q}}{d\vec{x}}\right|_{\vec{x}^*}}_{C} \delta\vec{x}. \tag{7.4}$$

Substituting Equations (7.3) and (7.4) in Equation (7.2), we obtain

$$\frac{d}{dt}\left[\vec{q}(\vec{x}^*) + C\delta\vec{x}(t)\right] + \vec{f}(\vec{x}^*) + G\delta\vec{x}(t) + \vec{b}^* + \delta\vec{b}(t) = \vec{0},$$

or

$$C\frac{d}{dt}\delta\vec{x}(t) + G\delta\vec{x}(t) + \delta\vec{b}(t) = \vec{0}, \tag{7.5}$$

which is an LTI DAE in the form (Equation (7.1)). This LTI DAE approximates the input–output relationship between $\delta\vec{b}(t)$ and $\delta\vec{x}(t)$ directly, under the assumption that both are small.

7.2 Computing Sinusoidal Responses in the Frequency Domain

We are interested in solving for periodic steady states — i.e., if the input is time-periodic, we are interested in finding a time-periodic output. While it is possible to solve Equation (7.5) using transient simulation techniques like those outlined in Section 5, it turns out that *sinusoidal periodic steady state solutions* of Equation (7.5) can be computed far more cheaply and elegantly. This technique, frequency-domain periodic steady-state analysis of LTI systems, is also known as *AC analysis*[1] in electronic circuit applications.

If the input of Equation (7.1) $\vec{b}(t)$ is sinusoidal (and, of course, real), we can use de Moivre's theorem to express it as:

$$\vec{b}(t) = \vec{b}_S\, e^{j\omega t} + \vec{b}_S^*\, e^{-j\omega t}, \tag{7.6}$$

[1] AC = Alternating Current, i.e., referring to sinusoidal waveforms.

where * denotes complex conjugation, $\vec{b}_S$ is a *constant* vector of complex numbers and $\omega = 2\pi f = 2\pi\frac{1}{T}$ (f being the frequency of the sinusoidal input, and T its period). The complex vector $\vec{b}_S$, a known quantity since the input is given, incorporates both amplitude and phase information of the sinusoidal components of $\vec{b}(t)$.

We now assume that the output $\vec{x}(t)$ is also sinusoidal, and with the same frequency f (equivalently, the same period T) as $\vec{b}(t)$. This assumption can be shown to hold asymptotically for stable LTI systems; i.e., if we "run" a stable LTI system for long enough.[2] Under this assumption, $\vec{x}(t)$ can be expressed in a form similar to Equation (7.6):

$$\vec{x}(t) = \vec{x}_S\, e^{j\omega t} + \vec{x}_S^*\, e^{-j\omega t}. \tag{7.7}$$

Finding the complex vector $\vec{x}_S$ is equivalent to finding a sinusoidal steady solution for $\vec{x}(t)$. Because $\vec{x}_S$ and $\vec{b}_S$ correspond to the fundamental terms of the Fourier series expansions of $\vec{x}(t)$ and $\vec{b}(t)$, respectively, they are often termed *frequency domain* quantities.

Substituting Equations (7.7) and (7.6) in Equation (7.1) leads to

$$\begin{aligned} C\frac{d}{dt}\left[\vec{x}_S\, e^{j\omega t} + \vec{x}_S^*\, e^{-j\omega t}\right] + G\left[\vec{x}_S\, e^{j\omega t} + \vec{x}_S^*\, e^{-j\omega t}\right] \\ + \vec{b}_S\, e^{j\omega t} + \vec{b}_S^*\, e^{-j\omega t} = \vec{0}, \end{aligned} \tag{7.8}$$

which simplifies to

$$\left[(j\omega C + G)\vec{x}_S + \vec{b}_S\right]e^{j\omega t} + \left[(j\omega C + G)\vec{x}_S + \vec{b}_S\right]^* e^{-j\omega t} = \vec{0}. \tag{7.9}$$

Since $e^{j\omega t}$ and $e^{-j\omega t}$ are linearly independent functions (over the field of complex numbers), their coefficients must individually equate to $\vec{0}$ for Equation (7.9) to hold; hence we obtain

$$(j\omega C + G)\vec{x}_S + \vec{b}_S = \vec{0}, \tag{7.10}$$

which is a *linear system of equations*, i.e., in the form $A\vec{x} = \vec{b}$, involving complex matrices and vectors.

Since G and C are often sparse in electronics (Section 3), biochemical reactions (Section 6) and other application domains, Equation (7.10) can be solved efficiently via a single sparse linear solution.

[2] The assumption typically is *not* valid for nonlinear systems (see Section 9) and for marginally stable, or unstable, LTI systems.

In contrast, applying transient timestepping methods (Section 5) to Equation (7.1) involves many $Ax = b$ solves; i.e., one at each timestep, for a potentially long series of time steps (so that a reasonable approximation to the asymptotic periodic steady state may be arrived at).

7.3 Frequency Sweeps of Transfer Functions

Since Equation (7.1) represents an LTI input–output system, it has a Laplace-domain transfer function $H(s)$ (e.g., [21]). Taking Laplace transforms of Equation (7.1) leads to

$$sC\vec{X}(s) + G\vec{X}(s) + \vec{B}(s) = \vec{0}, \tag{7.11}$$

or

$$\vec{X}(s) = \underbrace{-(sC + G)^{-1}}_{H(s)}\vec{B}(s). \tag{7.12}$$

Finding $H(j\omega)$, corresponding to the transfer function for sinusoidal inputs, is important in many applications (e.g., analog circuit design). Observe that computing Equation (7.10) constitutes an efficient means for finding $H(j\omega)$, given any value for $\omega = 2\pi f$. Frequently, f or ω is swept over a range of different values and $H(j\omega)$ computed for each frequency point. Figure 7.1 depicts the results of such a frequency sweep for a circuit consisting of two resistors and two capacitors.

7.4 Transfer Function Eigenanalysis of LTI DAEs

Consider Equation (7.1) with a scalar input $u(t)$, i.e., $\vec{b}(t) = \vec{b}_0 u(t)$; also choose a scalar output of the system, $y(t) \triangleq \vec{c}_o^T \cdot \vec{x}(t)$. The input–output relationship between $u(t)$ and $y(t)$ is given by

$$\begin{aligned} C\frac{d}{dt}\vec{x}(t) + G\vec{x}(t) + \vec{b}_0 u(t) &= \vec{0}, \\ y(t) &= \vec{c}_o^T \vec{x}(t). \end{aligned} \tag{7.13}$$

The Laplace-domain transfer function from input to output can be expressed as:

$$H(s) = \frac{Y(s)}{U(s)} = -\vec{c}_0^T (sC + G)^{-1}\vec{b}_0. \tag{7.14}$$

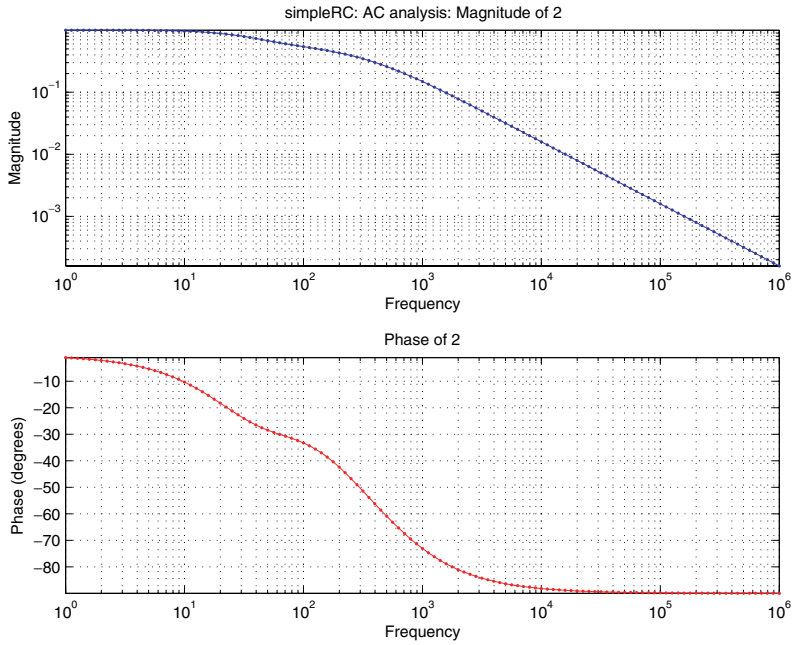


Fig. 7.1 Frequency sweep of a simple two-resistor, two-capacitor circuit.

If we assume G to be invertible[3] Equation (7.14) can be rewritten as:

$$H(s) = -\vec{c}_0^T \left(G(sG^{-1}C + I)\right)^{-1} \vec{b}_0 = -\vec{c}_0^T (s\underbrace{G^{-1}C}_{A} + I)^{-1} G^{-1}\vec{b}_0. \tag{7.15}$$

Eigendecompose

$$A = P\Lambda P^{-1}, \tag{7.16}$$

with Λ a diagonal matrix consisting of the eigenvalues $\lambda_1, \ldots, \lambda_n$. Using Equation (7.16), Equation (7.15) can be further expressed as

$$\begin{aligned} H(s) &= -\vec{c}_0^T (sP\Lambda P^{-1} + I)^{-1} G^{-1}\vec{b}_0 \\ &= -\vec{c}_0^T \left(P(s\Lambda + I)P^{-1}\right)^{-1} G^{-1}\vec{b}_0 \end{aligned}$$

[3] This is always the case if, e.g., Equation (7.13) has a well defined QSS (DC) solution that can be found by Newton–Raphson.

$$= \underbrace{-\vec{c}_0^T P}_{\vec{c}^T}(s\Lambda + I)^{-1} \underbrace{P^{-1}G^{-1}\vec{b}_0}_{\vec{r}}$$
$$= \vec{c}^T(s\Lambda + I)^{-1}\vec{r}. \tag{7.17}$$

Equation (7.17) can be written in expanded form as:

$$H(s) = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} \begin{bmatrix} s\lambda_1 + 1 & & & \\ & s\lambda_2 + 1 & & \\ & & \ddots & \\ & & & s\lambda_n + 1 \end{bmatrix}^{-1} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix}$$
$$= \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} \begin{bmatrix} \frac{1}{s\lambda_1+1} & & & \\ & \frac{1}{s\lambda_2+1} & & \\ & & \ddots & \\ & & & \frac{1}{s\lambda_n+1} \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix}$$
$$= \frac{c_1 r_1}{s\lambda_1 + 1} + \frac{c_2 r_2}{s\lambda_2 + 1} + \cdots + \frac{c_n r_n}{s\lambda_n + 1} = \sum_{i=1}^{n} \frac{c_i r_i}{s\lambda_i + 1}$$
$$= \frac{\frac{c_1 r_1}{\lambda_1}}{s + \frac{1}{\lambda_1}} + \frac{\frac{c_2 r_2}{\lambda_2}}{s + \frac{1}{\lambda_2}} + \cdots + \frac{\frac{c_n r_n}{\lambda_n}}{s + \frac{1}{\lambda_n}}$$
$$= \sum_{i=1}^{n} \frac{\overbrace{\frac{c_i r_i}{\lambda_i}}^{R_i}}{s + \underbrace{\frac{1}{\lambda_i}}_{p_i}} = \sum_{i=1}^{n} \frac{R_i}{s + p_i}. \tag{7.18}$$

Equation (7.18) shows that the Laplace domain transfer function has the simple form $H(s) = \sum_{i=1}^{n} \frac{R_i}{s+p_i}$, where R_i and p_i (known as *residues* and *poles*, respectively) are constant numbers, that can be calculated from G and C via the eigendecomposition procedure above.

The *impulse response* (or *kernel*) of the system, obtained by Laplace inversion of Equation (7.18), is

$$h(\tau) = \sum_{i=1}^{n} R_i e^{-p_i \tau}; \tag{7.19}$$

the time-domain input–output relationship of the system can then be expressed via the convolution relationship

$$y(t) = \int_{-\infty}^{\infty} h(t-\tau)u(\tau)\,d\tau = \sum_{i=1}^{n} R_i \int_{-\infty}^{\infty} e^{-p_i(t-\tau)}u(\tau)\,d\tau. \qquad (7.20)$$

Equation (7.18) establishes that the input–output relationship of the LTI DAE system (Equation (7.13)) can be expressed as a Laplace-domain transfer function with n terms, each involving a pole and a residue. Equivalently, its impulse response (Equation (7.19)) has n exponential terms, each with a time-constant $\lambda_i = \frac{1}{p_i}$ and of size R_i.

Observe that if $\Re(p_i) < 0$ for any i, the corresponding impulse response term blows up, implying that the system is *unstable*: the corresponding term in Equation (7.20) will blow up for most $u(t)$, even if it remains small and bounded for all time. Thus, the poles of the system, which can be computed for any DAE system (Equation (7.13)) given C, G, $\vec{b}_0$ and $\vec{c}_0$, provide direct information about the system's dynamical stability.

The eigendecomposition-based procedure above is computationally expensive, since it involves matrix inversion and eigendecomposition, both $O(n^3)$ operations. However, MOR-based techniques exist (see Section 10) that can estimate some of the dominant poles/residues with approximately $O(n)$ computation.

8

Direct and Adjoint Methods for Sensitivities and Noise

DAE models (Equation (3.1)) of physical systems usually involve many parameters, e.g., parameters of circuit devices (diode I_s and V_T, MOSFET V_t and β, and so on), reaction rates, catalyst concentrations, *etc.* Denoting parameters by the vector $\vec{p} \in \mathbb{R}^m$, Equation (3.1) can be rewritten as:

$$\frac{d}{dt}[\vec{q}(\vec{x}(t,\vec{p}),\vec{p})] + \vec{f}(\vec{x}(t,\vec{p}),\vec{p}) + \vec{b}(t,\vec{p}) = \vec{0}, \tag{8.1}$$

to emphasize the dependence of $\vec{q}(\cdot)$, $\vec{f}(\cdot)$ and $\vec{b}(\cdot)$ and $\vec{x}(t)$ on the parameters explicitly.

In this section, we look into techniques for efficient estimation of the effects of small variations of the parameters $\vec{p}$ on solutions of the DAE (Equation (8.1)). We will also examine the case when the input $b(t)$ is noisy, i.e., it is a *stochastic process*.

8.1 QSS (DC) Sensitivity Analysis

Given a nominal set of parameters $\vec{p}^*$, assume we have a QSS solution of Equation (8.1), $\vec{x}^*$; i.e.,

$$\vec{f}(\vec{x}^*,\vec{p}^*) + \vec{b}(\vec{p}^*) = \vec{0}. \tag{8.2}$$

If the parameters are changed away from nominal by a small amount $\Delta\vec{p}$, resulting in a change $\Delta\vec{x}$ to the QSS solution, Equation (8.1) can be written as:

$$\vec{f}(\vec{x}^* + \Delta\vec{x}, \vec{p}^* + \Delta\vec{p}) + \vec{b}(\vec{p}^* + \Delta\vec{p}) = \vec{0}. \tag{8.3}$$

If we now *assume* that Δx is small, we can expand $\vec{f}(\cdot)$ and $\vec{b}(\cdot)$ in Equation (8.3) in Taylor series, keeping only the linear terms, to obtain

$$\underbrace{\left.\frac{\partial\vec{f}}{\partial\vec{x}}\right|_{\vec{x}^*,\vec{p}^*}}_{G\in\mathbb{R}^{n\times n}}\Delta\vec{x} + \underbrace{\left(\left.\frac{\partial\vec{f}}{\partial\vec{p}}\right|_{\vec{x}^*,\vec{p}^*} + \left.\frac{\partial\vec{b}}{\partial\vec{p}}\right|_{\vec{p}^*}\right)}_{P\in\mathbb{R}^{n\times m}}\Delta\vec{p} \simeq \vec{0}. \tag{8.4}$$

Equation (8.4) can be rewritten as:

$$\lim_{\Delta\vec{p}\to\vec{0}}\frac{\Delta\vec{x}}{\Delta\vec{p}} = \underbrace{\left.\frac{\partial\vec{x}}{\partial\vec{p}}\right|_{\vec{x}^*,\vec{p}^*}}_{S\in\mathbb{R}^{n\times m}} = -G^{-1}P, \tag{8.5}$$

where the matrix S contains the *sensitivities* of the QSS solution $\vec{x}^*$ to changes in the nominal parameters $\vec{p}^*$.

8.1.1 Direct Sensitivity Computation

Computing the sensitivity matrix S directly using Equation (8.5) is computationally inefficient, since inverting G, an $n \times n$-matrix, takes $O(n^3)$ operations even if G is sparse; this is followed by $O(nm)$ operations for multiplying G^{-1} and P, assuming P sparse.

However, denoting by $\vec{s}_i$ and $\vec{p}_i$ the columns of S and P, respectively, Equation (8.5) can be expressed as:

$$G\vec{s}_i = -\vec{p}_i, \quad i = 1, \ldots, m. \tag{8.6}$$

Assuming G sparse, finding $\vec{s}_i$ involves a sparse matrix solution for each $i = 1, \ldots, m$, with total computational complexity $O(nm)$.

If the number of parameters m is about the same as, or larger than, the system size n, $O(nm) \sim O(n^2)$ or greater — i.e., sensitivity computation becomes quadratic in system size. Since large numbers of parameters are common in applications, this constitutes a significant computational limitation.

8.1.2 Adjoint Sensitivity Computation

It is possible to alleviate the quadratic computational complexity of direct sensitivity computation, even when there are many parameters, if we are not interested in the entire matrix S (which represents the sensitivities of every component of $\vec{x}$ to all parameters), but only in the sensitivities of a few outputs. Suppose we define an output y to be

$$y = \vec{c}^T \vec{x}, \tag{8.7}$$

where $\vec{c} \in \mathbb{R}^n$ is a vector of coefficients that combine the entries of $\vec{x}$ to produce a scalar. The sensitivity of y to the parameters is given by

$$\frac{\partial y}{\partial \vec{p}} = \vec{c}^T \frac{\partial \vec{x}}{\partial \vec{p}} = \vec{c}^T S = -\vec{c}^T G^{-1} P. \tag{8.8}$$

Taking the transpose (or *adjoint*) of Equation (8.8) results in

$$\left(\frac{\partial y}{\partial \vec{p}}\right)^T = -P^T \underbrace{G^{-T}\vec{c}}_{\vec{l}}. \tag{8.9}$$

Note that $\vec{l}$, a single column vector of size n, can be computed by solving

$$G^T \vec{l} = \vec{c}, \tag{8.10}$$

a matrix equation involving the *transpose*, or adjoint, of the QSS Jacobian matrix G. Since G is typically sparse, $\vec{l}$ can be found efficiently, i.e., with $O(n)$ computational cost.

Once $\vec{l}$ has been found, computing $P^T \vec{l}$ in Equation (8.9) requires another $O(m)$ operations if P is sparse. Hence the total computation involved in finding the sensitivity of a single output y to all parameters is $O(n + m)$. When the system size n and the number of parameters m are both large, this is substantially more efficient than direct sensitivity computation.

8.2 Stationary Noise Analysis

We now consider the situation where the DAE (Equation (3.1)) is perturbed, around a QSS solution $\vec{x}^*$, by a vector of small *random* input

perturbations $\vec{n}(t) \in \mathbb{R}^m$. Under the assumption that the impact on the solution is also small, the situation is the same as in Equation (7.2) or Equation (7.5), but with $\vec{b}(t) = M\vec{n}(t)$, $M \in \mathbb{R}^{n\times m}$. For notational convenience, we use the form (Equation (7.1)), i.e.,

$$C\frac{d}{dt}\vec{x}(t) + G\vec{x}(t) + M\vec{n}(t) = \vec{0}. \tag{8.11}$$

8.2.1 Stationarity, Autocorrelation, and Power Spectral Density

Because the input $\vec{n}(t)$ is a random or *stochastic* process, the solution $\vec{x}(t)$ will also be a stochastic process. $\vec{x}(t)$, being random, consists of a collection (*ensemble*) of sample waveforms. It is often more useful to find statistical characterizations of $\vec{x}(t)$, rather than some of its sample waveforms. Since Equation (8.11) represents the propagation of the input noise $\vec{n}(t)$ through an LTI system, the most relevant statistical quantity of interest is the *power spectral density* (PSD) of *stationary processes*.

A stochastic process $\vec{x}(t)$ is stationary[1] if its statistical properties do not depend on absolute time, but only on relative intervals. The less stringent concept of *wide-sense stationarity* (WSS) is typically most useful in applications. A WSS process is one in which the mean

$$\vec{m} = E\left[\vec{x}(t)\right] \tag{8.12}$$

is constant (independent of time), and the *autocorrelation function* (ACF)

$$R_{\vec{x}\vec{x}}(\tau) = E\left[(\vec{x}(t) - \vec{m})(\vec{x}(t+\tau) - \vec{m})^H\right] \tag{8.13}$$

is independent of t ($\cdot^H$ denotes the Hermitian or conjugate-transpose).

Without loss of generality, we assume a zero-mean process, i.e., $\vec{m} = \vec{0}$, hence Equation (8.13) becomes

$$R_{\vec{x}\vec{x}}(\tau) = E\left[\vec{x}(t)\vec{x}(t+\tau)^H\right]. \tag{8.14}$$

If $\vec{x} \in \mathbb{R}^n$, note that $R_{\vec{x}\vec{x}}$ is an $n \times n$-matrix containing the (stationary) correlation functions between $x_i(t)$ and $x_j(t)$, $\forall i, j = 1, \ldots, n$.

[1] The reader unfamiliar with stochastic processes may wish to consult, e.g., [30] or [79].

The diagonal entries of $\Sigma_{\vec{x}} = R_{\vec{x}\vec{x}}(0)$ are of particular interest since they are the variances $\sigma_{x_i}^2$ of the entries of $\vec{x}(t)$. These variances are useful because they represent the "noise power" of the stochastic waveforms $x_i(t)$.

The power spectral density of a stationary process is the Fourier transform of its ACF:

$$S_{\vec{x}\vec{x}}(f) = \mathcal{F}\{R_{\vec{x}\vec{x}}(\tau)\} = \int_{-\infty}^{\infty} R_{\vec{x}\vec{x}}(\tau)\, e^{-j2\pi f\tau}\, d\tau. \tag{8.15}$$

Equivalently, $R_{\vec{x}\vec{x}}$ can be expressed in terms of $S_{\vec{x}\vec{x}}$ using the inverse Fourier transform:

$$R_{\vec{x}\vec{x}}(\tau) = \mathcal{F}^{-1}\{S_{\vec{x}\vec{x}}(f)\} = \int_{-\infty}^{\infty} S_{\vec{x}\vec{x}}(f)\, e^{j2\pi f\tau}\, df. \tag{8.16}$$

Setting $\tau = 0$ in Equation (8.16) leads to

$$\Sigma_{\vec{x}} = R_{\vec{x}\vec{x}}(0) = \int_{-\infty}^{\infty} S_{\vec{x}\vec{x}}(f)\, df. \tag{8.17}$$

In other words, the integral of the PSD over all frequencies is the noise power.

8.2.2 Stationary Noise Propagation Through LTI Systems

If the input $\vec{n}(t)$ to the LTI system (Equation (8.11)) is stationary, and the system is stable, then it can be shown that the output of the system $\vec{x}(t)$ becomes stationary asymptotically. Finding the variance $\Sigma_{\vec{x}}$ of the stationary steady state of $\vec{x}(t)$ is useful in applications, since it provides a measure of noise power in the system.

One might speculate that if the variance of the input stochastic process, $\Sigma_{\vec{n}}$, is available, the variance $\Sigma_{\vec{x}}$ of the output can in principle be determined. It can be shown, however, that this is not true — knowledge of only the variance of the input is not sufficient to determine the variance of the output of the LTI system (Equation (8.11)). Instead of just the variance, knowledge of the *entire autocorrelation function* (or equivalently, the power spectral density) of the input noise is required.

The key to the propagation of noise statistics through an LTI system is a simple and elegant relationship [79] between the PSDs of the input

and output:

$$\underbrace{S_{\vec{x}\vec{x}}(f)}_{n\times n} = \underbrace{H(j2\pi f)}_{n\times m}\underbrace{S_{\vec{n}\vec{n}}(f)}_{m\times m}\underbrace{H^H(j2\pi f)}_{m\times n}. \tag{8.18}$$

$H(s)$ in Equation (8.18) is the *frequency-domain transfer function* of the LTI system, already encountered in Equation (7.12); for Equation (8.11),

$$H(s) = -(sC + G)^{-1}M. \tag{8.19}$$

Therefore, given the input PSD $S_{\vec{n}\vec{n}}(f)$, Equation (8.18) can be applied to find the output PSD $S_{\vec{x}\vec{x}}(f)$; following which the output variance can be found using Equation (8.17).

8.2.3 Uncorrelated Inputs

In general, the PSD $S_{\vec{n}\vec{n}}(f)$ of the input noise is a dense matrix function of the frequency f. In many applications, however, individual input noise source are *uncorrelated*, i.e.,

$$E[n_i(t_1)n_j(t_2)] \equiv 0 \quad \text{if } i \neq j. \tag{8.20}$$

If all inputs are uncorrelated, $S_{\vec{n}\vec{n}}(f)$ becomes *diagonal*, consisting of the PSDs of the individual noise sources $n_i(t)$.

8.2.4 White and Coloured Noise

The PSDs of many elementary noise sources in applications can be usefully idealized as constants, i.e., they are independent of the frequency f. Indeed, such *white noise* sources have far reaching practical and theoretical uses — even though they are not physically realizable or mathematically well defined, since (from Equation (8.17)) they have infinite variance. In circuits, for example, the *thermal noise* current across a linear resistor is white, with PSD[2]

$$S_T(f) = \frac{2kT}{R}, \tag{8.21}$$

[2] This is the two-sided PSD of thermal noise, consistent with Equation (8.17). Circuit designers typically use double this quantity, the one-sided PSD, changing the lower limit of the definite integral in Equation (8.17) from $-\infty$ to 0.

where k is Boltzmann's constant, T is the absolute temperature and R is the resistance. From the fundamental statistical physics of current transport, it can be shown that this PSD expression is an excellent approximation for $f \in [-10^{15}, 10^{15}]$ or so. Similarly, the *shot noise* current through a PN junction (diode) has PSD:

$$S_S(f) = qI_D, \tag{8.22}$$

where q is the electronic charge and I_D is the QSS (DC) current through the PN junction. Input noises in biochemical reaction systems, which stem from the underlying randomness of molecular dynamics, can also be shown to be white.

If a noise source is not white, it is termed *coloured*. An idealized type of coloured noise — also fundamentally important in many application domains — is *flicker noise*, which features a PSD that varies as $\frac{1}{f}$. Like the white noise idealization, flicker noise is also not well defined mathematically, since its integral over any frequency range that includes $f = 0$ is infinity. In reality, many flicker noise sources (such as flicker noise in MOS devices, engendered by the random filling and emptying of trap states in the gate oxide) have a lower cut-off frequency f_c, below which the PSD tends to a constant, instead of continuing to increase as $\frac{1}{f}$.

8.2.5 Direct Computation of Noise PSDs

The output noise PSDs $S_{\vec{x}\vec{x}}(f)$ can be computed by direct evaluation of Equation (8.18). This can be efficient if the number of noise sources m is small. For example, if $m = 1$, M and $H(j2\pi f)$ are simply column vectors; $H(j2\pi f)$ can be computed by solving the sparse linear system (using Equation (8.19))

$$(j2\pi fC + G)H(j2\pi f) = -M, \tag{8.23}$$

a step that involves $O(n)$ computation. Once $H(j2\pi f)$ is available, finding the diagonal elements of $S_{\vec{x}\vec{x}}(f)$, which involves simply squaring the magnitudes of each element of $H(j2\pi f)$ and multiplying by the scalar $S_{\vec{n}\vec{n}}(f)$, is also an $O(n)$ operation. This computation needs to be performed, of course, for each value of f; following which, the diagonal elements of $S_{\vec{x}\vec{x}}(f)$ can be integrated over f numerically (Equation (8.17))

to obtain the diagonal elements of $\Sigma_{\vec{x}}$, i.e., the variances of the output noise $\vec{x}(t)$.

As m increases, the computation and memory requirements of this procedure both increase as $O(mn)$. For small m, or if the PSDs of all the components of $\vec{x}(t)$ are desired, direct noise PSD computation is useful. In many applications, however, there are many input noise sources — in fact, the situation $m > n$ is quite common in practice. In such situations, the complexity of direct noise PSD computation is $O(mn) > O(n^2)$, which can be prohibitive when the system size n is large.

8.2.6 Adjoint Computation of Noise PSDs

As in the case of parameter sensitivities, it is possible to improve the computational efficiency of output PSD computation for large m, if we are interested only in a single output y (given by Equation (8.7)). From definition (Equations (8.14) and (8.15)), the PSD of $y(t)$ is

$$S_{yy}(f) = \vec{c}^T S_{\vec{x}\vec{x}}(f)\vec{c} = \underbrace{\vec{c}^T H(j2\pi f)}_{\vec{l}^H(f)}\, S_{\vec{n}\vec{n}}(f)\, \underbrace{H^H(j2\pi f)\vec{c}}_{\vec{l}(f)}, \tag{8.24}$$

where $\vec{l}(f)$, a column vector of size m, is given by (using Equation (8.19))

$$\vec{l}(f) = H^H(j2\pi f)\vec{c} = -M^H \underbrace{(j2\pi f C + G)^{-H}\vec{c}}_{\vec{r}(f)}. \tag{8.25}$$

Observe that $\vec{r}(f)$, a column vector of size n, can be computed efficiently by solving the sparse linear matrix equation:

$$(j2\pi f C + G)^H \vec{r}(f) = \vec{c}. \tag{8.26}$$

Computing $\vec{r}(f)$ via Equation (8.26) has $O(n)$ cost (for each frequency point f); once $\vec{r}(f)$ is available, $\vec{l}(f)$ can be obtained immediately via Equation (8.25), at cost $O(m)$ if M is sparse. Finally, if $S_{\vec{n}\vec{n}}(f)$ is also sparse, $S_{yy}(f)$ can be computed using Equation (8.24) at $O(m)$ cost.

Thus, the total computational cost for finding $S_{yy}(f)$ via this adjoint technique, at each frequency point, is $O(m + n)$. As in the case of

adjoint computation of parameter sensitivities, this can be substantially more efficient than direct noise computation if the system size n and the number of noise sources are large n. Moreover, if $S_{\vec{n}\vec{n}}(f)$ is diagonal (i.e., the input noise sources are uncorrelated) with entries $s_{ii}(f), \forall i = 1, \ldots, m$, Equation (8.24) can be expressed as:

$$S_{yy}(f) = \sum_{i=1}^{m} |l_i(f)|^2 s_{ii}(f). \tag{8.27}$$

In other words, the sensitivity of the PSD of the output y to that of the i^{th} noise source is $|l_i|^2$; i.e., $s_{ii}|l_i|^2$ can be used to rank-order noise sources in terms of their relative contribution to the output noise. Such information is valuable in applications, e.g., during the design of low-noise circuits.

9

Nonlinear Periodic Steady State and Multitime Analyses

In many applications, particularly within RF design, it is important to find the periodic steady state of a circuit driven by one or more periodic inputs. For example, a power amplifier driven to saturation by a large single-tone input is operating in a periodic steady state, or in a quasiperiodic steady state if the circuit is driven by more than one signal tone (e.g., an amplifier driven by two sinusoidal tones at 1 GHz and 990 MHz; such excitations being closer approximations to real-life signals than pure tones, particularly useful for estimating intermodulation distortion).

Transient analysis can of course be applied to find the (quasi) periodic steady state of a circuit, simply by performing a time-stepping integration of the circuit's differential equations long enough for the transients to subside and the circuit's response to become (quasi)periodic. This approach can have severe disadvantages, however. In typical RF circuits, the transients take thousands of periods to die out, hence the procedure can be very inefficient. Further, the strength of harmonic components is typically order of magnitude smaller than that of the fundamental, numerical errors from time-stepping transient integration can mask them completely. These issues are exacerbated in the presence of quasiperiodic excitations, because simulations need to be

much longer — e.g., for excitations of 1 GHz and 990 MHz, the system needs to be simulated for thousands of multiples of the common period, $\frac{1}{10\text{Mhz}}$, yet the simulation time steps must be much smaller than 1 ns, the period of the fastest tone. For these reasons, specialized techniques that are more efficient and accurate are needed. We will first outline two such methods, harmonic balance (HB) and shooting, that have complementary properties. We will then review a general mathematical framework, based on partial differential equations (PDEs) involving artificial time scales, that are useful for representing systems with two or more widely separated tones of excitation and response.

9.1 Methods for Nonlinear Periodic Steady States

9.1.1 Harmonic Balance

In the method of *Harmonic Balance (HB)* (e.g., [31, 47, 61, 62, 66, 67, 71, 93, 94]), $x(t)$ and $b(t)$ of Equation (3.1) are expanded in a Fourier series. The Fourier series can be one-tone for periodic excitations (e.g., 1 Ghz and its harmonics) or multi-tone in the case of quasiperiodic excitations (e.g., 1 GHz and 990 MHz, their harmonics, and intermodulation mixes). The DAE is rewritten directly in terms of the Fourier coefficients of $x(t)$ (which are unknown) and of $b(t)$; the resulting system of nonlinear equations is larger by a factor of the number of harmonic/mix components used, but are algebraic (i.e., there are no differential components). Hence they can be solved numerically using, e.g., the Newton–Raphson method [86].

Example 9.1. We illustrate the one-tone procedure with the following scalar DAE:

$$\dot{x} + x - \epsilon x^2 - \cos(2\pi 1000t) = 0 \tag{9.1}$$

First, we expand all time-variations in a Fourier series of (say) $M = 3$ terms, i.e., the DC component, fundamental and second harmonic components.

$$x(t) = \sum_{i=-2}^{2} X_i e^{j2\pi i 10^3 t}$$

Here, X_i are the unknown Fourier coefficients of $x(t)$. For notational convenience, we express them as the vector $X = [X_2, \ldots, X_{-2}]^T$.

Similarly, $x^2(t)$ is also expanded in a Fourier series in t; the Fourier coefficients of this expansion are functions of the elements of X, which we denote by $F_i(X)$.

$$x^2(t) = \sum_{i=-2}^{2} \sum_{k=-2}^{2} X_i X_k e^{j2\pi(i+k)10^3 t} = \sum_{i=-2}^{2} F_i(X) e^{j2\pi i 10^3 t} + \text{higher terms}$$

In this case, where the nonlinearity is a simple quadratic, F_i can be obtained analytically; but in general, numerical techniques like those used in harmonic balance need to be employed for computing these functions. For convenience, we write $F(X) = [F_2(X), \ldots, F_{-2}(X)]^T$.

We also write the Fourier coefficients of the excitation $\cos(2\pi 1000t)$ as a vector $B = \left[0, \frac{1}{2}, 0, \frac{1}{2}, 0\right]^T$. Finally, we write the differential term $\dot{x}$ also as a vector of Fourier coefficients. Because the differentiation operator is diagonal in the Fourier basis, this becomes simply ΩX, where $\Omega = j2\pi 1000 \text{ diag}(2, 1, 0, -1 - 2)$ is the diagonal frequency-domain differentiation matrix.

Invoking the orthogonality of the Fourier basis, we now obtain the HB equations for our DAE:

$$H(X) \equiv \Omega X + X - \epsilon F(X) - B = 0.$$

This is a set of nonlinear algebraic equations in five unknowns, and can be solved by numerical techniques such as Newton–Raphson.

The above example illustrates that the size of the HB equations is larger than that of the underlying DAE, by a factor of the number of harmonic/mix components used for the analysis. In fact, the HB equations are not only larger in size than the DAE, but also considerably more difficult to solve using standard numerical techniques. The reason for this is the dense structure of the derivative, or Jacobian matrix, of the HB equations. If the size of the DAE is n, and N harmonics/mix components are used for the HB analysis, the Jacobian matrix is of size $Nn \times Nn$. Simply the storage for the nonzero entries can become prohibitive for relatively moderate values of n and N; for example,

$n = 1000$ (for a medium-sized circuit) and $N = 100$ (e.g., for a two-tone problem with about 10 harmonics each) leads to 10 GB of storage for the matrix alone. Further, inverting the matrix, or solving linear systems with it, requires $O(N^3 n^3)$ operations, which is usually infeasible for moderate to large-sized problems. Such linear solutions are typically required as steps in solving the HB equations, for example by the Newton–Raphson method. Despite this disadvantage, HB is a useful tool for small circuits and few harmonics, especially for microwave circuit design. Moreover, as we will see later, new algorithms have been developed for HB that make it much faster for larger problems.

9.1.2 Shooting

Another technique for finding periodic solutions is the *Shooting* method (e.g., [3, 73, 107, 114]). Shooting works by finding an initial condition for the DAE that also satisfies the periodicity constraint. A guess is made for the initial condition, the system is simulated for one period of the excitation using time-stepping DAE solution methods, and the error from periodicity used to update the initial condition guess, often using a Newton–Raphson scheme.

More precisely, shooting computes the *state transition function* $\Phi(t, x_0)$ of Equation (3.1). $\Phi(t, x_0)$ represents the solution of the system at time t, given initial condition x_0 at time 0. Shooting finds an initial condition x^* that leads to the same state after one period T of the excitation $b(t)$; in other words, shooting solves the equation $H(x) \equiv \Phi(t, x) - x = 0$.

The shooting equation is typically solved numerically using the Newton–Raphson method, which requires evaluations of $H(x)$ and its derivative (or Jacobian) matrix. Evaluation of $H(x)$ is straightforward using time-stepping, i.e., transient simulation, of Equation (3.1). However, evaluating its Jacobian is more involved. The Jacobian matrix is of the same size n as the number of circuit equations, but it is dense, hence storage of its elements and linear solutions with it is prohibitive in cost for large problems. In this respect, shooting suffers from size limitations, similar to harmonic balance.

9.1.3 HB versus Shooting

In other respects, though, shooting has properties complementary to harmonic balance. The following list contrasts the main properties of HB and shooting.

- **Problem size:** The problem size is limited for both HB and shooting, due to the density of their Jacobian matrices. However, since the HB system is larger by a factor of the number of harmonics used, shooting can handle somewhat larger problems given the same resources. Roughly speaking, sizes of about 40 for HB and 400 for shooting represent practical limits.
- **Accuracy/Dynamic Range:** Because HB uses orthogonal Fourier bases to represent the waveform, it is capable of very high dynamic range — a good implementation can deliver 120 dB of overall numerical accuracy. Shooting, being based on time-stepping solution of the DAE with time-steps of different sizes, is considerably poorer in this regard.
- **Handling of nonlinearities:** HB is not well suited for problems that contain strongly nonlinear elements. The main reason for this is that strong nonlinearities (e.g., clipping elements) generate sharp waveforms that do not represent compactly in a Fourier series basis. Hence, many harmonics/mix components need to be considered for an accurate simulation, which raises the overall problem size.
  Shooting, on the other hand, is well suited for strong nonlinearities. By approaching the problem as a series of initial value problems for which it uses time-stepping DAE methods, shooting is able to handle the sharp waveform features caused by strong nonlinearities quite effectively.
- **Multitone problems:** A big attraction of HB is its ability to handle multitone or quasiperiodic problems as a straightforward extension of the one tone case, by using multitone Fourier bases to represent quasiperiodic signals. Shooting, on the other hand, is limited in this regard. Since it uses time-stepping DAE solution, shooting requires an excessive

number of timepoints when the waveforms involved have widely separated rates of change; hence it is not well suited for such problems.

9.1.4 "Fast" Versions of HB and Shooting

A disadvantage of both HB and shooting is their limitation to equation systems of relatively small size. This was not a serious problem as long as microwave/RF circuits contained only a few nonlinear devices. Since about the mid-1990s, however, economic and technological developments changed this situation. The market for cheap, portable wireless communication devices expanded greatly, leading to increased competition and consequent cost pressures. This spurred on-chip integration of RF communication circuits and the reduction of discrete (off-chip) components. On-chip design techniques favor the use of many integrated nonlinear transistors over even a few linear external components. Hence the need arose to apply HB and shooting to large circuits.

To address this issue, so-called *fast* algorithms arose that enable both HB and shooting to handle large circuits. The key property of these methods is that computation/memory usage grow approximately linearly with problem size. The enabling idea behind the improved speed is to express the dense Jacobian matrices as sums and products of simpler matrices that are either sparse, or have very regular structure so can be applied/inverted efficiently. Using these expansions for the Jacobian, special solution algorithms called *preconditioned iterative linear solvers* are applied to solve linear equations involving the Jacobian without forming it explicitly.

A detailed description of fast techniques is beyond the scope of this chapter; the interested reader is referred to [20, 61, 66, 94, 114] for further information. Here we outline the main ideas behind these methods in a simplified form using Example 9.1 for illustration, and summarize their main properties.

From Example 9.1, the Jacobian matrix of the HB system is

$$J = \frac{\partial H(X)}{\partial X} = \Omega + I - \epsilon \frac{\partial F(X)}{\partial X}.$$

Now, $F(X)$ in this case represents the vector of Fourier coefficients of the nonlinear term $f(x) = x^2$. One way in which these can be computed numerically is to (1) use the inverse Fast Fourier Transform (FFT) to convert the Fourier coefficients X to samples of the time-domain waveform $x(t)$, then (2) evaluate the nonlinear function $f(x) = x^2(t)$ at each of these samples in the time domain and finally (3) use the FFT to reconvert the time-domain samples of $f(x)$ back to the frequency domain, to obtain $F(X)$. The derivative of these operations can be expressed as:

$$\frac{\partial F(X)}{\partial X} = DGD^*,$$

where D is a block-diagonal matrix with each block equal to the Discrete Fourier Transform matrix, D^* is its inverse, and G is a diagonal matrix with entries $f'()$ evaluated at the time-domain samples of $x(t)$. Hence, the overall Jacobian matrix can be represented as:

$$J = \Omega + I - \epsilon DGD^*.$$

Observe that each of the matrices in this expansion is either sparse or consists of DFT matrices. Hence, multiplication of J with a vector is efficient, since the sparse matrices can be applied in approximately linear time, and the DFT matrix and its inverse can be applied in $N \log N$ time using the FFT, where N is the number of harmonics. It is this key property, that multiplications of J with a vector can be performed in almost-linear computation despite its dense structure, that enables the use of preconditioned iterative linear methods for this problem.

Preconditioned iterative linear methods (e.g., [25, 26, 102]) are a set of numerical techniques for solving linear systems of the form $Jc = d$. Modern iterative solvers like QMR [25] and GMRES [102] use Krylov-subspace techniques for superior performance. The key feature of these solvers is that the only way in which J is used is in matrix–vector products Jz. This contrasts with traditional methods for linear solution, which use Gaussian elimination or variants like LU factorizations directly on elements of J. Due to this property of iterative linear solvers, it is not necessary to even form J explicitly in order to solve linear systems with it, so long as a means is available for computing matrix–vector products with it.

As we have seen above, products with the HB Jacobian can be conveniently computed in almost-linear time without having to build the matrix explicitly. Hence preconditioned linear iterative techniques are well-suited to solving the linear systems that arise when solving the nonlinear HB equations using the Newton–Raphson method. If the iterative linear methods use only a few matrix–vector products with the Jacobian to compute the linear system's solution, and the Newton–Raphson is well-behaved, the overall cost of solving the HB equations remains almost-linear in problem size.

An important issue with preconditioned iterative linear solvers, especially those based on Krylov subspace methods, is that they require a good *preconditioner* to converge reliably in a few iterations. Convergence of the iterative linear method is accelerated by applying a preconditioner $\tilde{J}$, replacing the original system $Jc = d$ with the *preconditioned* system $\tilde{J}^{-1}Jc = \tilde{J}^{-1}d$, which has the same solution. For robust and efficient convergence, the preconditioner matrix $\tilde{J}$ should be, in some sense, a good approximation of J, and also "easy" to invert, usually with a direct method such as LU factorization. Finding good preconditioners that work well for a wide variety of circuits is a challenging task, especially when the nonlinearities become strong.

The ideas behind the fast techniques outlined above are applicable not just to HB but also to shooting [114]. Jacobian matrices from shooting can be decomposed as products and sums of the sparse circuit Jacobians matrices. Preconditioned linear iterative techniques can then applied to invert the Jacobian efficiently.

As an example of the application of the fast methods, consider the HB simulation of an RFIC quadrature modulator reported in [61]. The circuit, of about 9500 devices, was simulated by fast HB with a three-tone excitation, with a baseband signal at 80 kHz, and local oscillators at 178 MHz and 1.62 Ghz. The size of the circuit's DAE was $n = 4800$; the three tones, their harmonics and mixes totalled $N = 4320$ components. Simulating a circuit with these specifications is completely infeasible using tradition HB techniques. Using fast HB, however, the simulation required only 350 MB of memory, and took five days of computation on an SGI 150 MHz R4400 machine.

9.2 Multitime Partial Differential Equations (MPDEs)

HB is best suited for problems in which relatively few Fourier coefficients suffice to capture all the waveforms accurately. Conversely, if strong nonlinearities in the system lead to clipping and sharp spikes or edges in some waveforms, HB suffers from an explosion in the number of Fourier coefficients needed, and consequently, in the size of the system and the expense of solving it. Despite this shortcoming, the use of a two-tone (or multi-tone) Fourier basis enables HB to solve a widely separated time-scale problem directly in the desired asymptotic steady state form.

In contrast, shooting is able to handle the sharp waveform features caused by strong nonlinearities quite effectively, because it approaches the problem as a series of initial value problems for one cycle for which it uses time-stepping DAE methods. Unfortunately, for the same reason, it requires an excessive number of timepoints when the waveforms involved have widely separated rates of change; hence it is not really suitable for such problems. Nevertheless, for single-tone problems at least, shooting is able to deal with strong nonlinearities by solving a periodic boundary value in the time domain, a feature that eludes HB.

To help resolve this dichotomy, a formulation termed the **Multitime Partial Differential Equation** (MPDE) arose, resulting in an elegant way to combine the advantages of harmonic balance and shooting; i.e., to devise time-domain numerical methods that deal efficiently with widely separated rates of change, while retaining their ability to handle strong nonlinearities well.

9.2.1 Artificial Time Scales and the MPDE

Consider the waveform $y(t)$ shown in Figure 9.1, a simple two-tone quasiperiodic signal given by

$$y(t) = \sin\left(\frac{2\pi}{T_1}t\right)\sin\left(\frac{2\pi}{T_2}t\right), \qquad T_1 = 0.02\text{s}, \quad T_2 = 1\text{s}. \tag{9.2}$$

The two tones are at frequencies $f_1 = \frac{1}{T_1} = 50$ Hz and $f_2 = \frac{1}{T_2} = 1$ Hz, i.e., there are 50 fast-varying cycles of period $T_1 = 0.02$ s

modulated by a slowly-varying sinusoid of period $T_2 = 1$ s. Such waveforms (i.e., with two or more "components" varying at widely separated rates) arise in many practical situations, including oscillators, mixers, switched-capacitor filters, planetary systems, combustion engines, *etc.* As noted earlier, when DAEs describing such systems are solved by numerical integration (i.e., transient simulation), the time-steps taken need to be spaced closely enough that each rapid undulation of $y(t)$ is sampled accurately.

If each fast cycle is sampled at n points, the total number of time-steps needed for one period of the slow modulation is $n\frac{T_2}{T_1}$. To generate Figure 9.1, 15 points were used per cycle, hence the total number of samples was 750. This number can be much larger in applications where the rates are more widely separated, e.g., separation factors of 1000 or more are common in electronic circuits. It is worth noting that $y(t)$ in Equation (9.2) can be compactly represented with only two coefficients in a multitone Fourier series. But this property does not hold in general; consider, e.g., the product of a sine wave and a square wave, which would require a large number of Fourier components (as well as of time samples). Hence frequency-domain representations do not, in general, solve the problem of inefficient numerical representation of multi-rate signals.

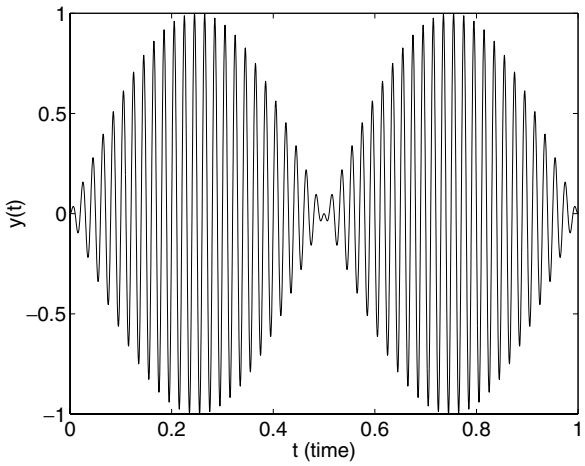


Fig. 9.1 Example two-tone quasi-periodic signal $y(t)$.

Now consider a multivariate representation of $y(t)$ using two artificial time scales, as follows: for the "fast-varying" parts of $y(t)$, t is replaced by a new variable t_1; for the "slowly-varying" parts, by t_2. The resulting function of two variables is denoted by

$$\hat{y}(t_1, t_2) = \sin\left(\frac{2\pi}{T_1} t_1\right) \sin\left(\frac{2\pi}{T_2} t_2\right). \tag{9.3}$$

The plot of $\hat{y}(t_1, t_2)$ on the rectangle $0 \le t_1 \le T_1,\ 0 \le t_2 \le T_2$ is shown in Figure 9.2. Observe that $\hat{y}(t_1, t_2)$ does not have many undulations, unlike $y(t)$ in Figure 9.1. *Hence it can be represented by relatively few points, which, moreover, do not depend on the relative values of* T_1 *and* T_2. Figure 9.2 was plotted with 225 samples on a uniform 15×15 grid — three times fewer than for Figure 9.1. This saving increases with increasing separation of the periods T_1 and T_2.

Further, note that $\hat{y}(t_1, t_2)$ is periodic with respect to both t_1 and t_2, i.e., $\hat{y}(t_1 + T_1, t_2 + T_2) = \hat{y}(t_1, t_2)$. This makes it easy to recover $y(t)$ from $\hat{y}(t_1, t_2)$, simply by setting $t_1 = t_2 = t$, and using the fact that $\hat{y}$ is bi-periodic. Given any value of t, the arguments to $\hat{y}$ are given by

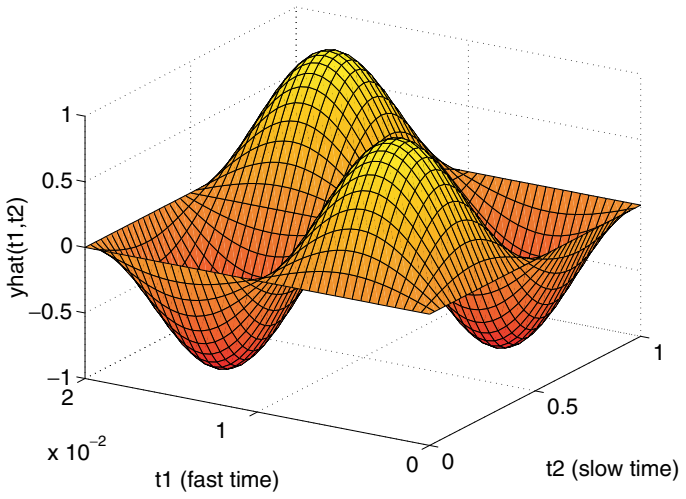


Fig. 9.2 Corresponding 2-periodic bivariate form $\hat{y}(t_1, t_2)$.

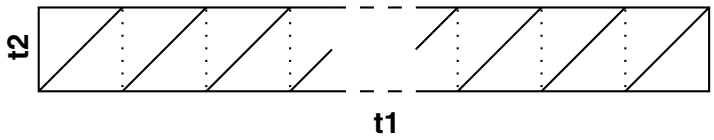


Fig. 9.3 Path in the t_1-t_2 plane.

$t_i = t \bmod T_i$. For example,

$$\begin{aligned} y(1.952\text{s}) &= \hat{y}(1.952\text{s}, 1.952\text{s}) \\ &= \hat{y}(97T_1 + 0.012\text{s}, T_2 + 0.952\text{s}) \\ &= \hat{y}(0.012\text{s}, 0.952\text{s}). \end{aligned}$$

It is easy, from direct inspection of the three-dimensional plot of $\hat{y}(t_1, t_2)$, to visualise what $y(t)$ looks like. As t increases from 0, the path given by $\{t_i = t \bmod T_i\}$ traces the sawtooth path shown in Figure 9.3. By noting how $\hat{y}$ changes as this path is traced in the $t_1 - t_2$ plane, $y(t)$ can be traced. Therefore, when the two time scales are widely separated, a cross-section along the fast time scale allows quick visualization of the fast-changing parts of $y(t)$, while a section along the slow time scale indicates the shape of the envelope riding on the fast variations.

From the above, it is clear that the multi-time form can be a very compact representation for multi-rate signals. To use this fact to advantage for solving dynamical systems, it is first necessary to reformulate a given DAE directly in terms of the multi-time forms of all its unknowns. Once this is done, the multi-time forms can be solved for directly, without involving the numerically inefficient single-time forms at any point.

Assuming that the DAE representation of our circuit/system is Equation (3.1), consider the following *Multitime Partial Differential Equation (MPDE)*:

$$\boxed{\frac{\partial \vec{q}(\hat{x})}{\partial t_1} + \frac{\partial \vec{q}(\hat{x})}{\partial t_2} + \vec{f}(\hat{x}) + \hat{b}(t_1, t_2) = \vec{0}.} \tag{9.4}$$

Equation (9.4) is a reformulation of the DAE (Equation (3.1)), in the following sense: if $\hat{b}(t_1, t_2)$ is chosen to satisfy $\vec{b}(t) = \hat{b}(t, t)$, and $\hat{x}(t_1, t_2)$ is a solution of Equation (9.4), then $\vec{x}(t)$ given by $\vec{x}(t) = \hat{x}(t, t)$ is a

solution[1] of Equation (3.1). In other words, any solution of the MPDE generates one for the underlying DAE. This fact is a simple consequence of the chain rule of differentiation, applied to Equation (9.4); see Section 9.2.3.

Example 9.2. Given the following scalar DAE:

$$\dot{x} = -\tanh(10x) + \sin(2\pi 1t)\tanh\left(2\sin(2\pi 10^4 t)\right),$$

the corresponding MPDE is

$$\left(\frac{\partial}{\partial t_1} + \frac{\partial}{\partial t_2}\right)\hat{x}(t_1,t_2) = -\tanh(10\hat{x}) + \sin(2\pi 1 t_1)\tanh\left(2\sin(2\pi 10^4 t_2)\right).$$

The excitation to the DAE is plotted in Figure 9.4 for a quarter cycle of the slow envelope; a blown-up segment is shown in Figure 9.5. This plot required 80,000 time samples. The two-time excitation to

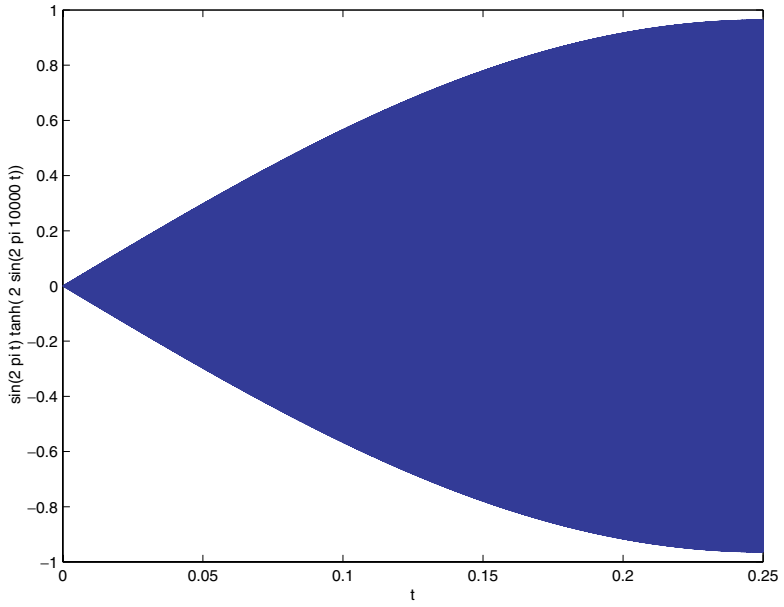


Fig. 9.4 Plot of $\sin(2\pi t)\tanh(2\sin(2\pi 10^4 t))$.

[1] A result in the opposite direction, that any quasiperiodic solution of Equation (3.1) generates a corresponding periodic solution of Equation (9.4), has also been established [72].

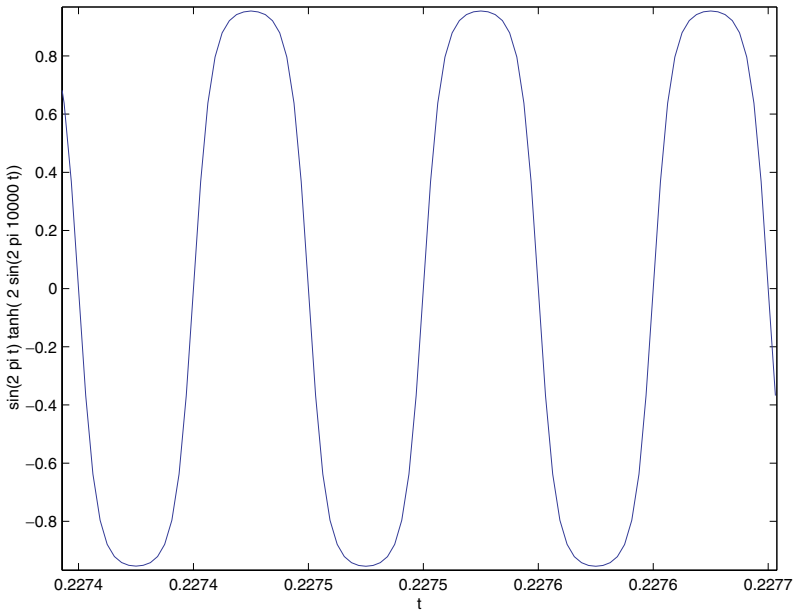


Fig. 9.5 Plot of $\sin(2\pi t)\tanh(2\sin(2\pi 10^4 t))$.

the MPDE is plotted (using 2,500 samples) in Figure 9.6. The global features of the waveform (i.e., the sinusoidal envelope and the flattened fast variations) are apparent from inspection of Figure 9.6, whereas closer inspection of the single-time plot is needed to observe the fast time scale detail.

Next, we consider how the MPDE can solved numerically for periodic and envelope-modulated solutions.

By solving the MPDE numerically in the time domain, strong nonlinearities can be handled efficiently. Several numerical methods are possible, including discretization of the MPDE on a grid in the $t_1 - t_2$ plane, or using a mixed time-frequency method in which the variation along one of the time scales is expressed in a short Fourier series. Quasiperiodic and envelope solutions can both be generated, by appropriate selection of boundary conditions for the MPDE. Sparse matrix and iterative linear methods are used to keep the numerical algorithms efficient even for large systems.

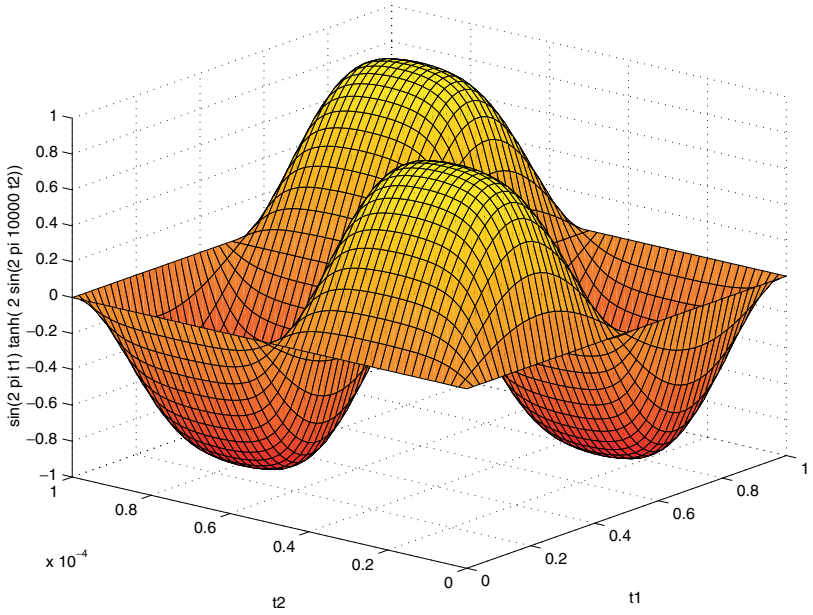


Fig. 9.6 Plot of $\sin(2\pi t_1)\tanh(2\sin(2\pi 10^4 t_2))$.

9.2.2 Boundary Conditions: Quasiperiodic and Envelope Solutions

Unlike for the DAE (Equation (3.1)), an initial condition at a single timepoint, or a two-point boundary condition specification, do not suffice to determine the solution of the MPDE (Equation (9.4)) uniquely. However, by specifying analogous boundary conditions, the MPDE can be used to find *quasiperiodic* or *envelope-modulated* solutions.

To find quasiperiodic solutions of the MPDE, it is necessary to simply set the *bi-periodic* boundary condition:

$$\hat{x}(t_1 + T_1, t_2 + T_2) = \hat{x}(t_1, t_2) \quad \forall t_1, t_2,$$

provided the excitation $\hat{b}(t_1, t_2)$ obeys the same condition. To obtain a numerical solution on the rectangle $0 \le t_1 \le T_1, 0 \le t_2 \le T_2$, it suffices to enforce this condition explicitly only on the boundaries. The single-time solution $x(t)$ derived by $x(t) = \hat{x}(t, t)$ is automatically quasiperiodic when the above condition is enforced.

Envelope-modulated solutions of the MPDE are also of interest in many applications. An envelope-modulated solution results from $\hat{x}(t_1, t_2)$ being periodic in only one of its arguments (say t_2), with the rate of variation with respect to the other argument t_1 typically being much slower than w.r.t t_2. This slower variation is useful for, e.g., capturing information signals that are not periodic.

Example 9.3. Figure 9.7 depicts the single-time form of the envelope modulated signal

$$b(t) = (1 - e^{-t})sin(2\pi 10t),$$

while Figure 9.8 depicts its two-time form, given by

$$\hat{b}(t_1, t_2) = (1 - e^{-t_1})sin(2\pi 10t_2).$$

Specifying a line of initial conditions along the $t_1 = 0$ boundary, together with periodicity constraints over the $t_2 = 0$ and $t_2 = T_2$ boundaries, suffices to generate envelope-modulated solutions of the MPDE.

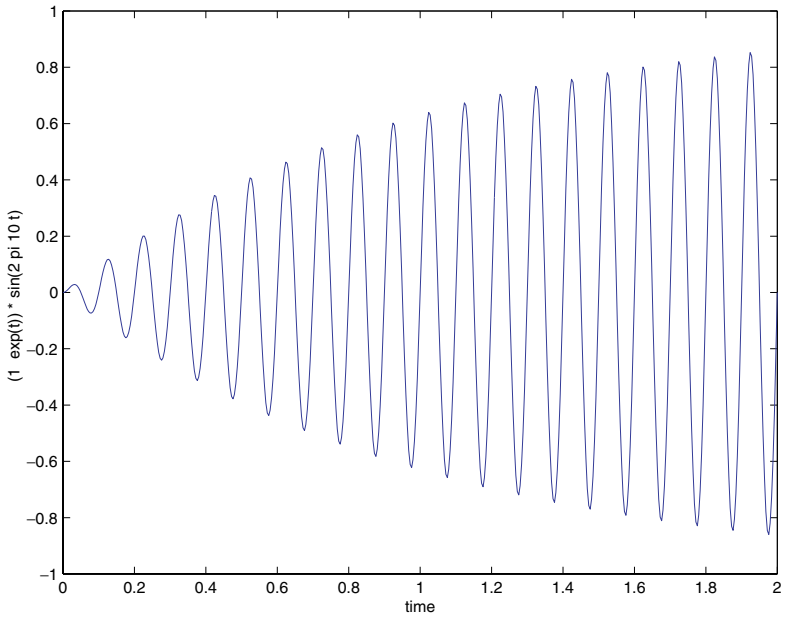


Fig. 9.7 Envelope $b(t)$.

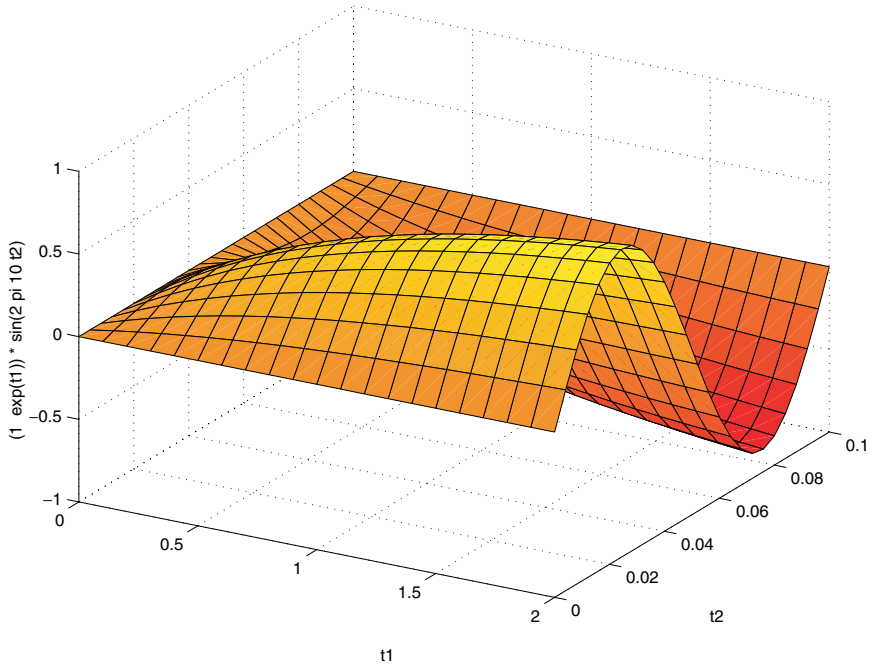


Fig. 9.8 Envelope $\hat{b}(t)$.

The conditions are

$$\hat{x}(t_1, t_2 + T_2) = \hat{x}(t_1, t_2) \quad \forall t_1, t_2,$$
$$\text{with } \hat{x}(0, t_2) = h(t_2), \text{ for some given } T_2\text{-periodic function } h.$$

9.2.3 Relationship Between the DAE and MPDE

The usefulness of the MPDE stems from strong relationships between its solutions and those of the underlying DAE. These relationships are summarized below for an MPDE with two time scales; similar relationships hold if there are more time scales.

Theorem 9.4. Let $\hat{b}$ be chosen to satisfy $\vec{b}(t) = \hat{b}(t, t)$. Then if $\hat{x}$ is a solution of the MPDE, $\vec{x}(t) = \hat{x}(t, t)$ is a quasiperiodic function satisfying the DAE.

Proof. Follows from substituting $\vec{x}(t) = \hat{x}(t,t)$ in Equation (3.1) and applying the chain rule of differentiation, together with Equation (9.4). □

Lemma 9.5. Let $\vec{b}(t)$ be periodic or quasiperiodic. Let $\hat{b}$ be chosen to satisfy $\vec{b}(t) = \hat{b}(t,t)$. Then if $\vec{x}(t)$ is a similarly periodic or quasiperiodic solution of the DAE, there exists (at least one) $\hat{x}$ such that $\vec{x}(t) = \hat{x}(t,t)$. In the case of strict quasiperiodicity, $\hat{x}$ is unique.

Lemma 9.6. If the DAE has a unique solution for any initial condition, then the MPDE has a unique solution given any envelope-type mixed initial/boundary condition. Furthermore, given any solution $\vec{x}(t)$ of the DAE, an infinite number of envelope-type solutions $\hat{x}$ of the MPDE can be found such that $\vec{x}(t) = \hat{x}(t,\ldots,t)$.

9.2.4 Numerical Methods for the MPDE

In general, MPDEs resulting from even simple problems are too complex to be solved analytically, hence numerical approaches are essential. Several choices are available for solving the MPDE numerically for quasiperiodic or envelope solutions. The nature of the underlying physical problem described by the MPDE dictates whether to use time-domain or frequency-domain expansions for some or all the time-scales of the multivariate unknowns.

Numerical analysis of the MPDE proceeds in two steps: (1) reformulation of the MPDE into a "discretized" (finite-dimensional) system of algebraic nonlinear equations, or, in the case of envelope type solutions, into a DAE in a single, envelope time scale; and (2) application of existing numerical techniques for nonlinear equation or DAE solution to the discretized equations.

The MPDE is discretized by expanding $\hat{x}(t_1,t_2)$ in a basis, along each of the time scales t_i. Different basis expansions may be used for different time scales. So far, collocation (in the time domain) and Fourier series expansions (in the frequency domain) have been used, though it

is possible to use other bases (e.g., wavelets). In general, if the variation of $\hat{x}$ with respect to t_i is expected to be very non-sinusoidal, a time-domain collocation procedure is preferred, whereas Fourier bases are used when relatively sinusoidal signals are expected. If $\hat{x}$ is periodic with respect to (say) t_1 (with period T_1), and if a time-domain expansion is being used, an additional equation setting $\hat{x}(0,t_2) = \hat{x}(T_1,t_2)$ is put in, or equivalently, the unknowns $\hat{x}(T_1,t_2)$ are simply replaced by $\hat{x}(0,t_2)$ in the formulation, thereby removing one point on the discretized time axis. If a Fourier series expansion is used, no explicit periodicity condition is required since the basis is, by its nature, valid only for periodic functions.

Example 9.7. Consider the scalar ODE

$$\dot{x} = -\lambda x - \epsilon x^3 + b(t)\sin(2\pi 10^3 t),$$

where $b(t)$ is much slower than the 1 kHz sinusoidal excitation. The corresponding MPDE is

$$\left(\frac{\partial}{\partial t_1} + \frac{\partial}{\partial t_2}\right)\hat{x} = -\lambda\hat{x} - \epsilon\hat{x}^3 + b(t_1)\sin(2\pi 10^3 t_2).$$

Suppose $b(t)$ is periodic, $= cos(2\pi t)$. We will expand the t_1 axis on a time-domain basis, and t_2 on a Fourier basis. The MPDE is

$$\left(\frac{\partial}{\partial t_1} + \frac{\partial}{\partial t_2}\right)\hat{x} = -\lambda\hat{x} - \epsilon\hat{x}^3 + \cos(2\pi t_1)\sin(2\pi 10^3 t_2).$$

We proceed in two steps. First, we expand all t_2 variations in a Fourier series of (say) $M = 3$ terms:

$$\hat{x}(t_1,t_2) = \sum_{i=-3}^{3} \hat{X}_i(t_1) e^{j2\pi i 10^3 t_2}.$$

Here, $\hat{X}_i(t_1)$ are the Fourier coefficients, functions of t_1. For notational convenience, we express the set as the vector $\hat{X} = [\hat{X}_3, \ldots, \hat{X}_{-3}]^T$.

Similarly, $\hat{x}^3(t_1,t_2)$ can also be expanded in a Fourier series in t_2; the Fourier coefficients of this expansion are functions of $\hat{X}$, which we denote by F_i. In this case, where the nonlinearity is a simple cubic, F_i can be obtained analytically; but in general, numerical techniques like

those used in harmonic balance need to be employed for computing these functions.

$$\hat{x}^3(t_1,t_2) = \sum_{i=-3}^{3} F_i(\hat{X}(t_1))e^{j2\pi i 10^3 t_2}.$$

For convenience, we write $F(\hat{X}(t_1)) = [F_3(\hat{X}(t_1)),\ldots,F_{-3}(\hat{X}(t_1))]^T$.

The MPDE can now be written in terms of the Fourier coefficients as:

$$\frac{\partial \hat{X}}{\partial t_1} + \overset{\text{FD}}{\Omega}\hat{X}(t_1) = -\lambda\hat{X}(t_1) - \epsilon F(\hat{X}(t_1)) + b(t_1)[0,0,-0.5j,0,0.5j,0,0]^T. \tag{9.5}$$

This is a vector differential equation of size $N = 2M+1 = 7$, the total number of Fourier coefficients. $\overset{\text{FD}}{\Omega} = j2\pi 10^3 \text{diag}(3,2,1,0,-1,-2,-3)$ is a diagonal matrix that results from differentiation of the Fourier coefficients.

We can now discretize Equation (9.5) further, using a time-domain collocation scheme along the t_1 axis. We collocate the range $t_1 = [0,1)$ (corresponding to the period of $b(t)$) at (say) $n = 10$ equally spaced points: $t_1 = 0, 0.1, \ldots, 0.9$. Note that due to periodicity, $\hat{X}(1.0) = \hat{X}(0.0)$, so the point $t_1 = 1$ need not be considered explicitly in our discretization.

$$\begin{aligned}
t_1 = 0: \quad & \left.\frac{\partial \hat{X}}{\partial t_1}\right|_0 + \overset{\text{FD}}{\Omega}\hat{X}(0) = -\lambda\hat{X}(0) - \epsilon F(\hat{X}(0)) \\
& \qquad + \cos(2\pi 0)[0,0,-0.5j,0,0.5j,0,0]. \\
t_1 = 0.1: \quad & \left.\frac{\partial \hat{X}}{\partial t_1}\right|_{0.1} + \overset{\text{FD}}{\Omega}\hat{X}(0.1) = -\lambda\hat{X}(0.1) - \epsilon F(\hat{X}(0.1)) \\
& \qquad + \cos(2\pi 0.1)[0,0,-0.5j,0,0.5j,0,0]. \\
& \vdots \\
t_1 = 0.9: \quad & \left.\frac{\partial \hat{X}}{\partial t_1}\right|_{0.9} + \overset{\text{FD}}{\Omega}\hat{X}(0.9) = -\lambda\hat{X}(0.9) - \epsilon F(\hat{X}(0.9)) \\
& \qquad + \cos(2\pi 0.9)[0,0,-0.5j,0,0.5j,0,0].
\end{aligned}$$

We now have $n = 10$ sets of equations, each of size $N = 7$. Finally, we express the differential terms using some numerical differentiation rule, say Backward Euler, to obtain the discretized MPDE:

$$t_1 = 0: \quad \frac{\hat{X}(0) - \hat{X}(0.9)}{0.1} + \overset{\text{FD}}{\hat{\Omega}}\hat{X}(0) = -\lambda\hat{X}(0) - \epsilon F(\hat{X}(0)) + \cos(2\pi 0)[0,0,-0.5j,0,0.5j,0,0]^T.$$

$$t_1 = 0.1: \quad \frac{\hat{X}(0.1) - \hat{X}(0)}{0.1} + \overset{\text{FD}}{\hat{\Omega}}\hat{X}(0.1) = -\lambda\hat{X}(0.1) - \epsilon F(\hat{X}(0.1)) + \cos(2\pi 0.1)[0,0,-0.5j,0,0.5j,0,0]^T.$$

$$\vdots$$

$$t_1 = 0.9: \quad \frac{\hat{X}(0.9) - \hat{X}(0.8)}{0.1} + \overset{\text{FD}}{\hat{\Omega}}\hat{X}(0.9) = -\lambda\hat{X}(0.9) - \epsilon F(\hat{X}(0.9)) + \cos(2\pi 0.9)[0,0,-0.5j,0,0.5j,0,0]^T.$$

Note the use of periodicity of $\hat{X}$ in the first equation, where $\frac{\partial \hat{X}}{\partial t_1}\Big|_0$ has been approximated by $\frac{\hat{X}(0)-\hat{X}(0.9)}{0.1}$.

The discretized equations, being a set of nonlinear algebraic equations in $nN = 70$ unknowns, can now be solved by numerical techniques, such as the Newton–Raphson method (Section 4).

In addition to an unlimited number of periodic time scales, it is also possible to have one nonperiodic time scale for capturing envelope behavior. To be of practical value, the envelope time scale should be the slowest-varying of all the time scales. Since a nonperiodic response is desired in this time scale, a Fourier series basis (which enforces periodicity) should not be used; instead, a nonperiodic time-domain basis should be applied. To formulate the MPDE for envelope simulation, it is convenient to discretize all the periodic time-scales initially, but not the envelope time scale itself. This results in an ordinary differential equation in the envelope time scale, which can be solved using standard numerical methods for DAEs.

Example 9.8. We change our above example by making $b(t)$ nonperiodic, e.g., $b(t) = 1 - e^{-t}$. To obtain the envelope differential equation

in t_1, we expand the t_2 axis in Fourier series, as before, resulting in Equation (9.5):

$$\frac{\partial \hat{X}}{\partial t_1} + \overset{\text{FD}}{\Omega}\hat{X}(t_1) = -\lambda \hat{X}(t_1) - \epsilon F(\hat{X}(t_1)) + b(t_1)[0,0,-0.5j,0,0.5j,0,0]^T.$$

This system, of size $N = 7$, is the envelope differential equation. This can be solved as an initial value problem, using any numerical method for integrating DAEs, to obtain the envelope solution $\hat{X}(t_1)$.

The discretized equations above have structural properties that mirror those from harmonic balance (for Fourier series expanded time scales) and shooting (for the axes expanded in time-domain bases). Specifically, they are large systems of equations with dense, though structured, Jacobian matrices. Applying straightforward numerical techniques to solve the linear Jacobian equation (a step required by, e.g., Newton–Raphson) can therefore become computationally infeasible as the size of the underlying DAE system, or the number of discretized basis functions along the multiple time axes, becomes large. Practical solution of medium and large-sized problems requires the use of more efficient linear solution techniques, such as the "fast" methods exploiting factored-matrix structure in the Jacobian already mentioned previously.

9.2.4.1 Example: Diode Rectifier with Pulse-Width Modulated Input

The diode rectifier circuit of Figure 3.1, powered by a large quasiperiodic two-tone power source $b(t)$, is simulated for a quasiperiodic solution using time-domain gridding. The power source was a train of fast pulses whose duty cycle was modulated at a much slower rate. Using $\mathsf{pulse}(\frac{t}{0.1\mu}, \text{duty})$ to denote each pulse (illustrated in Figure 9.9 for a duty cycle of 0.3), the excitation $b(t)$ was:

$$b(t) = \mathsf{pulse}\left(\frac{t}{T_2}, 0.2 + 0.3\sin(\frac{2\pi t}{T_1})\right), \quad T_1 = 1\text{ms}, \quad T_2 = 0.1\mu\text{s}$$

Two segments of $b(t)$, at widely separated times, are plotted in Figure 9.10, illustrating the variation in duty cycle. The duty-cycle

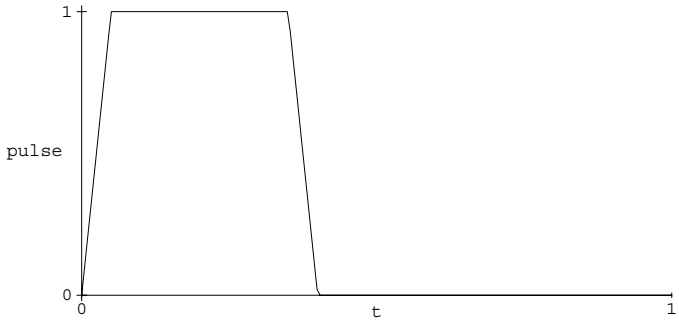


Fig. 9.9 pulse $(t, 0.3)$.

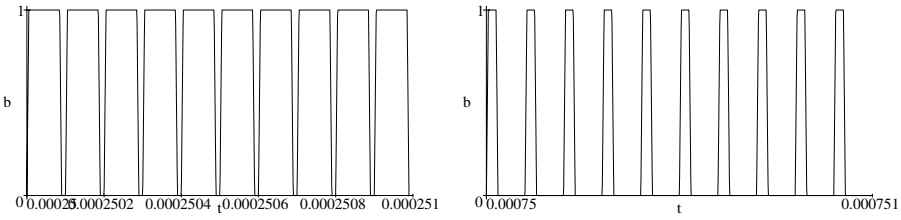


Fig. 9.10 $b(t)$: detail at widely separated times.

variation of $b(t)$ is more apparent in its bivariate form $\hat{b}(t_1, t_2)$:

$$\hat{b}(t_1, t_2) = \mathsf{pulse}\left(\frac{t_2}{T_2}, 0.2 + 0.3\sin(\frac{2\pi t_1}{T_1})\right), \qquad T_1 = 1\text{ms}, \quad T_2 = 0.1\mu\text{s}$$

$\hat{b}(t_1, t_2)$ is plotted in Figure 9.11. The duty cycle is the extent of the high region while moving along the t_2 direction, varying sinusoidally with respect to the slow variable t_1.

The bivariate form $\hat{x}$ of the output $x(t)$ is shown in Figure 9.12. The low-pass filter has smoothed out the fast variations in the t_2 direction. Since the rectified output depends on the duty cycle on the input, a slow-scale sinusoidal variation is observed as a function of t_1. The circuit was also simulated by univariate shooting for comparison.

The MPDE-based method was faster by over two orders of magnitude (40 s versus 1 h 20 m CPU time). Plots of the univariate solution $x(t)$ are shown in Figure 9.13. The waveform obtained by setting $t_1 = t_2 = t$ in the bivariate solution is denoted by the legend "new", and those from univariate shooting using 20 and 50 time-steps per fast

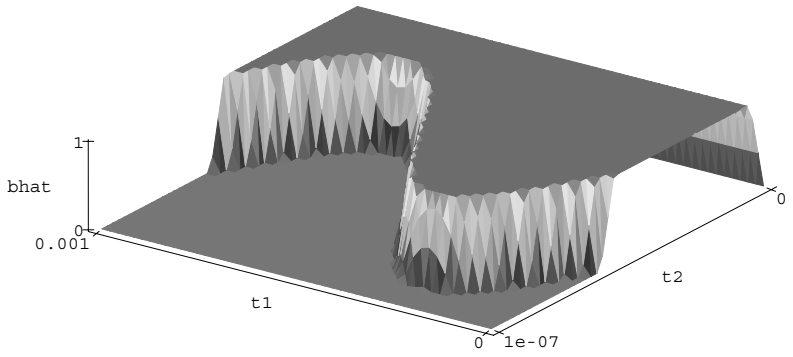


Fig. 9.11 The bivariate excitation $\hat{b}(t_1, t_2)$.

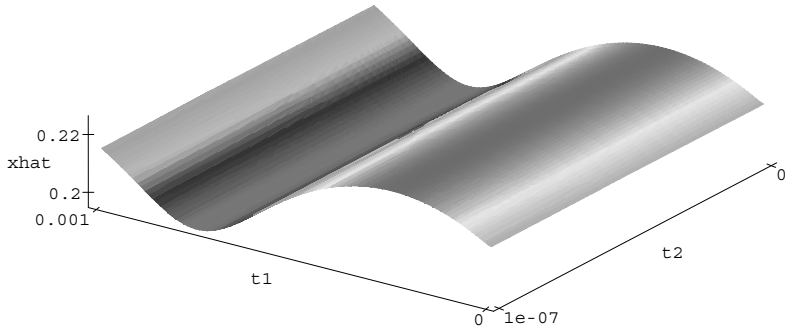


Fig. 9.12 The bivariate solution $\hat{x}(t_1, t_2)$.

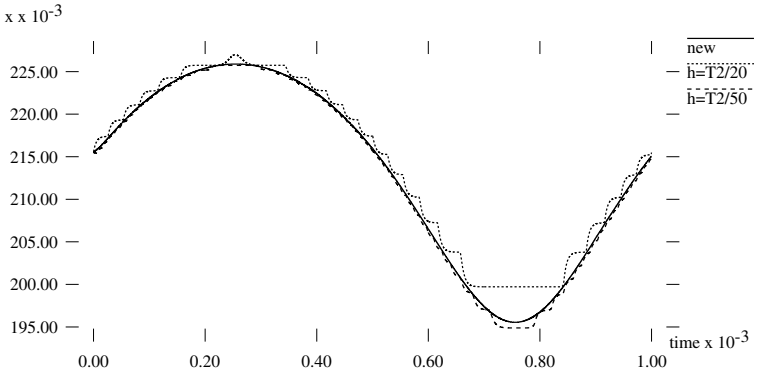


Fig. 9.13 Rectifier $x(t)$ slow-scale variation.

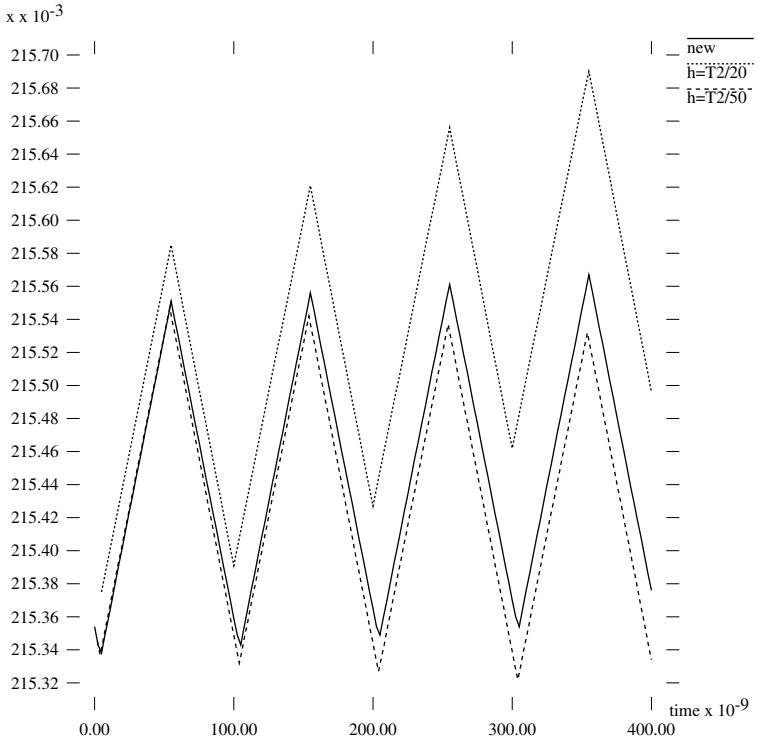


Fig. 9.14 Rectifier output fast-scale variation.

pulse by "h=T2/20" and "h=T2/50" respectively. Univariate shooting using 20 time-steps per pulse accumulated errors that grew to 15% near $t = 0.8$ ms, despite tight local error control. Increasing the number of time-steps to 50 per pulse reduced the error, but it remained significant at about 3%. The MPDE-based simulation produced the correct waveform.

The fast-scale detail of $x(t)$ near $t = 0$ is shown in Figure 9.14. Because of the relatively long time-constant of the smoothing R-C filter, the shape of the ripple is nearly triangular.

9.3 Multitime Analysis of Autonomous Systems

As described so far, the multi-time approach is based on the premise that multivariate forms are a numerically efficient way to represent

signals. Not all signals with widely separated time scales satisfy this premise. An important class is frequency-modulated (FM) signals, which can be generated in oscillatory DAEs that are externally forced. Such DAEs arise in, e.g., voltage-controlled oscillators (VCOs) and phase-locked loops (PLLs), circuits important in communications.

We illustrate the difficulty with an example. Consider the following prototypical FM signal

$$x(t) = \cos\left(2\pi f_0 t + k\cos(2\pi f_2 t)\right), \quad f_0 \gg f_2, \tag{9.6}$$

with instantaneous frequency

$$f(t) = f_0 - k f_2 \sin(2\pi f_2 t). \tag{9.7}$$

$x(t)$ is plotted in Figure 9.15 for $f_0 = 1$ MHz, $f_2 = 20$ KHz, and modulation index $k = 8\pi$. Following the same approach as for Equation (9.2), a bivariate form can be defined to be

$$\hat{x}_1(t_1, t_2) = \cos\left(2\pi f_0 t_1 + k\cos(2\pi f_2 t_2)\right), \quad \text{with } x(t) = \hat{x}_1(t,t). \tag{9.8}$$

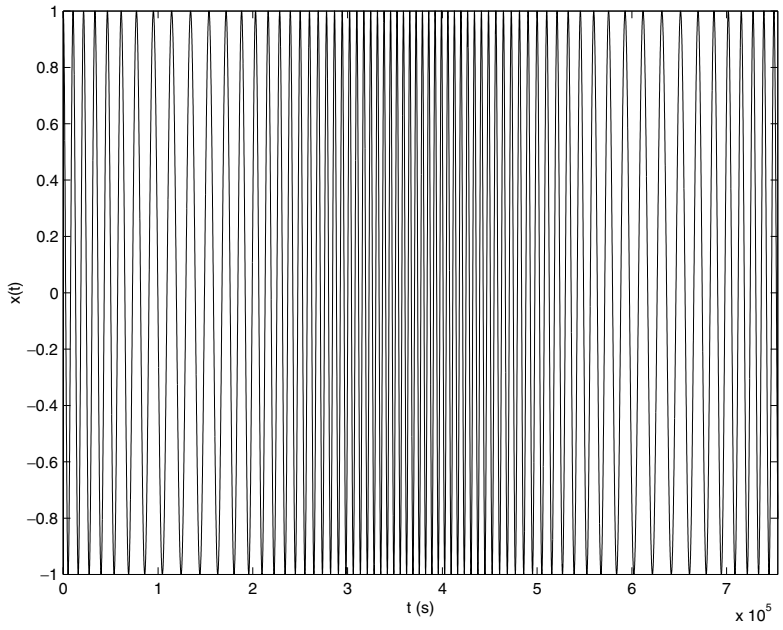


Fig. 9.15 FM signal.

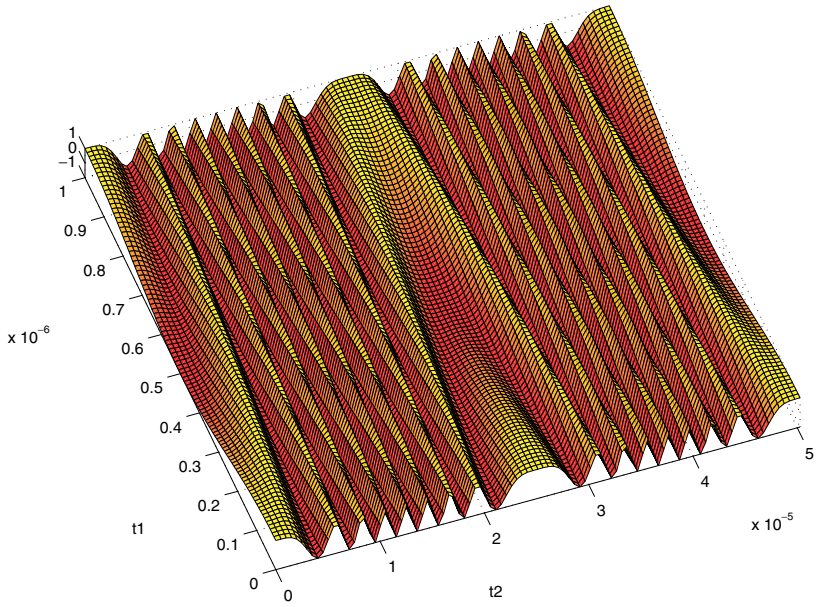


Fig. 9.16 $\hat{x}_1$: unwarped bivariate representation of FM signal.

Note that $\hat{x}_1$ is periodic in t_1 and t_2, hence $x(t)$ is quasiperiodic with frequencies f_0 and f_2. Unfortunately, $\hat{x}_1(t_1, t_2)$, illustrated in Figure 9.16, is not a simple surface with only a few undulations like Figure 9.2. When $k \gg 2\pi$, i.e., $k \approx 2\pi m$ for some large integer m, then $\hat{x}_1(t_1, t_2)$ will undergo about m oscillations as a function of t_2 over one period T_2. In practice, k is often of the order of $\frac{f_0}{f_2} \gg 2\pi$, hence this number of undulations can be very large. Therefore it becomes difficult to represent $\hat{x}_1$ efficiently by sampling on a two-dimensional grid. It is also clear, from Figure 9.16, that representing Equation (9.6) in the frequency domain will require a large number of Fourier coefficients to capture the undulations.

A seemingly promising approach towards resolving this representation problem is based on the intuition that FM is a slow change in the instantaneous frequency of a fast-varying signal. In the multivariate representation (Equation (9.3)), the high-frequency component is the inverse of T_1, the time-period along the t_1 (fast) time axis. It is natural to hope, therefore, that FM solutions can be captured by making this

time-period change along the slow time axis t_2, i.e., change T_1 to a periodic function $T_1(t_2)$, itself periodic with period T_2. Unfortunately, it can be shown [72] that FM-quasiperiodicity in a DAE *cannot* be captured by making T_1 a function of t_2.[2] It is easy to see qualitatively why this is the case. Although Figures 9.2 and 9.16 show the signal over only one period in each of the two time directions, the bivariate form is actually periodic over the entire $t_1 - t_2$ plane. However, making the time-period T_1 a function $T_1(t_2)$ turns the rectangular domain $[0,T_1] \times [0,T_2]$ (of Figures 9.2 and 9.16) into a non-rectangular domain of variable width. While it is possible to obtain a periodic function on the t_1-t_2 plane by placing rectangular boxes side by side to tile the entire plane, it is obvious that this cannot be done with boxes of variable width.

An alternative approach resolves this problem by preserving the rectangular shape of the domain boxes, and bending the path along which $y(t)$ is evaluated away from the diagonal line shown in Figure 9.3, so that its slope changes slowly. Since along the bent path $t_2 = t$, but t_1 is no longer equal to t, t_1 is referred to as a *warped* time-scale. This effectively results in stretching and squeezing the time axis differently at different times to even out the period of the fast undulations.

We illustrate this by returning to Equation (9.6). Consider the following new multivariate representation

$$\hat{x}_2(\tau_1, \tau_2) = \cos(2\pi\tau_1), \tag{9.9}$$

together with the warping function

$$\phi(\tau_2) = f_0\tau_2 + \frac{k}{2\pi}\cos(2\pi f_2\tau_2). \tag{9.10}$$

We now retrieve our one-dimensional FM signal (i.e., Equation (9.6)) as:

$$x(t) = \hat{x}_2(\phi(t), t). \tag{9.11}$$

Note that both $\hat{x}_2$ and ϕ, given in Equations (9.9) and (9.10), can be easily represented with relatively few samples, unlike $\hat{x}_1$ in Equation (9.8). $\hat{x}_2$ and ϕ are plotted in Figures 9.17 and 9.18. Note further

[2] Even so, Brachtendorf [5] has shown that this concept can be used to analyse transients in the special case of oscillators that eventually become T_1-periodic.

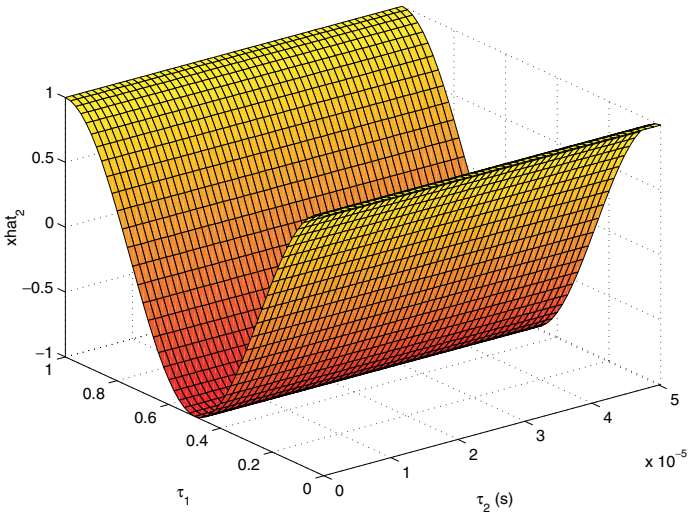


Fig. 9.17 $\hat{x}_2$: warped bivariate representation of FM signal.

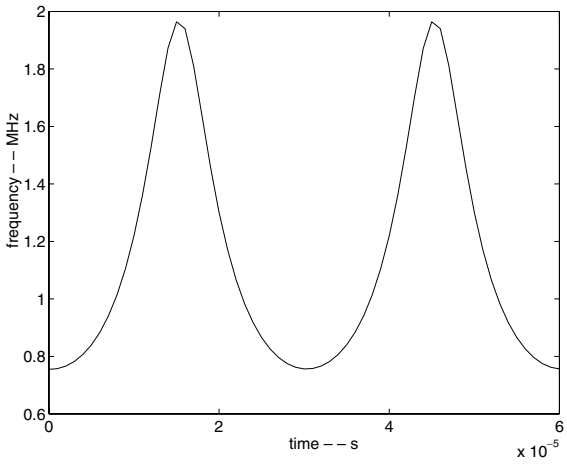


Fig. 9.18 $\phi(\tau_2)$: time warping function.

that $\phi(t)$ is the sum of a linearly increasing term and a periodic term, hence its derivative is periodic. This periodic derivative is equal to the instantaneous frequency, given in Equation (9.7), of $x(t)$. We will elaborate further on the significance of $\frac{\partial \phi}{\partial t}$ shortly.

It is apparent that there is no unique bivariate form and warping function satisfying Equation (9.11) — for example, two representations $\hat{x}_1$ and $\hat{x}_2$ have already been given (the warping function for $\hat{x}_1$ is $\phi(t) = t$). More generally, any warping function can be chosen, but at the possible cost of a resulting bivariate representation that cannot be sampled efficiently. To find an efficient bivariate representation, a crucial step in the warped-time approach is to avoid specifying the function $\phi(t)$ *a priori*, but to impose a smooth "phase condition" instead on the bivariate function, and use this to calculate ϕ. The phase condition can, for instance, require that the phase of the τ_1-variation of the function should vary only slowly (or not at all) as τ_2 is changed. Alternatively, a time-domain condition on the bivariate function (or a derivative) can be specified. As an illustration, consider the requirement that the τ_1-derivative along the line $\tau_1 = 0$ be a slowly-varying function (of τ_2):

$$\frac{\partial \hat{x}_3(0,\tau_2)}{\partial \tau_1} = -2\pi \sin(2\pi f_2 \tau_2), \tag{9.12}$$

together with

$$\hat{x}_3(\phi_3(t),t) = x(t) = \cos\left(2\pi f_0 t + k\cos(2\pi f_2 t)\right). \tag{9.13}$$

As is easily verified, these conditions lead to the following solutions for $\hat{x}_3$ and ϕ_3:

$$\begin{aligned} \hat{x}_3(\tau_1,\tau_2) &= \cos(2\pi\tau_1 + 2\pi f_2 \tau_2), \\ \phi_3(t) &= f_0 t + \frac{k}{2\pi}\cos(2\pi f_2 t) - f_2 t. \end{aligned} \tag{9.14}$$

Although $\hat{x}_3$ and ϕ_3 are not identical to $\hat{x}_2$ and ϕ in Equations (9.9) and (9.10), they retain the desired property of being easy to sample.

When $\hat{x}_2$ and ϕ in Equations (9.9) and (9.10) are chosen to be the warped bivariate representation of $x(t)$, the instantaneous frequency in Equation (9.7) is the derivative of $\phi(t)$, as already noted. The derivative of $\phi_3(t)$, on the other hand, differs from the instantaneous frequency by the constant $-f_2$. In general, all choices of $\phi(t)$ that result in compact representations will differ in their derivatives $\frac{\partial \phi}{\partial t}$ by amounts only of the order of the slow frequency f_2. When the fast frequency is much greater than the slow one, this difference is small compared to the instantaneous

frequency in Equation (9.7), therefore the term *local frequency* for $\frac{\partial \phi}{\partial t}$ is justified. The utility of the local frequency is that it is well-defined for *any* FM signal (possibly with non-sinusoidal waveforms and varying amplitudes), not just the ideal one of Equation (9.6), yet retains the essential intuition of FM. The ambiguity in $\frac{\partial \phi}{\partial t}$, of order f_2, is quite reasonable, since the intuitive concept of changing frequency is only meaningful to the same order. It should be kept in mind, of course, that concepts of varying frequency make intuitive sense only when the fast and slow time scales are widely separated.

The time warping concept can also be understood in a different, visual, manner. The difficulty in using $\hat{x}_1$ of Equation (9.8) is due to the fact that changing t_2 by even a small amount results in a large change in the phase of the outer cosine function, because k is large. Thus, the function is the same on all lines parallel to the t_1 axis, except for a phase that differs substantially even for lines that are nearby. The representation problem that this causes can be dealt with by sliding these lines up and down in the t_1 direction till there is no variation (or slow variation) in the phase from one line to another. This results in changing the rectangular domain box of Figure 9.2 to a non-rectangular one, but whose *width* is constant (i.e., with curved but parallel boundaries). In addition, the straight-line path $t_1 = t_2$ changes to a curved path because of the phase adjustment. The doubly periodic bivariate representation can be obtained by tiling the t_1-t_2 plane with the curved domain boxes (possible because the width is constant); in fact, after extending the function to the entire plane, it is possible to redefine the domain box to be a rectangle once again, resulting in Figure 9.17.

9.3.1 The Warped MPDE

These notions of warped time discussed above can be applied to the DAE in Equation (3.1), to obtain a modified version of the MPDE, dubbed the Warped MPDE or WaMPDE. This is a partial differential equation similar to the MPDE, but with a multiplicative factor of $\frac{\partial \phi}{\partial t}$ modifying one of the differential terms. By solving the WaMPDE together with the phase condition mentioned above, compact representations of the solutions of autonomous systems can be found by efficient

methods. The two-time WaMPDE corresponding to Equation (3.1) is:

$$\omega(\tau_2)\frac{\partial \vec{q}(\hat{x})}{\partial \tau_1} + \frac{\partial \vec{q}(\hat{x})}{\partial \tau_2} + \vec{f}(\hat{x}(\tau_1,\tau_2)) = \vec{b}(\tau_2). \tag{9.15}$$

The usefulness of Equation (9.15) lies in that specifying:

$$\vec{x}(t) = \hat{x}(\phi(t),t), \quad \phi(t) = \int_0^t \omega(\tau_2)\, d\tau_2 \tag{9.16}$$

results in $\vec{x}(t)$ being a solution to Equation (3.1).

The WaMPDE (Equation (9.15)) can be discretized and solved using the same numerical methods mentioned earlier for the MPDE. An important difference, though, is that an extra phase condition equation is required to fix the local frequency. For concreteness, we sketch the numerical procedure (using a Fourier basis) for solving the WaMPDE.

We first assume that $\hat{x}(\tau_1,\tau_2)$ is periodic in τ_1 with period 1:

$$\hat{x}(\tau_1,\tau_2) = \sum_{i=-\infty}^{\infty} \hat{X}_i(\tau_2) e^{ji\tau_1}. \tag{9.17}$$

Note that if $\hat{x}(\tau_1,\tau_2)$ satisfies Equation (9.15), then so does for $\hat{x}(\tau_1 + \Delta,\tau_2)$, for any $\Delta \in \mathbb{R}$ — this is simply because Equation (9.15) is autonomous in the τ_1 time scale. This ambiguity is removed in the same way as for unforced autonomous systems, i.e., by fixing the phase of (say) the k^{th} variable to some value,[3] e.g., 0. This is the phase constraint mentioned earlier.

We expand Equation (9.15) in one-dimensional Fourier series in τ_1, and also include the phase constraint, to obtain

$$\sum_{i=-\infty}^{\infty} \left(\frac{\partial \hat{Q}_i(\tau_2)}{\partial \tau_2} + ji\omega(\tau_2)\hat{Q}_i(\tau_2) + \hat{F}_i(\tau_2) \right) e^{ji\tau_1} = b(\tau_2), \tag{9.18}$$

$$\Im\{\hat{X}_l^k(\tau_2)\} = 0. \tag{9.19}$$

$\hat{Q}_i(\tau_2)$ and $\hat{F}_i(\tau_2)$ are the Fourier coefficients of $\vec{q}(\hat{x}(\tau_1,\tau_2))$ and $\vec{f}(\hat{x}(\tau_1,\tau_2))$, respectively. k and l are fixed integers; $\hat{X}_l^k(\tau_2)$ denotes the l^{th} Fourier coefficient of the k^{th} element of $\hat{x}$.

[3] Or some slow function of τ_2; the selection of a slowly varying phase condition is, in fact, the key to compact numerical representation of $\hat{x}(\cdot,\cdot)$.

Equations (9.18) and (9.19) together form a DAE system which can be solved for isolated solutions. In practice, the Fourier series (Equation (9.18)) can be truncated to $N_0 = 2M + 1$ terms with i restricted to $-M, \ldots, M$. In this case, Equations (9.18) and Equation (9.19) lead to $N_0 + 1$ equations in the same number of unknown functions of τ_2. As in the case of the MPDE, solving this DAE as an initial value problem leads to envelope solutions, while solving it as a periodic boundary value problem leads to quasiperiodic solutions.

9.3.2 Voltage-controlled Oscillator Example

The following example illustrates the application of the WaMPDE to a voltage-controlled oscillator (VCO). The oscillator consisted of an LC tank in parallel with a nonlinear resistor, whose resistance was negative in a region about zero and positive elsewhere. This led to a stable limit cycle. The capacitance was varied by adjusting the physical plate separation of a novel MEMS (**M**icro **E**lectro**M**echanical **S**ystem) varactor with a separate control voltage. The damping parameter of the mechanical structure was initially assumed small, corresponding to a near vacuum.

An envelope simulation was conducted using purely time-domain numerical techniques for both τ_1 and τ_2 axes. The initial control voltage of 1.5 V resulted in an initial frequency of about 0.75 MHz; Figure 9.19 shows the variation of the sinusoidal controlling voltage, with time-period 30 times that of the unforced oscillator. Figure 9.20 shows the resulting change in local frequency, which varies by a factor of almost 3.

Figure 9.21 depicts the bivariate waveform of the capacitor voltage (i.e., one entry of the vector $\hat{x}(\tau_1, \tau_2)$, with the warped τ_1 axis scaled to the oscillator's nominal time-period of 1 μs) . It is seen that the controlling voltage changes not only the local frequency, but also the amplitude and shape of the oscillator waveform.

The circuit was also simulated by traditional numerical ODE methods ("transient simulation"). The waveform from this simulation, together with the one-dimensional waveform obtained by applying Equation (9.16) to Figure 9.21, is shown in Figure 9.22. The match is so close that it is difficult to tell the two waveforms apart; however, the

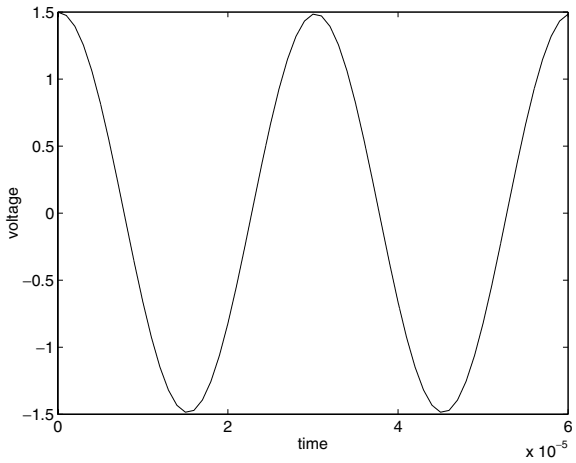


Fig. 9.19 VCO: controlling voltage.

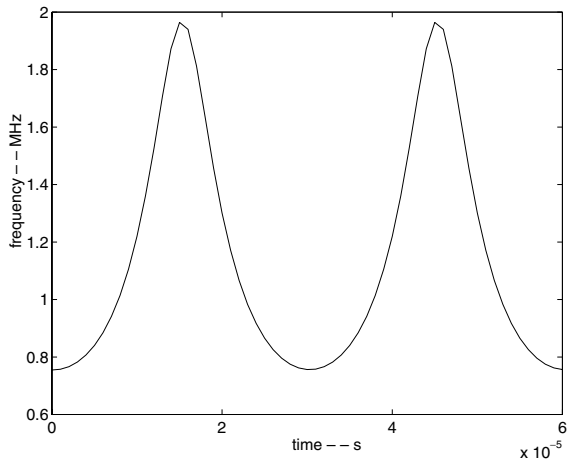


Fig. 9.20 VCO: frequency modulation.

thickening of the lines at about 60 μs indicates a deviation of the transient result from the WaMPDE solution. Frequency modulation can be observed in the varying density of the undulations.

The VCO was simulated again after two modifications: the damping of the MEMS varactor was increased to correspond to an air-filled cavity, and the controlling voltage was varied much more slowly, i.e., about

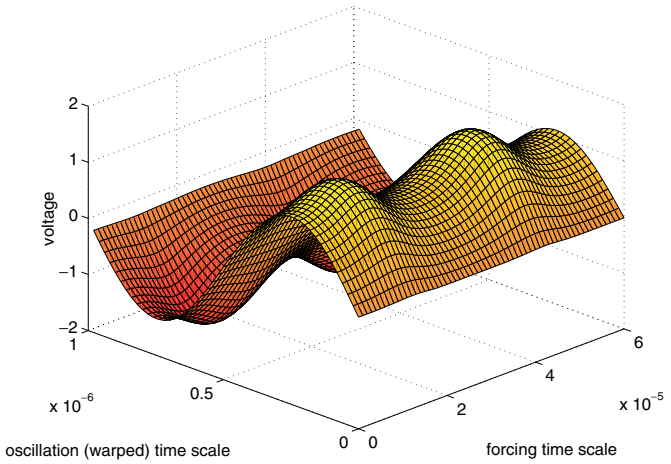


Fig. 9.21 VCO: bivariate representation of capacitor voltage.

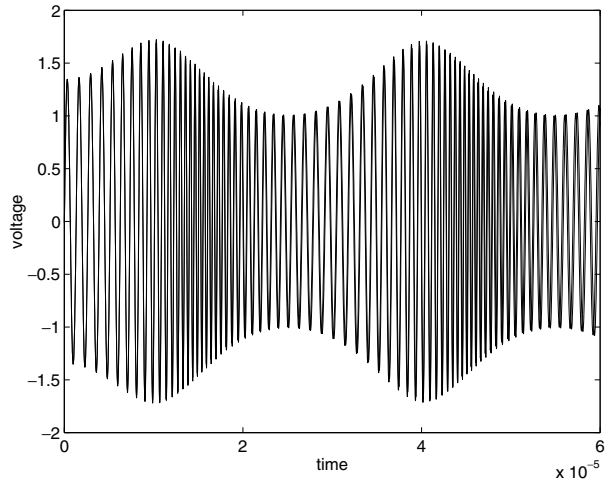


Fig. 9.22 VCO: WaMPDE versus transient simulation.

1,000 times slower than the nominal period of the oscillator. The controlling voltage was the same sinusoid shown in Figure 9.19, but with a period of 1 ms. Figure 9.23 shows the new variation in frequency; note the settling behavior and the smaller change in frequency, both due to the slow dynamics of the air-filled varactor.

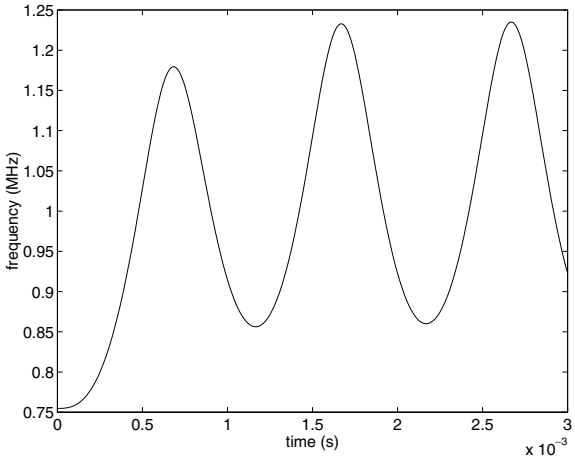


Fig. 9.23 Modified VCO: frequency modulation.

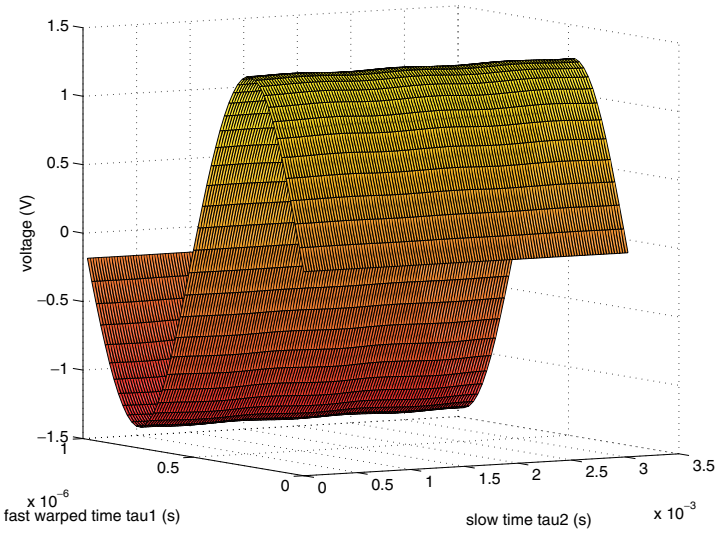


Fig. 9.24 Modified VCO: bivariate capacitor voltage.

Figure 9.24 depicts the new bivariate capacitor voltage waveform. Note that unlike Figure 9.21, the amplitude of the oscillation changes very little with the forcing. This was corroborated by transient simulation, the full results of which are not depicted due to the density of the fast oscillations. A small section of the one-dimensional waveform,

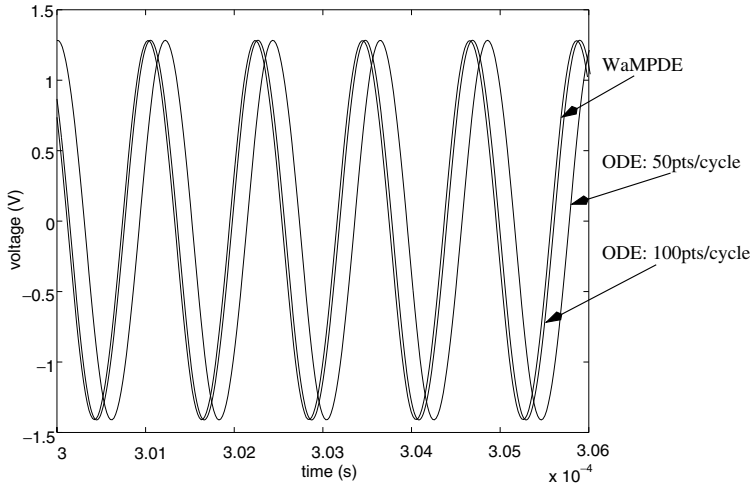


Fig. 9.25 Modified VCO: WaMPDE versus transient (a few cycles at 10% of the full run; phase errors from transient increase later).

consisting of a few cycles around 0.3 ms, is shown in Figure 9.25. The one-dimensional WaMPDE output of Equation (9.16) is compared against two runs of direct transient simulation, using 50 and 100 points per nominal oscillation period, respectively. It can be seen that even at an early stage of the simulation, direct transient simulation with 50 points per cycle builds up significant phase error. This is reduced considerably when 100 points are taken per cycle, but further along (not shown), the error accumulates again, reaching many multiples of 2π by the end of the simulation at 3 ms. In contrast, the WaMPDE achieves much tighter control on phase because the phase condition (a time-domain equivalent of Equation (9.19)) explicitly prevents build-up of error. To achieve accuracy comparable to the WaMPDE, transient simulation required 1,000 points per nominal cycle, with a resulting speed disadvantage of two orders of magnitude.

9.4 Generalized MPDE Formulations

The MPDE and WaMPDE of the previous sections are merely special cases of a considerably more general and abstract family of multitime partial differential equations. Although these generalized

MPDE formulations have already led to powerful theoretical results and computational methods [65], their uses and implications have yet to be fully explored.

Let $d > 0$ be an integer number of artificial time scales. Define a vector of d "generalized phase functions" to be

$$\hat{\Phi}(t_1, \ldots, t_d) = \begin{pmatrix} \tau_1 \\ \vdots \\ \tau_d \end{pmatrix} = \begin{pmatrix} \hat{\phi}_1(t_1, \ldots, t_d) \\ \vdots \\ \hat{\phi}_d(t_1, \ldots, t_d) \end{pmatrix}. \tag{9.20}$$

Similarly, define d "local frequency functions" to be

$$\hat{\Omega}(\tau_1, \ldots, \tau_d) = \begin{pmatrix} \hat{\omega}_1(\tau_1, \ldots, \tau_d) \\ \vdots \\ \hat{\omega}_d(\tau_1, \ldots, \tau_d) \end{pmatrix}. \tag{9.21}$$

Define the relationship between $\hat{\Phi}$ and $\hat{\Omega}$ to be

$$\left(\frac{\partial}{\partial t_1} + \cdots + \frac{\partial}{\partial t_d} \right) \hat{\Phi}(t_1, \ldots, t_d) = \hat{\Omega} \left(\hat{\Phi}(t_1, \ldots, t_d) \right). \tag{9.22}$$

Note that Equation (9.22) is a nonlinear PDE, similar in form to an ODE (as opposed to a DAE) in that the left-hand side differentiates the unknown vector $\hat{\Phi}$. This relation is an abstract generalization of the notion that phase is the integral of frequency.

Using the above, define the *generalized multitime PDE* (GeMPDE) to be:

$$\begin{aligned} \left[\hat{\Omega}^T(t_1, \ldots, t_d) \cdot \left[\tfrac{\partial}{\partial t_1}, \quad \cdots, \quad \tfrac{\partial}{\partial t_d} \right]^T \right] \vec{q}(\hat{x}(t_1, \ldots, t_d)) \\ + \vec{f}(\hat{x}(t_1, \ldots, t_d)) + \hat{b}(t_1, \ldots, t_d) = \vec{0}. \end{aligned} \tag{9.23}$$

Theorem 9.9. If $(\hat{\Omega}, \hat{x})$ solves the GeMPDE Equation (9.23), with $\hat{\Phi}$ and $\hat{\Omega}$ obeying the generalized phase-frequency relation (9.22), then $\vec{x}(t) \triangleq \hat{x}(\hat{\Phi}(t, \ldots, t))$ solves the DAE (Equation (3.1)) for $\vec{b}(t) \triangleq \hat{b}(\hat{\Phi}(t, \ldots, t))$.

Proof.

$$\begin{aligned}\frac{d}{dt}\vec{q}(x(t)) &= \frac{\partial \vec{q}(\hat{x}(\hat{\Phi}(t,\ldots,t)))}{\partial t} \\ &= \frac{\partial \vec{q}(\hat{x}(t_1=\phi_1,\ldots,t_d=\phi_d))}{\partial t_1}\frac{\partial \phi_1(t,\ldots,t)}{\partial t} \\ &\quad + \cdots + \frac{\partial \vec{q}(\hat{x}(t_1=\phi_1,\ldots,t_d=\phi_d))}{\partial t_d}\frac{\partial \phi_d(t,\ldots,t)}{\partial t}.\end{aligned}$$

Denoting $\vec{x}(\hat{\Phi}) \triangleq \vec{x}(\hat{\phi}_1,\ldots,\hat{\phi}_d)$ and $\hat{q}(\hat{\Phi}) \triangleq \vec{q}(\vec{x}(\hat{\Phi}))$, this can be written compactly using matrix–vector products as

$$\frac{d}{dt}\vec{q}(x(t)) = \left[\frac{\partial \hat{q}}{\partial t_1},\ldots,\frac{\partial \hat{q}}{\partial t_d}\right]\Big|_{\hat{\Phi}(t,\ldots,t)} \cdot \begin{bmatrix}\frac{\partial \phi_1(t,\ldots,t)}{\partial t}\\ \vdots \\ \frac{\partial \phi_d(t,\ldots,t)}{\partial t}\end{bmatrix}.$$

Since each component of the column vector above can be expanded using the chain rule as:

$$\frac{\partial \phi_k(t,\ldots,t)}{\partial t} = \frac{\partial \phi_k(\tau_1=t,\ldots,\tau_d=t)}{\partial \tau_1} + \cdots + \frac{\partial \phi_k(\tau_1=t,\ldots,\tau_d=t)}{\partial \tau_d},$$

we have

$$\begin{bmatrix}\frac{\partial \phi_1(t,\ldots,t)}{\partial t}\\ \vdots \\ \frac{\partial \phi_d(t,\ldots,t)}{\partial t}\end{bmatrix} = \left(\frac{\partial}{\partial \tau_1} + \cdots + \frac{\partial}{\partial \tau_d}\right)\hat{\Phi}(t,\ldots,t);$$

which, using Equation (9.22), becomes

$$\begin{bmatrix}\frac{\partial \phi_1(t,\ldots,t)}{\partial t}\\ \vdots \\ \frac{\partial \phi_d(t,\ldots,t)}{\partial t}\end{bmatrix} = \hat{\Omega}\left(\hat{\Phi}(t,\ldots,t)\right).$$

Hence we have

$$\frac{d}{dt}\vec{q}(x(t)) = \left[\frac{\partial \hat{q}}{\partial t_1},\ldots,\frac{\partial \hat{q}}{\partial t_d}\right]\Big|_{\hat{\Phi}(t,\ldots,t)} \hat{\Omega}\left(\hat{\Phi}(t,\ldots,t)\right),$$

which is simply the first term of Equation (9.23), evaluated at $(t_1, \ldots, t_d) = \hat{\Phi}(t, \ldots, t)$. Hence, applying Equation (9.23), we have proved that:

$$\frac{d}{dt}\vec{q}(x(t)) = -f(\hat{x}(\hat{\Phi}(t, \ldots, t)) + b(\hat{x}(\hat{\Phi}(t, \ldots, t)) = -f(x(t)) + b(t). \quad \square$$

10

Model Order Reduction of Linear, Nonlinear and Oscillatory Systems

On-chip and on-package integrated systems, such as systems-on-chip (SoCs) and systems-in-packages (SiPs), are typically composed of a complex mix of digital and mixed-signal circuit blocks. Verifying such systems prior to fabrication is challenging due to their size and complexity. Automated approaches towards extracting system-level macromodels from SPICE-level descriptions of circuit blocks is an increasingly important component of sustainable methodologies for system verification.

We present an overview of algorithmic methods for extracting linear and nonlinear macromodels of mixed-signal circuits. We then present applications that highlight the potential impact of such techniques.

10.1 Overview: LTI, LTV and Nonlinear MOR Techniques

A common and useful approach towards verification of systems containing mixed-signal, RF and analog blocks, both during early system design and after detailed block design, is to replace large and/or complex blocks by small *macromodels* that replicate their input–output functionality well, and then to verify the macromodeled system. The

macromodeled system can be simulated rapidly in order to evaluate different choices of design-space parameters. Such a macromodel-based design and verification process affords circuit and system designers considerable flexibility and convenience through the design process, especially if performed hierarchically using macromodels of differing sizes and fidelity.

A key issue in the above methodology is, of course, the creation of macromodels that represent the blocks of the system well. This is a challenging task for the wide variety of mixed-signal blocks in use today. The most prevalent approach towards creating macromodels is *manual abstraction*. Macromodels are usually created by the same person who designs the original block, often aided by simulations. While this is the only feasible approach today for many complex blocks, it does have a number of disadvantages compared to the *automated alternatives* that are the subject of this chapter. Simulation often does not provide abstracted parameters of interest directly (such as poles, residues, modulation factors, *etc.*); obtaining them by manual postprocessing of simulation results is inconvenient, computationally expensive and error-prone. Manual structural abstraction of a block can easily miss the very nonidealities or interactions that detailed verification is meant to discover. With device dimensions shrinking below 100 nm and non-idealities (such as substrate/interconnect coupling, degraded device characteristics, *etc.*) becoming increasingly critical, the fidelity of manually generated macromodels to the real subsystems to be fabricated eventually is becoming increasingly suspect. Adequate incorporation of non-idealities into behavioral models, if at all possible by hand, is typically complex and laborious. Generally speaking, manual macromodelling is heuristic, time-consuming and highly reliant on detailed internal knowledge of the block under consideration, which is often unavailable when IP blocks that are not designed in-house are utilized. As a result, the potential time-to-market improvement via macromodel-based verification can be substantially negated by the time and resources needed to first generate the macromodels.

It is in this context that there has been considerable interest in *automated techniques* for the creation of macromodels. Such techniques take a detailed description of a block — for example, a layout-extracted

SPICE-level netlist — and generate, via an automated computational procedure, a much smaller macromodel. The macromodel, fundamentally a small system of equations, is usually translated into a description language appropriate for system-level simulation, such as Matlab/Simulink, Verilog-A, VHDL-AMS or a SPICE subcircuit. Such an automated approach, i.e., one that remains sustainable as devices shrink from deep submicron to nanoscale, is essential for realistic exploration of the design space in current and future mixed-signal SoCs/SiPs.

Several broad methodologies for automated macromodelling have been proposed, as depicted in Figure 10.1. One is to generalize, abstract and automate the manual macromodelling process. For example, common topological elements in a circuit are recognized, approximated and conglomerated (e.g., [22, 37]) to create a macromodel. Another class of approaches attempts to capture *symbolic* macromodels that capture the system's input-output relationship, e.g., [10, 88, 110, 111, 112, 116]. Yet another class (e.g., [36, 105, 122]) employs a *black-box* methodology. Data are collected via many simulations or measurements of the full

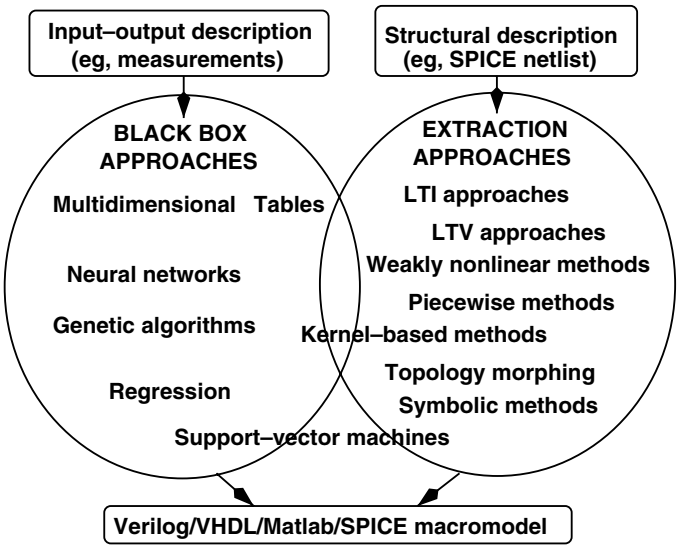


Fig. 10.1 Automated macromodelling approaches.

system and a regression-based model created that can predict outputs from inputs. Various methods are available for the regression, including data mining, multi-dimensional tables, neural networks, genetic algorithms, *etc.*

In this section, we focus on another methodology for macromodelling, often termed *algorithmic*. Algorithmic macromodelling methods approach the problem as the transformation of a large set of mathematical equations to a much smaller one. The principal advantage of these methods is generality — so long as the equations of the original system are available numerically (e.g., from within SPICE), knowledge of circuit structure, operating principles, *etc.* is not critical. A single algorithmic method may therefore apply to *entire classes* of physical systems, encompassing circuits and functionalities that may appear very disparate. Figure 10.2 depicts three such classes, namely linear time invariant (LTI), linear time varying (LTV), and nonlinear, which are discussed in Sections 10.1.1, 10.1.2 and 10.1.3 of this chapter. Algorithmic methods also tend to be more rigorous about important issues such as fidelity and stability, and often provide better guarantees of such characteristics than other methods.

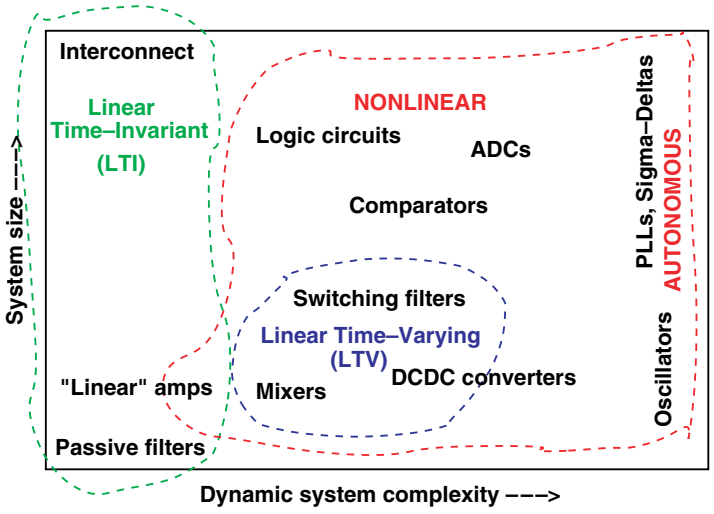


Fig. 10.2 Different classes of systems present different macromodelling challenges.

10.1.1 Macromodelling Linear Time Invariant (LTI) Systems

Often referred to as reduced-order modelling (ROM) or model-order reduction (MOR), automated model generation methods for Linear Time-Invariant (LTI) systems are the most mature amongst algorithmic macromodelling methods. Any block composed of resistors, capacitors, inductors, linear controlled sources and distributed interconnect models is LTI (often referred to simply as "linear"). The development of LTI MOR methods has been driven largely by the need to "compress" the huge interconnect networks, such as clock distribution nets, that arise in large digital circuits and systems. Replacing these networks by small macromodels makes it feasible to complete accurate timing simulations of digital systems at reasonable computational expense. Although interconnect-centric applications have been the main domain for LTI reduction, it is appropriate for any system that is linear and time-invariant. For example, "linear amplifiers", i.e., linearizations of mixed-signal amplifier blocks, are good candidates for LTI MOR methods.

Figure 10.3 depicts the basic structure of an LTI block. $u(t)$ represents the inputs to the system, and $y(t)$ the outputs, in the time domain; in the Laplace (or frequency) domain, their transforms are $U(s)$ and $Y(s)$, respectively. The definitive property of any LTI system [124] is that the input and output are related by convolution with an *impulse response* $h(t)$ in the time domain, i.e., $y(t) = x(t) * h(t)$). Equivalently, their transforms are related by multiplication with a *system transfer function* $H(s)$, i.e., $Y(s) = H(s)X(s)$. Note that there may be many internal nodes or variables within the block. The goal of LTI MOR methods is to replace the block by one with far fewer internal

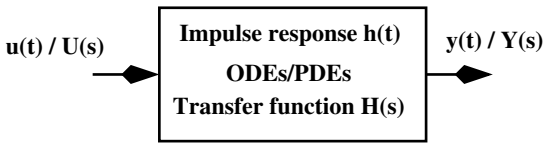


Fig. 10.3 Linear Time Invariant block.

variables, yet with an acceptably similar impulse response or transfer function.

In the majority of circuit applications, the LTI block is described to the MOR method as a set of differential equations, i.e.,

$$\begin{aligned} E\dot{x} &= Ax(t) + Bu(t) \\ y(t) &= C^T x(t) + Du(t) \end{aligned} \tag{10.1}$$

In Equation (10.1), $u(t)$ represents the input waveforms to the block and $y(t)$ the outputs. Both are relatively few in number compared to the size of $x(t)$, the state of the internal variables of the block. A, B, C, D and E are constant matrices. Such differential equations can be easily formed from SPICE netlists or AHDL descriptions; especially for interconnect applications, the dimension n of $x(t)$ can be very large.

The first issue in LTI ROM is to determine what aspect of the transfer function of the original system should be retained by the reduced system; in other words, what metric of fidelity is appropriate. In their seminal 1990 paper [84], Pileggi and Rohrer used *moments* of the transfer function as fidelity metrics, to be preserved by the model reduction process. The moments m_i of an LTI transfer function $H(s)$ are related to its derivatives, i.e.,

$$m_1 = \left.\frac{dH(s)}{ds}\right|_{s=s_0}, \quad m_2 = \left.\frac{d^2H(s)}{ds^2}\right|_{s=s_0}, \ldots, \tag{10.2}$$

where s_0 is a frequency point of interest. Moments can be shown to be related to practically useful metrics, such as delay in interconnects.

In [84], Pileggi and Rohrer proposed a technique, Asymptotic Waveform Evaluation (AWE), for constructing a reduced model for the system (Equation (10.1)). AWE first computes a number of moments of the full system (Equation (10.1)), then uses these in another set of linear equations, the solution of which results in the reduced model. Such a procedure is termed *explicit moment matching*. The key property of AWE was that it could be shown to produce reduced models whose first several moments (at a given frequency point s_0) were identical to those of the full system. The computation involved in forming the reduced model was roughly linear in the size of the (large) original system.

While explicit moment matching via AWE proved valuable and was quickly applied to interconnect reduction, it was also observed

to become numerically inaccurate as the size of the reduced model increased beyond about 10. To alleviate these, variations based on matching moments at *multiple frequency points* were proposed [6] that improved numerical accuracy. Nevertheless, the fundamental issue of numerical inaccuracy as reduced model sizes grew remained.

In 1994, Gallivan et al. [28] and Feldmann/Freund [19, 27] identified the reason for this numerical inaccuracy. Computing the k^{th} moment explicitly involves evaluating terms of the form $A^{-k}r$, i.e., the k^{th} member of the *Krylov subspace* of A and r. If A has well separated eigenvalues (as it typically does for circuit matrices), then for $k \sim 10$ and above, only the dominant eigenvalue contributes to these terms, with non-dominant ones receding into numerical insignificance. Furthermore, even with the moments available accurately, the procedure of finding the reduced model is also poorly conditioned.

Recognizing that these are not limitations fundamental to the goal of model reduction [19, 28] proposed alternatives. They showed that numerically robust procedures for computing Krylov subspaces, such as the Lanczos and Arnoldi (e.g., [102]) methods, could be used to produce reduced models that match any given number of moments of the full system. These approaches, called *Krylov-subspace MOR techniques*, do not compute the moments of the full system explicitly at any point, i.e., they perform *implicit moment matching*. In addition to matching moments in the spirit of AWE, Krylov-subspace methods were also shown to capture well the dominant poles and residues of the system. The Padé-via-Lanczos (PVL) technique [19] gained rapid acceptance within the MOR community by demonstrating its numerical robustness in reducing the DEC Alpha chip's clock distribution network.

Krylov-subspace methods are best viewed as reducing the system (Equation (10.1)) via *projection* [32]. They produce two projection matrices, $V \in \mathcal{R}^{n\times q}$ and $W^T \in \mathcal{R}^{q\times n}$, such that the reduced system is obtained as:

$$\begin{aligned} \underbrace{W^T E}_{\hat{E}}\dot{x} &= \underbrace{W^T A V}_{\hat{A}}x(t) + \underbrace{W^T B}_{\hat{B}}u(t) \\ y(t) &= \underbrace{C^T V}_{\hat{C}^T}x(t) + Du(t). \end{aligned} \tag{10.3}$$

For the reduction to be practically meaningful, q, the size of the reduced system, must be much smaller than n, the size of the original. If the Lanczos process is used, then $W^T V \approx I$ (i.e., the two projection bases are bi-orthogonal). If the Arnoldi process is applied, then $W = V$ and $W^T V = I$.

The development of Krylov-subspace projection methods marked an important milestone in LTI macromodelling. However, reduced models produced by both AWE and Krylov methods retained the possibility of *violating passivity*, or even being *unstable*. A system is passive if it cannot generate energy under any circumstances; it is stable if for any bounded inputs, its response remains bounded. In LTI circuit applications, passivity guarantees stability. Passivity is a natural characteristic of many LTI networks, especially interconnect networks. It is essential that reduced models of these networks also be passive, since the converse implies that under some situation of connectivity, the reduced system will become unstable and diverge unboundedly from the response of the original system.

The issue of stability of reduced models was recognized early in [28], and the superiority of Krylov-subspace methods over AWE in this regard also noted. Silveira et al. [106] proposed a co-ordinate transformed Arnoldi method that guaranteed stability, but not passivity. Kerns et al. [42] proposed reduction of admittance-matrix-based systems by applying a series of non-square congruence transformations. Such transformations preserve passivity properties while also retaining important poles of the system. However, this approach does not guarantee matching of system moments. A symmetric version of PVL with improved passivity and stability properties was proposed by Freund and Feldmann in 1996 [24].

The passivity-retaining properties of congruence transformations were incorporated within Arnoldi-based reduction methods for RLC networks by Odabasioglu et al. [75, 76] in 1997, resulting in an algorithm dubbed PRIMA (Passive Reduced-Order Interconnect Macromodelling Algorithm). By exploiting the structure of RLC network matrices, PRIMA was able to preserve passivity *and* match moments. Methods for Lanczos-based passivity preservation [4, 23] followed.

All the above LTI MOR methods, based on Krylov-subspace computations, are efficient (i.e., approximately linear-time) for reducing large systems. The reduced models produced by Krylov-subspace reduction methods are not, however, optimal, i.e., they do not necessarily minimize the error for a macromodel of given size. The theory of balanced realizations, well known in the areas of linear systems and control, provides a framework in which this optimality can be evaluated. LTI reduced-order modelling methods based on *truncated* balanced realizations (TBR) (e.g., [56, 83]) have been proposed. Balanced realizations are a canonical form for linear differential equation systems that "balance" controllability and observability properties. While balanced realizations are attractive in that they produce more compact macromodels for a given accuracy, the process of generating the macromodels is computationally very expensive, i.e., cubic in the size of the original system. However, recent methods [39] that combine Krylov-subspace techniques with TBR methods have been successful in approaching the improved compactness of TBR, while substantially retaining the attractive computational cost of Krylov methods.

10.1.2 Macromodelling Linear Time Varying (LTV) Systems

LTI macromodelling methods, while valuable tools in their domain, are inapplicable to many functional blocks in mixed-signal systems, which are usually nonlinear in nature. For example, distortion or clipping in amplifiers, switching and sampling behavior, *etc.* , cannot be captured by LTI models. In general, generating macromodels for nonlinear systems (see Section 10.1.3) is a difficult task.

However, a class of nonlinear circuits (including RF mixing, switched-capacitor and sampling circuits) can be usefully modeled as *linear time-varying* (LTV) systems. The key difference between LTV systems and LTI ones is that if the input to an LTV system is time-shifted, it does not necessarily result in the same time shift of the output. The system remains linear, in the sense that if the input is scaled, the output scales similarly. This latter property holds, at least ideally,

for the input-to-output relationship of circuits such as mixers or samplers. It is the effect of a separate local oscillator or clock signal in the circuit, *independent of the signal input*, that confers the time-varying property. This is intuitive for sampling circuits, where a time-shift of the input, relative to the clock, can be easily seen not to result in the same time shift of the original output — simply because the clock edge samples a different time-sample of the input signal. In the frequency domain, more appropriate for mixers, it is the time-varying nature that confers the key property of frequency shifting of input signals. The time-varying nature of the system can be "strongly nonlinear", with devices switching on and off — this does not impact the linearity of the signal input-to-output path.

Figure 10.4 depicts the basic structure of an LTV system block. Similar to LTI systems, LTV systems can also be completely characterized by impulse responses or transfer functions; however, these are now functions of two variables, the first capturing the time-variation of the system, the second the changes of the input [124]. The detailed behavior of the system is described using time-varying differential equations, e.g.,

$$\begin{aligned} E(t)\dot{x} &= A(t)x(t) + B(t)u(t) \\ y(t) &= C(t)^T x(t) + D(t)u(t). \end{aligned} \tag{10.4}$$

Time variation in the system is captured by the dependence of A, B, C, D and E on t. In many case of practical interest, this time-variation is periodic. For example, in mixers, the local oscillator input is often a sine or a square wave; switched or clocked systems are driven by periodic clocks.

The goal of macromodelling LTV systems is similar to that for LTI ones: to replace Equation (10.4) by a system identical in form, but

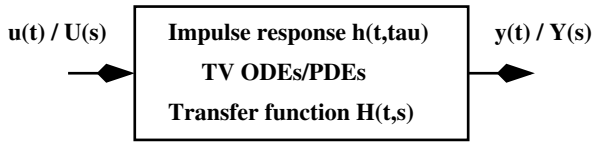


Fig. 10.4 Linear time varying block.

with the state vector $x(t)$ much smaller in dimension than the original. Again, the key requirement is to retain meaningful correspondence between the transfer functions of the original and reduced systems. Because of the time-variation of the impulse response and transfer function, LTI MOR methods cannot directly be applied to LTV systems. However, Roychowdhury [96, 97, 98] showed that LTI model reduction techniques can be applied to LTV systems, by first reformulating Equation (10.4) as an LTI system similar to Equation (10.1), but with extra *artificial inputs* that capture the time-variation. The reformulation first separates the input and system time variations explicitly using multiple time scales [99] in order to obtain an operator expression for $H(t, s)$. This expression is then evaluated using periodic steady-state methods [47, 95, 114] to obtain an LTI system with extra artificial inputs. Once this LTI system is reduced to a smaller one using any LTI MOR technique, the reduced LTI system is reformulated back into the LTV system form Equation (10.4). The use of different LTI MOR methods within this framework has been demonstrated, including explicit moment matching [96] and Krylov-subspace methods [80, 97, 98]. Moreover, Phillips [80] showed that the LTV-to-LTI reformulation could be performed using standard linear system theory concepts [124], without the use of multiple time scales.

10.1.3 Macromodelling Nonlinear Systems

While wires, interconnect and passive lumped elements are purely linear, any mixed-signal circuit block containing semiconductor devices is nonlinear. Nonlinearity is, in fact, a fundamental feature of any block that provides signal gain, or performs any function more complex than linear filtering. Even though linear approximations of many nonlinear blocks are central to their design and intended operation, it is usually important to consider the impact of nonlinearities with a view to limiting their impact. For example, in "linear" amplifiers and mixers, distortion and intermodulation, caused solely by nonlinearities, must typically be guaranteed not to exceed a very small fraction of the output of the linearized system. This is especially true for traditional RF

and microwave designs. Such *weakly nonlinear systems* comprise an important class of blocks that can benefit from macromodelling.

Additionally, many nonlinear blocks of interest are not designed to be approximately linear in operation. Examples include digital gates, switches, comparators, *etc.* , which are intended to switch abruptly between two states. While such operation is obviously natural for purely digital systems, strongly nonlinear behavior is also exploited in analog blocks such as sampling circuits, switching mixers, analog-to-digital converters *etc.* Furthermore, oscillators and PLLs, which are common and basic components in mixed-signal systems, exhibit complex dynamics which are fundamentally strongly nonlinear.

Unlike for the classes of linear systems considered in the previous sections, no technique currently exists that is capable, even in principle, of producing a macromodel that conforms to any reasonable fidelity metric for *completely general* nonlinear systems. The difficulty stems from the fact that nonlinear systems are richly varied, with extremely complex dynamical behavior possible that is very far from being exhaustively investigated or understood. This is in contrast to linear dynamical systems, for which comprehensive mathematical theories exist (see, e.g., [124]) that are universally applicable. In view of the diversity and complexity of nonlinear systems in general, it is difficult to conceive of a single overarching theory or method that can be employed for effective macromodelling of an arbitrary nonlinear block.

It is not surprising, therefore, that macromodelling of nonlinear systems has tended to be manual, relying heavily on domain-specific knowledge for specialized circuit classes, i.e., ADCs, phase detectors, *etc.*

In recent years, however, linear macromodelling methods have been extended to handle weakly nonlinear systems. Other techniques based on piecewise approximations have also been devised that are applicable some strongly nonlinear systems. As described below in more detail, these approaches start from a general nonlinear differential equation description of the full system, but first approximate it to a more restrictive form, which is then reduced to yield a macromodel of the same form. The starting point is a set of nonlinear differential–algebraic

equations (DAEs) of the form:

$$\begin{aligned} \dot{q}(x(t)) &= f(x(t)) + b\,u(t) \\ y(t) &= c^T x(t), \end{aligned} \tag{10.5}$$

where $f(\cdot)$ and $q(\cdot)$ are nonlinear vector functions.

10.1.3.1 Polynomial-based Weakly Nonlinear Methods

To appreciate the basic principles behind weakly nonlinear macromodelling, it is first necessary to understand how the full system can be treated if the nonlinearities in Equation (10.5) are approximated by low-order polynomials. The polynomial approximation concept is simply an extension of linearization, with $f(x)$ and $q(x)$ replaced by the first few terms of a Taylor series about an expansion point x_0 (typically the DC solution); for example,

$$f(x) = f(x_0) + A_1(x - x_0) + A_2(x - x_0)^{\textcircled{2}} + \cdots, \tag{10.6}$$

where $a^{\textcircled{i}}$ represents the Kronecker product of a with itself i times. When Equation (10.6) and its $q(\cdot)$ counterpart are used in Equation (10.5), a system of polynomial differential equations results. If $q(x) = x$ (assumed for simplicity), these equations are of the form:

$$\begin{aligned} \dot{x}(t) &= f(x_0) + A_1(x - x_0) + A_2(x - x_0)^{\textcircled{2}} + \cdots + b\,u(t) \\ y(t) &= c^T x(t). \end{aligned} \tag{10.7}$$

The utility of this polynomial system is that it becomes possible to leverage an existing body of knowledge on weakly polynomial differential equation systems, i.e., systems where the higher-order nonlinear terms in Equation (10.6) are small compared to the linear term. In particular, *Volterra series theory* [103] and weakly nonlinear perturbation techniques [73] justify a relaxation-like approach for such systems, which proceeds as follows. First, the response of the linear system, ignoring higher-order polynomial terms, is computed — denote this response by $x_1(t)$. Next, $x_1(t)$ is inserted into the quadratic term $A_2(x - x_0)^{\textcircled{2}}$ (denoted a *distortion input*), the original input is *substituted* by this waveform, and the *linear* system solved again to obtain a *perturbation due to the quadratic term* — denote this by $x_2(t)$. The sum

of x_1 and x_2 is then substituted into the cubic term to obtain another weak perturbation, the linear system solved again for $x_3(t)$, and so on. The final solution is the sum of x_1, x_2, x_3 and so on. An attractive feature of this approach is that the perturbations x_2, x_3, *etc.* , which are available *separately* in this approach, correspond to quantities like distortion and intermodulation which are of interest in design. Note that at every stage, to compute the perturbation response, a *linear* system is solved — nonlinearities are captured via the distortion inputs to these systems.

The basic idea behind macromodelling weakly nonlinear systems is to exploit this fact; in other words, to apply linear macromodelling techniques, appropriately modified to account for distortion inputs, to each stage of the relaxation process above. In the first such approach, proposed in 1999 by Roychowdhury [98], the linear system is first reduced by LTI MOR methods to a system of size q_1, as shown in Figure 10.5, via a projection basis obtained using Krylov-subspace methods. The distortion inputs for the quadratic perturbation system are then expressed in terms of the *reduced* state vector of the linear term, to obtain an input vector of size q_1^2 . The quadratic perturbation system (which has the same linear system matrix, but a different input vector) is then

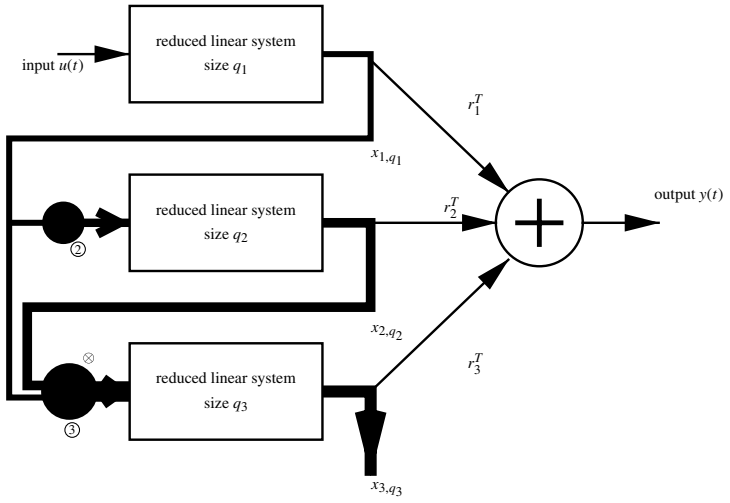


Fig. 10.5 Block structure of reduced polynomial system.

again reduced via another projection basis, to size q_2. This process is continued for higher order terms. The overall reduced model is the union of the separate reduced models with outputs summed together, as depicted in Figure 10.5.

By tailoring projection bases for each nonlinearly perturbed linear system, this approach focusses on accuracy; however, this is achieved at the cost of increased macromodel size $q_1 + q_2 + \cdots$. Recognizing the size issue, Phillips in 2000 [81, 82] proposed that a *single* projection basis be applied to the system (Equation (10.7)) (analogous to LTI MOR systems), and also observed that Carlemann bilinearization [100] could be employed to obtain a canonical equation form. Intuitively, the use of a single projection basis consolidates the commonality in the three reduced models shown in Figure 10.5, leading to smaller overall models.

In 2003, Li and Pileggi proposed the NORM method [57], which combines and extends the above two approaches. Similar to [98], NORM generates tailored projection bases for each perturbed linear system, but instead of retaining separate macromodels as in Figure 10.5, it compresses these projection bases into a single projection basis. NORM then employs this single projection basis to reduce the system (Equation (10.7)) as proposed in [82]. A particularly attractive property of NORM is that it produces a macromodel that matches a number of *multidimensional moments* of the Volterra series kernels [103] of the system — indeed, the distortion terms for each perturbed system are pruned to ensure matching of a specified number of moments. The authors of NORM also include a variant that matches moments at multiple frequency points.

10.1.3.2 Trajectory-Piecewise Approximation Methods

The polynomial approximations discussed above are excellent when the intended operation of the system exercises only weak nonlinearities, as in power amplifiers, "linear" mixers, *etc.* Outside a relatively small range of validity, however, polynomials are well known to be extremely poor global approximators. This limitation is illustrated in Figure 10.6, where it can be seen that, outside a local region where there is a good

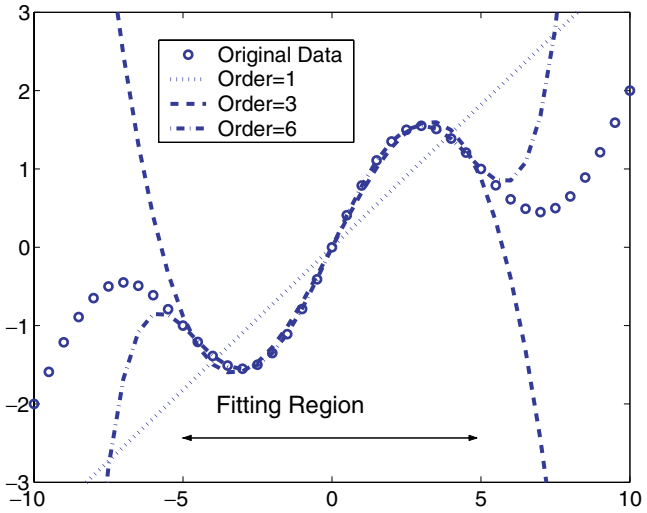


Fig. 10.6 Limitations of global polynomial approximations.

match, even a sixth-degree Taylor-series approximation diverges dramatically from the function it is meant to represent.

It is for this reason that other ways of approximating (Equation (10.5)) that have better global approximation properties than polynomials have been sought. One approach is to represent the nonlinear functions $f(\cdot)$ and $q(\cdot)$ in Equation (10.5) by *piecewise linear (PWL)* segments. The state space is split into a number of disjoint regions, and within each region, a linear approximation is used that matches the nonlinear function approximately within the region. By using a sufficiently large number of regions, the nonlinear function can be represented accurately over the entire domain of interest. From a macromodelling perspective, the motivation for PWL approximations is that since the system is linear within each region, linear macromodelling methods can be leveraged.

Piecewise linear approximations are not new in circuit simulation, having been employed in the past most notably in attempts to solve the DC operating point problem [41, 78]. One concern with these methods is a potential exponential explosion in the number of regions as the dimension of the state space grows. This is especially the case when each

elemental device within the circuit is first represented in piecewise form, and the system of circuit equations constructed from these piecewise elements. A combinatorial growth of polytope regions results, via cross-products of the hyperplanes that demarcate piecewise regions within individual devices.

To circumvent the explosion of regions, which would severely limit the simplicity of a small macromodel, Rewienski and White proposed the Trajectory PWL method (TPWL) [92] in 2001. In TPWL, a reasonable number of "center points" are first selected along a simulation trajectory in the state space, generated by exciting the circuit with a representative training input. Around each center point, system nonlinearities are approximated by linearization, with the region of validity of the linearization defined *implicitly*, as consisting of all points that are closer to the given center point than to any other. Thus there are only as many piecewise regions as center points, and combinatorial explosion resulting from intersections of hyperplanes is avoided. The implicit piecewise regions in TPWL are in fact identical to the Voronoi regions defined by the collection of center points chosen.

Within each piecewise region, the TPWL approach simply reduces the linear system using existing LTI MOR methods to obtain a reduced linear model. The reduced linear models of all the piecewise regions are finally stitched together using a *scalar weight function* to form a single-piece reduced model. The weight function identifies, using a closest-distance metric, whether a test point in the state space is within a particular piecewise region, and weights the corresponding reduced linear system appropriately.

The TPWL method, by virtue of its use of inherently better PWL global approximation, avoids the blow-up that occurs when polynomial-based methods are used with large inputs. It is thus better suited for circuits with strong nonlinearities, such as comparators, digital gates, *etc.* However, because PWL approximations do not capture higher-order derivative information, TPWL's ability to reproduce small-signal distortion or intermodulation is limited.

To address this limitation, Dong and Roychowdhury proposed a piecewise polynomial (PWP) extension [18] of TPWL in 2003. PWP combines weakly nonlinear MOR techniques with the piecewise idea,

by approximating the nonlinear function in each piecewise region by a polynomial, rather than a purely linear, Taylor expansion. Each piecewise polynomial region is reduced using one of the polynomial MOR methods outlined above, and the resulting polynomial reduced stitched together with a scalar weight function, similar to TPWL. Thanks to its piecewise nature, PWP is able to handle strong nonlinearities globally; because of its use of local Taylor expansions in each region, it is also able to capture small-signal distortion and intermodulation well. Thus PWP expands the scope of applicability of nonlinear macromodelling to encompass blocks in which strong and weak nonlinearities both play an important functional rôle. Amongst all currently available techniques, PWP may therefore be considered the most suitable for producing general-purpose macromodels.

For example, Figure 10.7 depicts a moderate-sized differential current-mirror op-amp that was macromodeled using PWP. When the op-amp is used as a linear amplifier with small inputs, linear amplification, distortion and intermodulation are important performance metrics. A comparison of frequency-domain harmonic balance (HB) analysis on the original circuit and the macromodel is shown in Figure 10.8. It can be seen that for the entire input range, there is an excellent match of the linear and distortion components from the macromodel versus that from the full circuit (at very small input magnitudes, the distortion component of both is dominated by numerical

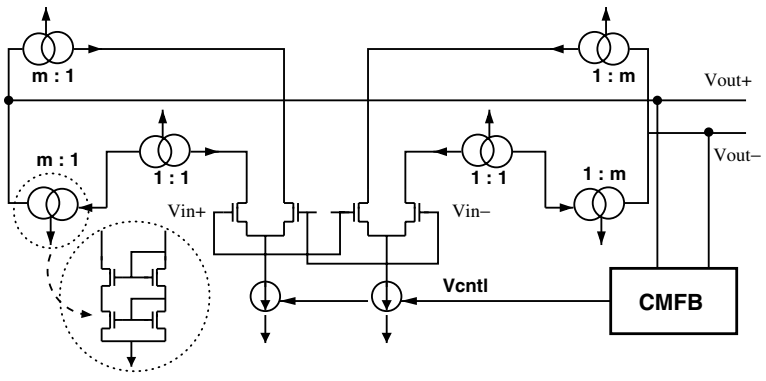


Fig. 10.7 Current-mirror op-amp with 50 MOSFETs and 39 nodes.

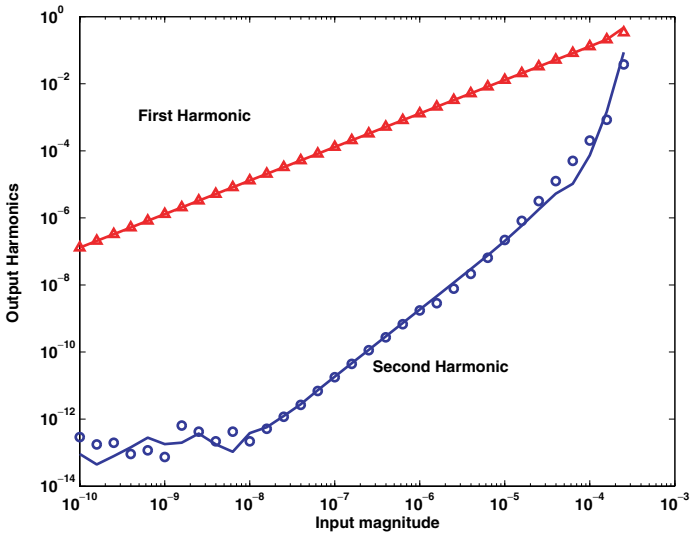


Fig. 10.8 Harmonic analysis of current op-amp. The solid lines represent simulation of the full circuit, while the discrete points represent simulations of the PWP-generated macromodel.

noise). At the same time, the PWP-generated macromodel is able to capture the effects of strong nonlinearities, excited by large signal swings, well. This is evident from the large-signal transient analysis results shown in Figure 10.9. The input was chosen to excite slew-rate limiting, a dynamical phenomenon caused by strong nonlinearities (saturation of differential amplifier structures); hard limiting due to the power and ground rails is also present. The excellent match between the PWP-generated macromodel and the original circuit is evident. The macromodel-based transient simulation was an order of magnitude faster than simulation of the original; this is in fact a conservative number, with further significant speedups to be expected, for example, simply from improved program structuring of the PWP algorithm.

10.2 Oscillator Phase Macromodelling and Applications

Oscillators are critical components in electronic and optical systems. They have many uses; for example, they are used to generate signals for frequency up/down-conversion in communication systems, as

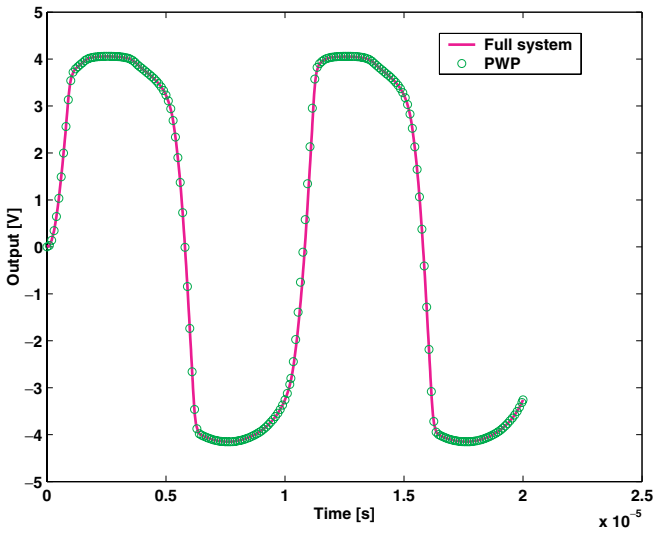


Fig. 10.9 Transient analysis of the current mirror op-amp, showing slewing and clipping.

clocks in digital systems, and so on. Oscillators are also important in phase-locked loops (PLLs) [108], which are widely used in both digital and analog circuits for clock generation and recovery, frequency synthesis, signal conditioning, and synchronization, *etc.* The design of oscillators and oscillator-based systems is an important part of overall system design; however, simulations of such systems present certain unique challenges, addressing which calls for specialized methods and techniques.

SPICE-level transient simulation is not well suited for oscillator-based systems, for which it can be very inefficient. The transient response of oscillators can last hundreds of thousands of cycles, with each cycle requiring many small time steps for accurate simulation of the embedded oscillator. A special problem, unique to oscillatory circuits, is that during their transient simulation, phase errors accumulate without limit. This problem, which does not afflict stable non-oscillatory circuits, stems from a fundamental property of oscillators, *neutral phase stability*. Because of this, taking time steps that would be adequate for other kinds of circuits can result in large phase errors for oscillators. To control this accuracy problem, very small time steps (e.g., hundreds of time steps per oscillatory cycle) are typically

taken, exacerbating the inefficiency already associated oscillators due to widely separated time scales of response.

To address the above issues, specialized simulation and analysis techniques have been developed for oscillators. One popular approach toward improving the efficiency of simulating oscillator-based systems is to use *oscillator phase macromodels*. Phase macromodels attempt to capture the relationship between inputs to an oscillator and the phase of its output waveform using equations that are simpler than the full DAEs describing the oscillator. Note that phase and its time-derivative the instantaneous frequency are typically the quantities of primary interest for most oscillator applications. The most basic class of phase macromodels assumes a *linear relationship* between input perturbations and the output phase of an oscillator. The most general linear model for the output phase $\phi(t)$ is given by a (time-varying) expression of the form:

$$\phi(t) = \sum_{k=1}^{n} \int_{-\infty}^{\infty} h_{\phi}^{k}(t,\tau) i_k(\tau) d\tau. \tag{10.8}$$

The summation is over all perturbations i_k to the circuit; $h_{\phi}^{k}(t,\tau)$ denotes a time-varying impulse response to the k^{th} noise source. Very frequently, time-invariant simplifications of Equation (10.8) are employed [46], where $h_{\phi}^{k}(t,\tau)$ are independent of t, and even of τ.

Linear models suffer, however, from a number of important deficiencies. For example, they have been shown to be inadequate for capturing fundamentally nonlinear effects such as injection locking [52]. As a result, *nonlinear phase macromodels* [15, 17, 52, 53, 54], which are considerably more accurate than linear ones, have in recent years taken over as the approach of choice. Nonlinear phase macromodels have the form:

$$\dot{\alpha}(t) = v_1^T(t + \alpha(t)) \cdot b(t). \tag{10.9}$$

In the above equation, $\alpha(t) = \frac{\phi(t)}{f_0}$ is a time shift (f_0 is the frequency of unperturbed oscillator). $v_1(t)$ is an important quantity, called the *perturbation projection vector (PPV)* [17]; it is a periodic vector function of time, with the same period as that of the unperturbed oscillator. A key

difference between the nonlinear phase macromodel (Equation (10.9)) and traditional linear phase models is the inclusion of the time shift $\alpha(t)$ within the argument of the PPV function. This makes the phase macromodel *implicit* as well as nonlinear, in contrast to the linear phase macromodel (Equation (10.8)). The nonlinearity of the macromodel has been shown to be the crucial factor enabling it to capture a wide variety of interesting dynamical effects. Furthermore, this nonlinear time-shifted phase macromodel has been proven to be exact for capturing oscillator phase responses to small perturbations.

Furthermore, it is possible to augment such phase-only macromodels with additional amplitude components, that capture the most important, slowly dying, modes of the oscillator's amplitude variations.

This results in comprehensive phase + amplitude macromodels, which are able to replicate a wide gamut of oscillator behaviors at far lower computational cost than full SPICE-level simulation of the original circuit, while at the same time substantially alleviating the phase error accumulation problem. Important design effects that are captured by such macromodels include the following:

- *Injection locking*: Injection locking is a nonlinear dynamical phenomenon that is often exploited in electronic and optical oscillator design (e.g., injection locked lasers, ultra-high-frequency frequency dividers). Nonlinear time-shifted phase macromodels are able to capture injection locking well while providing significant computational speedups over full SPICE-level circuit simulation. It is to be noted that linear phase macromodels can be proven to be inherently incapable of capturing locking phenomena.
- *PLL capture/lock/slipping/interference dynamics*: PLL malfunction due to loop non-idealities is one of the most important factors leading to re-fabs of SoCs. Fast and accurate simulation of PLLs, accounting for the effects of power supply interference, non-ideal filtering, real VCOs, *etc.* , is of great importance. Nonlinear oscillator phase macromodels are considerably more accurate than linear ones for PLL applications. They are able to predict the

dynamics of complex phenomena such as phase pulling, cycle slipping and power supply noise induced PLL jitter, and to replicate qualitative and quantitative features from full SPICE simulations accurately while providing great speedups.

- *Coupled oscillator dynamics*: Coupled self-oscillating systems appear in diverse natural and physical systems (Josephson junctions, spintronics, many different kinds of biological systems, cosmic structures, *etc.*). Fast and accurate simulation of the dynamics of such systems is therefore of significant theoretical and practical interest in a multitude of disciplines. Nonlinear phase macromodels are able to predict the behavior of coupled oscillating systems very effectively, with large speedups over full transient simulation.

10.2.1 Nonlinear Oscillator Phase Macromodel

Let the oscillator under perturbation be described by the ODE:

$$\frac{d}{dt}\vec{x} + f(\vec{x}) = \vec{b}(t), \tag{10.10}$$

where $\vec{b}(t)$ is a vector of perturbation signals applied to the free running oscillator.[1] After an unperturbed ($b(t) \equiv 0$) periodic steady state $x_s(t)$ is obtained numerically, the system can be linearized as follows:

$$\begin{aligned}\frac{d}{dt}\vec{w}(t) \approx & -\frac{\partial f(\vec{x})}{\partial \vec{x}}|_{\vec{x}_s(t)}\vec{w}(t) + \vec{b}(t) \\ = & A(t)\vec{w}(t) + \vec{b}(t).\end{aligned} \tag{10.11}$$

$\vec{w}(t)$ represents deviations from $x_s(t)$ that result from the perturbation $b(t)$. The periodic time-varying linear system (Equation (10.11)) can be solved using Floquet theory [33] to obtain an expression for its state transition matrix:

$$\Phi(t,\tau) = U(t)\exp(D(t-\tau))V(\tau). \tag{10.12}$$

[1] We use the ODE form here merely for simplicity of exposition; the results and techniques obtained are valid for DAE systems as well.

In the above, $U(t)$ and $V(t)$ are T-periodic nonsingular matrices, satisfying biorthogonality conditions $\vec{v}_i^T(t)\vec{u}_j(t) = \delta_{ij}$, and $D = diag[\mu_1, ..., \mu_n]$, where μ_i are the Floquet exponents. As proven in [16], one of the Floquet exponents must be 0 for oscillators. Assuming that first Floquet exponent $\mu_1 = 0$, it can be shown that $\vec{u}_1(t) = \frac{d}{dt}\vec{x}_s(t)$ is a solution of $\dot{\vec{w}}(t) = A(t)\vec{w}(t)$, the homogenous part of Equation (10.11). Similarly, $\vec{v_1}(t)$ can be shown to solve the adjoint of the homogeneous part of Equation (10.11); further, it can also be shown that $\vec{v}_1^T(t)\vec{u}_1(t) \equiv 1, \forall t$. $\vec{v}_1(t)$, which plays a central rôle in the oscillator's nonlinear macromodel, is termed the *Perturbation Projection Vector* or PPV. It can be thought of as capturing the oscillator's "nonlinear" phase sensitivity to perturbations.

The utility of the PPV lies in that once it is known for a given oscillator, the phase (or time) deviations of the oscillator due to external perturbations are entirely captured via the scalar, nonlinear differential equation Equation (10.9), which constitutes the nonlinear phase macromodel. Various methods [12, 14, 16, 17] in the time and frequency domains are available for calculating the PPV from SPICE-level circuit descriptions of oscillators (or equivalently, from full DAE descriptions).

10.2.2 Augmenting Phase Macromodels with Amplitude Modes

Once the phase deviation $\alpha(t)$ is obtained by solving Equation (10.9), a macromodel for dominant amplitude components can be built as well, by linearizing the oscillator over its perturbed time-shifted orbits $\vec{x}_s(t + \alpha(t))$. The oscillator is first linearized about $\vec{x}_s(t + \alpha(t))$:

$$\begin{aligned} \frac{d}{dt}\vec{y}(t) &\approx -\frac{\partial f(\vec{x})}{\partial \vec{x}}|_{\vec{x}_s(t+\alpha(t))}\vec{y}(t) + \vec{b}(t) \\ &= A(t + \alpha(t))\vec{y}(t) + \vec{b}(t), \end{aligned} \tag{10.13}$$

where $\vec{x}_s(t)$ is the oscillator's steady-state orbit, $\alpha(t)$ is the phase deviation due to perturbation $\vec{b}(t)$, and $\vec{y}(t)$ is a small amplitude deviation from the phase-shifted orbit, due to the perturbation $\vec{b}(t)$. By introducing a new variable $\hat{t} = t + \alpha(t)$ and defining $\hat{y}(\hat{t}) = \vec{y}(t)$ and $\hat{b}(\hat{t}) = \vec{b}(t)$,

we obtain a linear periodic time-varying (LPTV) system:

$$\frac{d}{dt}\hat{y}(\hat{t}) = A(\hat{t})\hat{y}(\hat{t}) + \hat{b}(\hat{t}). \tag{10.14}$$

Applying Floquet decomposition, the LPTV input–output relationship can be expressed as:

$$\hat{y}(\hat{t}) = \sum_{i=1}^{n} u_i(\hat{t}) \int_0^{\hat{t}} \exp(\mu_i(\hat{t} - \tau)) v_i^T(\tau)\hat{b}(\tau)d\tau. \tag{10.15}$$

By dropping the μ_1 term, corresponding to the marginally stable phase mode, as well as other, less important Floquet exponents, a reduced amplitude macromodel can be obtained. With phase deviation $\alpha(t)$ and amplitude variation $y(t)$ available, the oscillator's orbit under perturbation is given by:

$$\vec{x}_p(t) = \vec{x}_s(t + \alpha(t)) + \vec{y}(t), \tag{10.16}$$

where $\vec{x}_s(t)$ is the steady state orbit of the oscillator, and $\vec{x}_p(t)$ is the orbit of the oscillator under perturbation.

Macromodelling consists of replacing the oscillator's original DAEs with the nonlinear time-shifted phase equation Equation (10.9) and the reduced amplitude macromodel (Equation (10.15)). The macromodel can be simulated using any transient simulator, typically with large speedups compared to the full system, due to its smaller size. Moreover, since the macromodel is simulated in the phase domain directly, additional savings are typically possible (e.g., using larger time steps and simpler integration methods) without appreciable loss of accuracy.

10.2.3 Predicting Injection Locking

Injection locking is an interesting and useful phenomenon universally observed in all kinds of physical oscillators. The term refers to the fact that, under certain conditions, when an oscillator is perturbed by an external weak signal whose frequency is close (but not identical) to the oscillator's natural frequency, the oscillator's frequency changes to become identical to that of the perturbing signal — i.e., it "locks" to the external signal.

Most approaches toward understanding and predicting injection locking are all directly based on Adler's classic 1946 paper [1], which provides a simplified quantitative explanation of the phenomenon for simple harmonic oscillators, leading to formulas for their lock range. Adler's approach is not general, being limited to LC harmonic oscillators and relying on analytical simplifications. Indeed, it requires the Q factor of the oscillator, therefore cannot be applied to, e.g., ring oscillators, for which Q factors are not defined. One of the main uses of the nonlinear phase macromodels mentioned above is that they are capable of capturing injection locking in any kind of oscillator.

The nonlinearity of the phase macromodel (Equation (10.9)) enables it to capture injection locking effects in any oscillator. If the oscillator locks to an injected signal, the oscillator's phase follows that of the injected signal; this leads to the relationship:

$$\omega_0 t + \phi(t) = \omega_1 t + \theta, \text{ or } \quad \phi(t) = (\omega_1 - \omega_0)t + \theta, \tag{10.17}$$

where ω_0 is the frequency of the free-running oscillator, ω_1 is the frequency of the injected signal, $\phi(t)$ is the phase deviation of the perturbed oscillator, and θ is a constant which represents the phase difference between the locked oscillator and the injected signal. It is clear from Equation (10.17) that if the oscillator locks to the injected signal, the phase shift due to the injected signal should grow with time linearly with a slope of $\omega_1 - \omega_0$ (i.e., the difference between injected and natural frequencies). Since $\alpha(t)$ has units of time, the phase deviation in radians can be expressed as:

$$\phi(t) = \omega_0 \alpha(t). \tag{10.18}$$

Substituting Equation (10.18) into Equation (10.17), we have

$$\omega_0 \alpha(t) = (\omega_1 - \omega_0)t + \theta, \text{ or } \quad \alpha(t) = \frac{\Delta\omega_0}{\omega_0}\, t + \frac{\theta}{\omega_0}, \tag{10.19}$$

where $\Delta\omega_0 = \omega_1 - \omega_0$ is the frequency difference between the free-running oscillator and the injected signal.

This relationship provides a simple means to check for locking behavior in oscillators. For example, if an oscillator is injected with a signal with frequency 10% higher than its free-running frequency,

using Equation (10.19), the oscillator locks to the signal if its phase shift $\alpha(t)$ increases linearly with a slope of 0.1.

It is also possible to obtain an estimate of the lock range of the oscillator. Substituting Equation (10.19) into the nonlinear phase equation (10.9), we obtain

$$\begin{aligned}\frac{\Delta\omega_0}{\omega_0} &= v_1(t + \frac{\Delta\omega_0 t + \theta}{\omega_0})A_{inj}\sin(\omega_1 t) \\ &= v_1(\frac{\omega_1 t + \theta}{\omega_0})A_{inj}\sin(\omega_1 t),\end{aligned} \tag{10.20}$$

where A_{inj} is the amplitude of the injection signal and θ is the phase difference between the injection signal and the oscillator's output. Since the PPV $v_1(t)$ has the same frequency as the free running oscillator, the frequency of $v_1(\frac{\omega_1}{\omega_0}t)$ must equal the injection frequency ω_1. As $\Delta\omega_0$ in Equation (10.20) increases, the nonlinear locking mechanism changes the locked phase difference θ to match the slope $\frac{\Delta\omega_0}{\omega_0}$. Since Equation (10.20) is T_1-periodic, integrating both sides for one period of T_1 leads to

$$\int_0^{T_1}\frac{\Delta\omega_0}{\omega_0}dt = \int_0^{T_1} v_1(\frac{\omega_1 t + \theta}{\omega_0})A_{inj}\sin(\omega_1 t)dt \tag{10.21}$$

or

$$\frac{\Delta\omega_0}{\omega_0} = \frac{A_{inj}}{T_1}\int_0^{T_1} v_1(\frac{\omega_1 t + \theta}{\omega_0})\sin(\omega_1 t)dt. \tag{10.22}$$

Hence, the maximum locking range is given by

$$|\frac{\Delta\omega_0}{\omega_0}| < \eta A_{inj}, \tag{10.23}$$

where

$$\begin{aligned}\eta &= \max_{\theta=0\to 2\pi}(\frac{1}{T_1}\int_0^{T_1} v_1(\frac{\omega_1 t + \theta}{\omega_0})\sin(\omega_1 t))dt \\ &= \max_{t_0=0\to 1}(\int_0^1 v_1(\frac{t + t_0}{f_0}))\sin(2\pi t)dt.\end{aligned} \tag{10.24}$$

η is independent of the injection frequency f_1, and can be easily calculated by numerical methods if the PPV is available. Note that Equation (10.23) has a form similar to the Adler equation. Unlike Adler's

equation, however, it can apply to any physical oscillator, regardless of operating mechanism.

10.2.4 Example: 1 GHz LC Oscillator

Figure 10.10 depicts the schematic of an LC oscillator whose differential equations are:

$$\begin{aligned} -C\frac{d}{dt}v(t) &= \frac{v(t)}{R} + i(t) + S\tanh\Big(\frac{G_n}{S}v(t)\Big) + b(t) \\ L\frac{d}{dt}i(t) &= v(t). \end{aligned} \tag{10.25}$$

$L = 4.869 \times 10^{-7}/(2\pi)\ H$, $C = 2 \times 10^{-12}/(2\pi)\ F$, $R = 100\ \Omega$, $S = 1/R$ and $G_n = -1.1/R$. With these parameters, the LC tank has a resonant frequency of 1 GHz, and in steady-state oscillation, the inductor current has amplitude $A_0 = 1.2$ mA.

Equation (10.23) reveals a linear relationship between the injection amplitude A_{inj} and the frequency difference $\Delta\omega_0$; given the PPV, the slope η in Equation (10.23) can be calculated using Equation (10.24). Figure 10.11 plots the locking range of the LC oscillator as a function of injection amplitude. The nonlinear phase macromodel can capture injection locking well when the injection amplitude is less than about 15% of A_0. η, which can be calculated very quickly (in a few seconds), can be used to predict injection locking by evaluating Equation (10.23). In contrast, full SPICE-like simulation requires 4 min to predict locking, even for this relatively low-Q oscillator. η needs to be calculated only once; it can then be reused to investigate injection locking under different injection frequencies and strengths. A similar feature is not

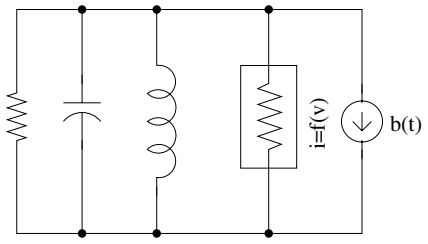
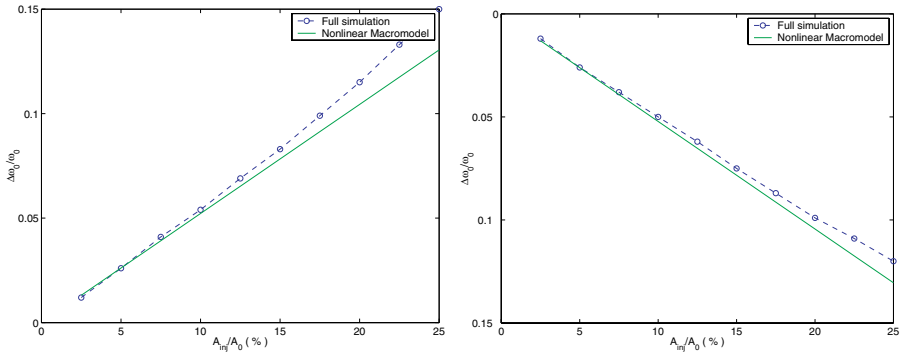


Fig. 10.10 A simple LC oscillator.



(a) Injection frequency higher than oscillator frequency

(b) Injection frequency lower than oscillator frequency

Fig. 10.11 Simple LC oscillator: locking range.

available when solving the full system of equations. Furthermore, when the Q-factor of the LC oscillator is made high, thousands of cycles elapse before lock. The simulation time for the full DAEs becomes more than 1 hour, while that for the macromodel remains, of course, unchanged.

10.2.5 Example: Three-Stage Ring Oscillator

Figure 10.12 is a block diagram of a three-stage ring oscillator described by the differential equations:

$$\begin{aligned} -C_1 \frac{d}{dt} v_1(t) &= \frac{v_1}{R1} - \frac{\tanh(G_{m3} v_3(t))}{R_1}, \\ -C_2 \frac{d}{dt} v_2(t) &= \frac{v_2}{R2} - \frac{\tanh(G_{m1} v_1(t))}{R_2}, \\ -C_3 \frac{d}{dt} v_3(t) &= \frac{v_3}{R3} - \frac{\tanh(G_{m2} v_2(t))}{R_3}. \end{aligned} \tag{10.26}$$

Each stage of this ring oscillator is identical, with $C_1 = C_2 = C_3 = 2nF$, $R_1 = R_2 = R_3 = 1\ \mathrm{k}\Omega$, and $G_{m1} = G_{m2} = G_{m3} = -5$. The oscillator has a natural frequency of 153,498 Hz.

As mentioned earlier, a key advantage of nonlinear phase macromodels is their general applicability. Using Equation (10.24), the maximum locking range of this ring oscillator is easily predicted, as

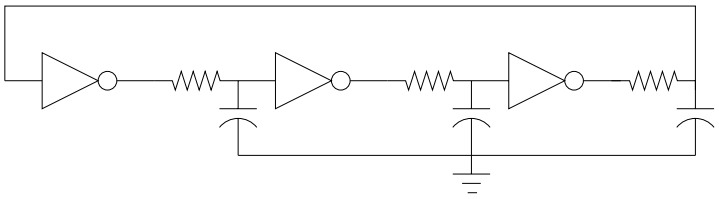
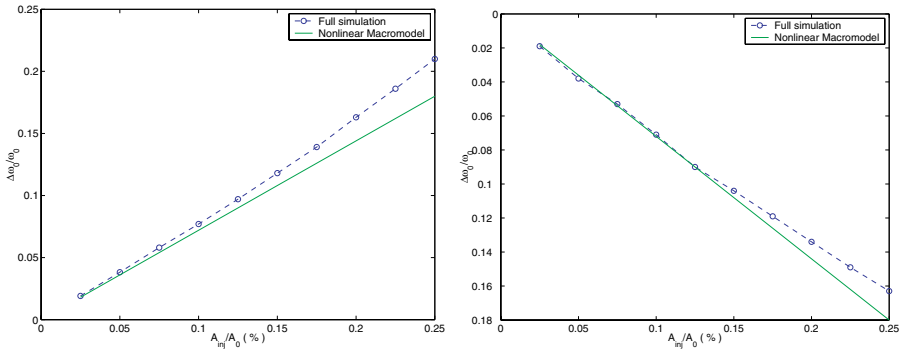


Fig. 10.12 A three-stage ring oscillator.



(a) Injection frequency higher than oscillator frequency

(b) Injection frequency lower than oscillator frequency

Fig. 10.13 Ring oscillator: locking range.

plotted in Figure 10.13. As before, locking is predicted well when the injection amplitude is below about 10% of A_0. For this case, η is calculated in several seconds, while full SPICE-like simulation requires about five minutes to capture locking.

10.3 Using Oscillator Phase Macromodels for PLL Analysis

PLLs [108] are extensively used in all analog and digital systems. In spite of their having been used over decades, modern PLL design still presents significant challenges. In actual fabrications and real designs, PLLs often do not function as they are ideally supposed to, due to second-order effects such as non-ideal filters, interference from power supply lines, loop stability issues, *etc.* In modern RF and mixed-signal designs (especially for SoCs), functional degradation of PLLs is a major cause of overall system malfunction, often resulting in many design

and fabrication re-spins. Costs of such re-fabs can run into the millions of dollars and significantly delay product time-to-market. As a result, accurate simulation of PLLs is of great practical importance.

However, direct time-domain simulation of PLLs at the level of SPICE circuits is typically impractical because of its great inefficiency. PLL transients can last hundreds of thousands of cycles, with each cycle requiring hundreds of small time steps for accurate simulation of the embedded voltage-controlled oscillator (VCO). Furthermore, extracting phase or frequency information, one of the chief metrics of PLL performance, from time-domain voltage/current waveforms is often difficult and inaccurate.

To alleviate the inefficiency of simulating PLLs at the SPICE level, it is very common during PLL design to use phase domain macromodels (e.g., [46]) to simplify the system. Each building block of a PLL (such as the VCO, the phase/frequency detector, and the low-pass filter) is represented approximately using small, simple macromodels, and the system of macromodels simulated. Furthermore, in contrast to SPICE-level circuits which use voltage/current domain device models, the PLL block macromodels used are typically in the phase domain. For example, it is common for the VCO to be represented as a simple linear integrator that converts input voltages to output phases; similar simple macromodels of other blocks are also employed. The use of such macromodels can lead to dramatic speedups (of many orders of magnitude over the SPICE level); however, such speedups are obtained at the expense of accuracy.

In particular, *linear* PLL block macromodels, while appropriate for simple, locked PLLs in near-ideal operation (e.g., the low-pass filter (LPF) rejects high frequencies well, there is no interference from power supply/ground lines, *etc.*), can be seriously inadequate for predicting the complex non-ideal phenomena that are typically responsible for performance degradation in today's industrial PLLs. For example, high-bandwidth PLLs (e.g., [60, 91]), popular in read channel and clock recovery applications for their fast tracking properties, are difficult to simulate accurately using linear macromodels because the high-frequency components transmitted from the phase/frequency detector to the VCO excite nonlinear mechanisms critical for capture/lock and

cycle slipping. Such mechanisms often result in changes to the *static phase offset* of the PLL in lock; these changes in static offset consume additional phase budgets for a stable lock and increase the likelihood of *cycle slipping* (e.g., [35]). Linear methods do not take these non-idealities into consideration, thus resulting in incorrect predictions. Moreover, charge pump PLLs (e.g., [2, 74]), widely used for frequency synthesis, usually feature a feed-forward channel bypassing the low-pass filter to ensure loop stability. The high-frequency components transmitted through these feed-forward paths again invalidate linear macromodels.

Furthermore, linear VCO macromodels typically have difficulty in accounting for supply- or substrate-interference-induced *PLL phase jitter* (e.g., [46, 64]). In the deep-submicron technologies in predominant use today, interference-induced jitter has grown to be a primary cause of PLL performance degradation. Since jitter is mainly created by direct interference to the VCO, the (ideally slow) loop dynamics of the PLL is unable to eliminate it appreciably; again, fast nonlinear mechanisms are excited which linear macromodels [13, 29, 34, 55, 109, 117] are ill-suited to cope with [16].

The nonlinear phase macromodel (Equation (10.9)) is successful, however, in alleviating virtually all these deficiencies. Phase noise in the PLL's reference signal, high-frequency components from the phase/frequency detector, power supply, ground and substrate interference, *etc.* , are all accounted for correctly.

10.3.1 Linear PLL Phase Macromodels

A PLL typically consists of a phase/frequency detector (PFD), a low-pass filter (LPF), a voltage-controlled oscillator (VCO) and a frequency divider (FD). The FD is usually used when the PLL is a frequency synthesizer. The PFD compares the phase difference between the reference frequency f_{ref} and the feedback frequency f_{fb} to produce a voltage which corresponds to the phase difference between two signals. Since the output of the PFD has high-frequency AC components, an LPF is applied to filter out these AC components and produce a DC voltage to set the VCO's frequency. In intended operation, the PLL

constitutes a stable negative feedback system in which the divided VCO frequency f_{fb} is forced to be identical to the reference frequency f_{ref}. Given an input frequency f_{ref}, the frequency at the output of the PLL is $f_{out} = N f_{ref}$.

As already mentioned earlier, direct time-domain simulation of PLLs is very expensive since many cycles typically elapse before the system reaches steady state. This is especially true for frequency synthesizers with large multiplication factors. Simulating the PLL directly in phase can save significant computation because the VCO is treated as a phase generator; no oscillator is involved in the simulation.

In linear phase domain macromodels, the PFD is modeled as $K_{pd} f(\Delta\phi)$, where K_{pd} is the gain of the PFD and $f(\Delta\phi)$ is a transformation function between the phase difference and the output voltage of the PFD. This function depends on the type of PFD; for example, if the PFD is a mixer/multiplier, the transformation function is typically taken to be $f(\Delta\phi) = \sin(\Delta\phi)$; if the PFD is an XOR phase detector, the transformation function is $f(\Delta\phi) = \Delta\phi \mod 2\pi$.

The VCO is an oscillator that uses its input voltage to determine its natural frequency. In the linear models typically used in PLL simulation, the VCO's frequency is assumed to be affinely related to the instantaneous input:

$$f_{out}(t) = K_{VCO}\, v_c(t) + f_0. \tag{10.27}$$

In the above, f_0 is the VCO's natural frequency, K_{VCO} is the "gain" of the VCO and $v_c(t)$ is the control voltage from the LPF. The output phase of the VCO is:

$$\phi_{out}(t) = 2\pi f_0 t + 2\pi \int K_{VCO} v_c(t) dt. \tag{10.28}$$

This linear VCO macromodel is often adequate for the specific case of small perturbations to a PLL in lock. However, in general, it is not suitable for capturing a variety of important nonlinear dynamical effects during PLL operation.

10.3.2 PLL Simulation Using Nonlinear VCO Macromodels

In nonlinear PLL macromodels the VCO is no longer a simple linear integrator; instead, it is replaced by an the nonlinear PPV phase

macromodel (Equation (10.9)). The PPV macromodel can accommodate multiple input signals, including not only the control signal from the LPF, but also other perturbations, e.g., from noise in the power supply. Equation (10.9) can be expanded as:

$$\dot{\alpha}(t) = V_{vc}(t + \alpha(t))v_c(t) + V_n^T(t + \alpha(t))n(t), \tag{10.29}$$

where $v_c(t)$ is the control voltage from the LPF, $V_{vc}(t)$ is the PPV component corresponding to the VCO's control node, $V_n(t)$ is a vector of PPV waveforms representing the VCO's phase sensitivity to noise injected into the corresponding circuit nodes, and $n(t)$ is a vector of noise signals affecting the VCO. Note that the key difference between the nonlinear phase macromodel and linear model is the inclusion of the time shift $\alpha(t)$ inside the PPV term $V_n(t)$; further, the gain of the VCO $V_{vc}(t)$ is no longer a constant. The phase of the VCO (in radians) can be expressed as:

$$\phi_{out}(t) = \omega_0 \cdot (t + \alpha(t)), \tag{10.30}$$

where ω_0 is the VCO's free-running frequency.

Because the nonlinear phase macromodel is capable of accounting for high-frequency injection effects (indeed, these constitute an important non-ideality), it is desirable that the model of the PFD retains any high-frequency components that may be being passed. The PFD is modeled as $K_{pd}f(\phi_1, \phi_2)$, where K_{pd} is the gain of the PFD and $f(\phi_1, \phi_2)$ is a function that takes two phases as inputs and produces a voltage as output. If we use a mixer/multiplier as the PFD, the function can be defined as:

$$\begin{aligned} f(\phi_1, \phi_2) &= k_{pd}\left(\sin(\phi_1 - \phi_2) + \sin(\phi_1 + \phi_2)\right) \\ &= k_{pd}\left[\sin(\phi_{ref}(t) - \omega_0(t + \alpha(t)) \right. \\ &\qquad \left. + \sin(\phi_{ref}(t) + \omega_0(t + \alpha(t))\right], \end{aligned} \tag{10.31}$$

where $\phi_{ref}(t)$ is the phase of the reference signal, ω_0 is the VCO's free-running frequency, and $\alpha(t)$ is the VCO's time shift computed by Equation (10.29). To build a phase macromodel for the entire PLL system, the transfer function $H(s)$ of the LPF is converted to ODE/DAE form and combined with the equations of the nonlinear VCO phase model.

10.3.3 PLL Macromodelling and Simulation Case Study

We apply the approach outlined in Section 10.1.3 to a PLL and use it to predict pull-in dynamics, cycle slipping and injection locking.

The PLL circuit used is shown in Figure 10.14. The PLL uses a Gilbert Cell mixer as its phase detector and an op-amp based low-pass filter with relatively high bandwidth. The center frequency of the LC VCO was designed to be f_0=100 MHz.

Characterizing the phase detector. Since the output of the PFD is defined by Equation (10.31), the only parameter that requires characterization is the gain, k_{pd}. This quantity can be obtained from the results of full circuit simulation on the PFD. Simulations with different input phases are performed and the relationship between the input phase difference and the output voltage of the PFD plotted, as shown in Figure 10.15.

It is apparent from the figure that the PFD is very linear in its working range: k_{pd} for this phase detector is -0.16.

Equations for the LPF. The transfer function of the LPF can be derived from the circuit diagram directly to be

$$H(s) = \frac{3 + 2\tau s}{(1 + \tau s)^2}, \tag{10.32}$$

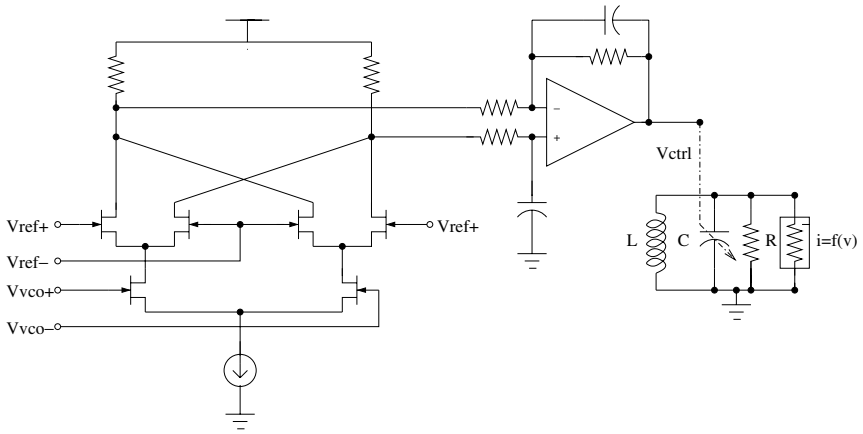


Fig. 10.14 An LC-oscillator based PLL.

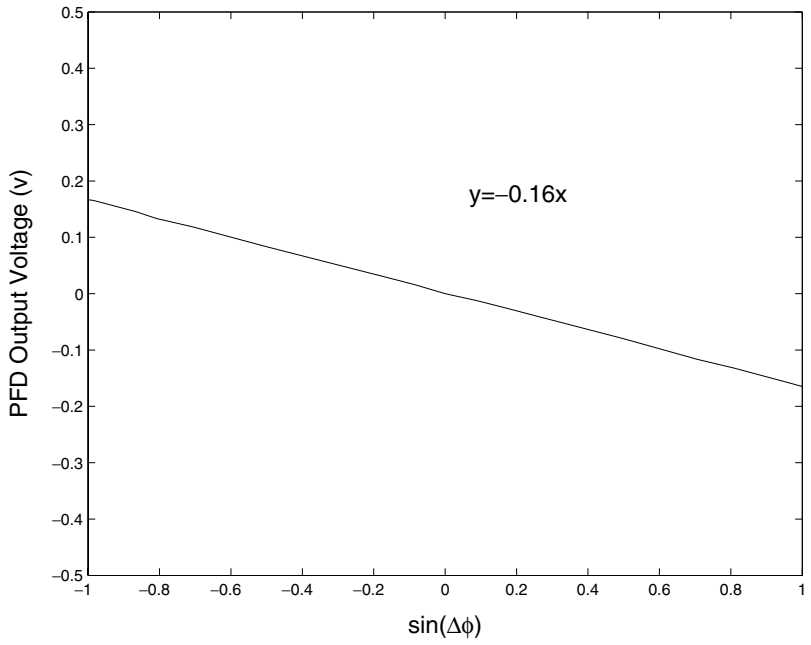


Fig. 10.15 Gain of the phase detector.

where τ is the time constant of the RC network. This transfer function can be expressed as an ODE using the companion form method:

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -\frac{x_1(t)}{\tau^2} - \frac{2}{\tau} x_2(t) + b(t), \end{cases} \tag{10.33}$$

where $b(t)$ is the output of the phase detector. The output of the LPF is

$$y(t) = 3x_1(t) + 2\tau x_2(t). \tag{10.34}$$

VCO macromodel and PPV characterization. The PPV of the VCO is calculated numerically [16, 17], directly from its SPICE-level circuit equations. The control voltage component of the PPV obtained is shown below in Figure 10.16.

It can be clearly seen that the VCO is not an LTI system, since the phase sensitivity of the control node is very dependent on when the control signal is applied.

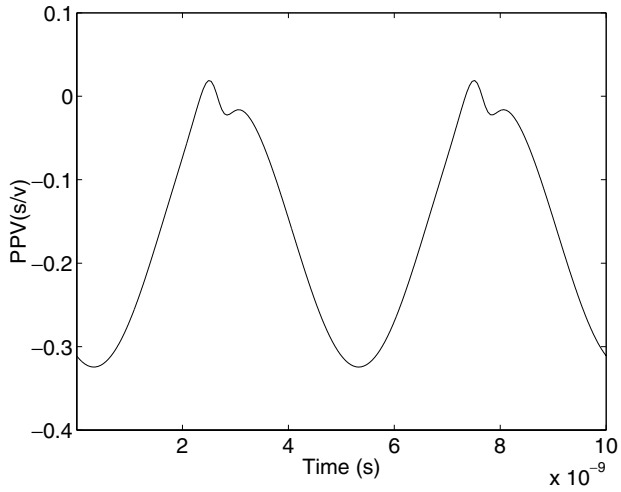


Fig. 10.16 Control node component of the LC VCO's PPV.

PLL system macromodel. Combining the equations of the LPF, the PFD and the VCO together, the PLL system macromodel is obtained to be

$$\begin{cases} \dot{\alpha}(t) = V_{vc}(t + \alpha(t))(3x_1(t) + 2\tau x_2(t)) + V_n^T(t)n(t), \\ \dot{x}_1(t) = x_2(t), \\ \dot{x}_2(t) = -\frac{x_1(t)}{\tau^2} - \frac{2}{\tau}x_2(t) + K_{pd}f(\phi_{ref}, \phi_{fb}), \end{cases} \tag{10.35}$$

where $f(\phi_{ref}, \phi_{fb})$ is given in Equation (10.31), and $n(t)$ represents noise signals to be applied to the VCO. It takes only about 1 min (in MATLAB) to simulate 600 cycles of the VCO using Equation (10.35). Simulating the original equations of the PLL, for the same number of cycles in the same MATLAB environment, takes about 150 min.

Static Phase Offset prediction. The average difference between the phases of the two inputs to the PFD, when the PLL is in lock, is known as the static phase offset. In an ideal PLL, this offset will be zero if the frequency of the reference signal equals the VCO's free-running frequency (since this will result in a zero control input when the PLL is in lock). However, if the LPF is not perfect and high-frequency components from the PFD are leaked into the VCO, the static phase

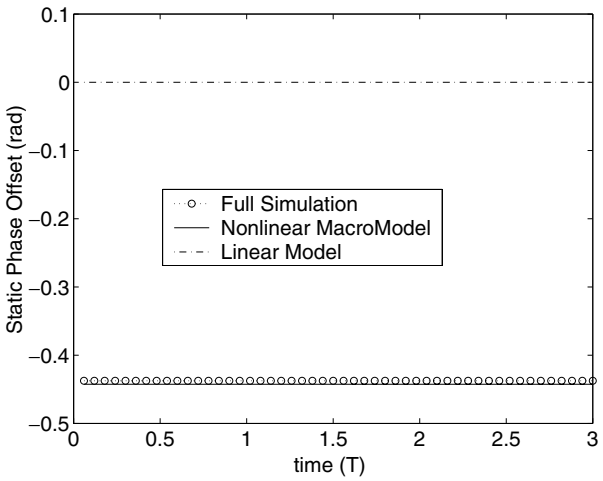
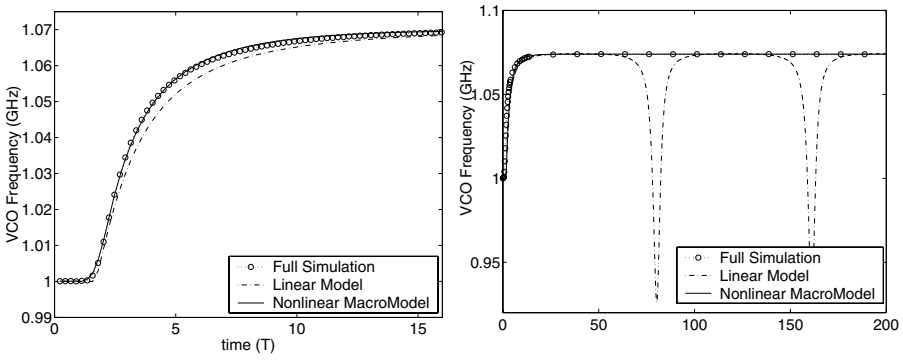


Fig. 10.17 PLL static phase offset when $f_{ref} = f_0$.

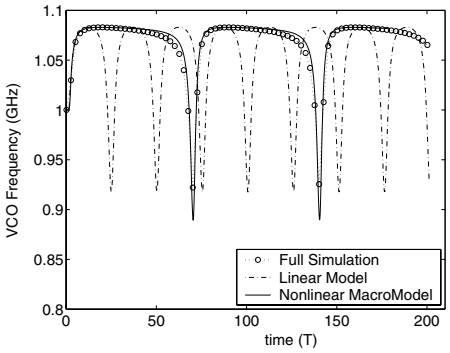
offset of the PLL is affected by injection pulling/locking phenomena in the VCO. Degradation of static phase offset due to such non-idealities is an important reason for overall PLL malfunction, leading, e.g., to increased probability of cycle slipping, loss of reference acquisition, *etc.*

Figure 10.17 depicts the static phase offset of the PLL when a reference with same frequency as the VCO's free-running frequency is applied. The effect of LPF imperfections on the static phase offset is clearly seen: both full simulation and our the nonlinear PLL macromodel predict a static phase offset of about 0.43 rad. However, linear VCO/PLL phase macromodels cannot capture this correctly, predicting a static phase offset of 0.

PLL response to abrupt reference frequency changes. Another important design metric for PLLs is their response to abrupt changes in reference frequency (or equivalently, to changes in frequency division ratio). Figure 10.18(a) depicts changes in VCO frequency as a function of time obtained using full simulation, the linear phase macromodel and the nonlinear macromodel, respectively, when the reference frequency is changed abruptly from f_0 to $1.07f_0$ (at time $t = 0$). Both linear and nonlinear macromodels track the reference frequency well, although, as expected, the nonlinear model provides a more accurate



(a) $f_{ref} = 1.07 f_0$ (b) $f_{ref} = 1.074 f_0$



(c) $f_{ref} = 1.083 f_0$

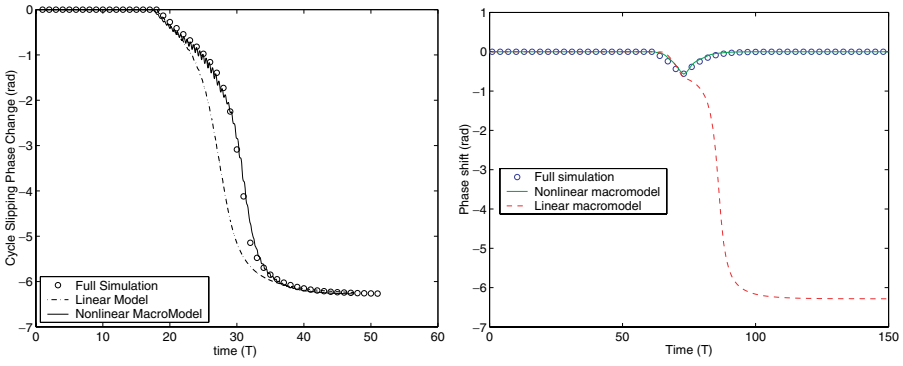
Fig. 10.18 PLL response to abrupt jumps in reference frequency.

simulation than the linear one. If the frequency jump is increased to $1.074 f_0$, however, the linear phase macromodel is unable to track the reference correctly, as shown in Figure 10.18(b); the nonlinear macromodel, however, remains accurate. When the jump is increased further to $1.083 f_0$, i.e., to beyond the pull-in range of the PLL, the PLL genuinely fails to lock, as predicted by both full simulation and the nonlinear macromodel (Figure 10.18(c)).

Prediction of PLL cycle slipping Once a PLL is phase locked to its reference, local negative feedback tends to maintain this lock so long as the phase error lies between its slip boundaries. As the phase error

increases (due to, e.g., external perturbations), the phase detector's output (Equation (10.31)) does not continue to be proportional to the phase difference between its inputs; instead, the outputs saturates and then reverses in direction. This can lead to loss of lock when a temporary disturbance to the PLL creates a large phase error, a phenomenon known as cycle slipping. Cycle slipping occurs when the phase error of the PLL accumulates to such a critical point that its feedback loop is unable to correct the error, resulting in a phase jump or "slip" from one locked steady-state point to another. Cycle slipping degrades the frequency estimation capability of PLLs and produces isolated bursts of large phase noise.

To induce cycle slipping, a momentary sinusoidal current is injected into the VCO, after the PLL is locked to a reference frequency $f_{ref} = 1.07 f_0$. The loop phase error that results over time is plotted in Figure 10.19. When the sinusoid perturbation has an amplitude of 5 mA over 10 periods, cycle slipping results, as shown in Figure 10.19(a). Full simulation, nonlinear and linear phase macromodels all predict cycle slipping, with the nonlinear macromodel matching the full simulation better than the linear one. When the injection amplitude is reduced to 3 mA, however, the linear macromodel erroneously predicts cycle slipping, whereas full simulation and the nonlinear macromodel predict recovery to the original locked state, as seen in Figure 10.19(b).



(a) 5*mA* sinusoidal injection for 10 cycles (b) 3*mA* sinusoidal injection for 10 cycles

Fig. 10.19 PLL cycle slipping.

Jitter and cycle slipping due to power supply noise in a ring VCO. Nonlinear phase macromodels are also able to capture phase noise and jitter resulting from power supply interference. Power supply and substrate interference is of concern for PLLs in most large chips today. Most on-chip PLLs that are subject to supply interference[2] use ring. The ring oscillator VCO in Figure 10.20 is used to replace the LC one of Figure 10.14. The PPV of the ring oscillator VCO is shown in Figure 10.21.

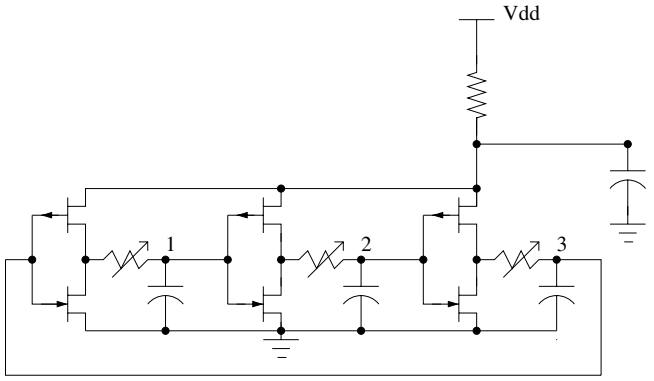
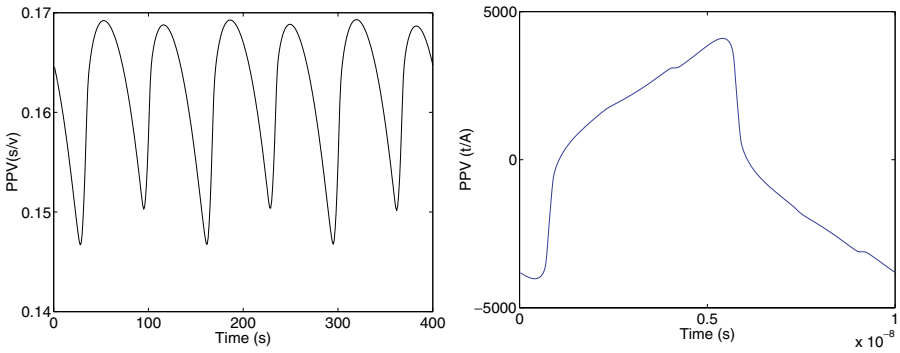


Fig. 10.20 A ring oscillator VCO.



(a) PPV of the control node (b) PPV of the stage-one output node

Fig. 10.21 PPV of ring oscillator VCO.

[2] Such PLLs are typically found in digital chips. For example, recent microprocessors use about 50 PLLs for clock distribution alone.

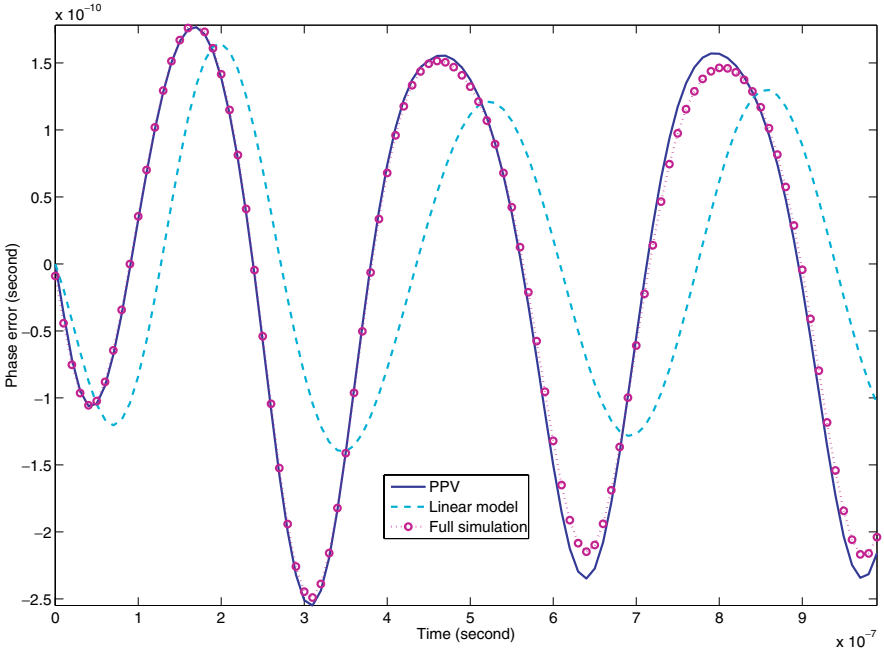


Fig. 10.22 Phase error due to sinusoidal supply interference with frequency $f = 3.03 f_0$.

A sinusoidal current perturbation of amplitude $A = 0.1$ mA and frequency $f = 3.03 f_0$ is injected into the power supply node and the PLL simulated for 100 cycles. The PLL's phase error is shown in Figure 10.22. The sinusoidal interference generates a periodic phase shift in the PLL system. The maximum phase error is about 250 ps, or 2.5% of the PLL's period. Note that the linear VCO macromodel does not match full simulation well, while the nonlinear macromodel does. The time-shift component of the nonlinear phase macromodel results in the negative part of the frequency sensitivity waveform being stretched and the positive part being compressed. As a result, the jittered waveforms have a negative DC value. The linear model, which is unable to capture such shifts, produces a more symmetric waveform, shown in Figure 10.22.

We also investigate the effect of interference frequency on phase errors by sweeping the interference frequency from $0.1 f_0$ to $10 f_0$, keeping the amplitude is fixed at 0.1 mA. Figure 10.23 plots the

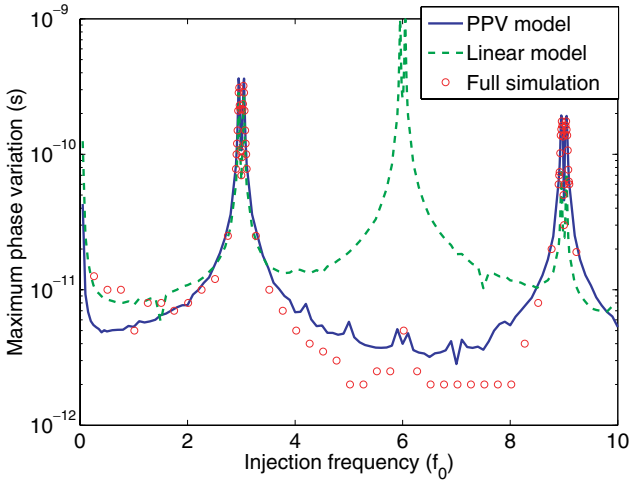
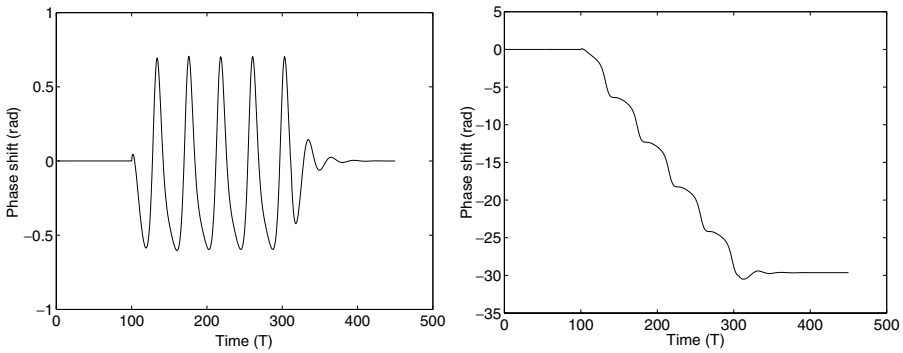


Fig. 10.23 PLL phase error as a function of interference frequency.

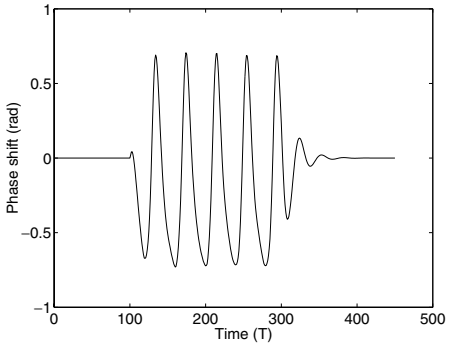
maximum phase error in the presence of such interference, while the PLL is still in lock. Observe that the phase error peaks at the third and ninth harmonics of the VCO's natural frequency; this is due to the fact that the three-stage ring oscillator's PPV waveforms have a strong third harmonic component and a relatively weak fundamental. Full simulation and the nonlinear phase macromodel agree well, while the linear model predicts a large spurious peak at the sixth harmonic.

The simulations above were for small perturbations that did not result in loss of lock; for larger perturbations, cycle slipping and loss of lock results. Starting with a PLL locked to a reference frequency of $1.06 f_0$, periodic perturbations are injected to node 1 of the VCO in Figure 10.20. The resulting phase shifts are shown in Figure 10.24. The frequency of the perturbation is 2.5% lower than that of the reference, and the duration is 200 nominal cycles. As can be seen, an injection amplitude of 0.01 mA is not sufficient to drive the PLL out of lock, while an amplitude of 0.02 mA is. With the latter amplitude, the PLL stops locking to the reference signal and instead locks to the injected perturbation signal, resulting in continuous phase loss. As before, the nonlinear phase macromodel replicates full simulation well, while the linear phase macromodel does not.

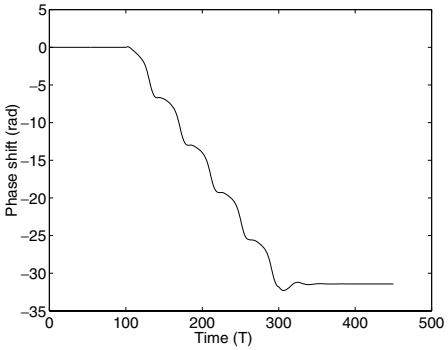


(a) Full simulation (injection amplitude = 0.01mA)

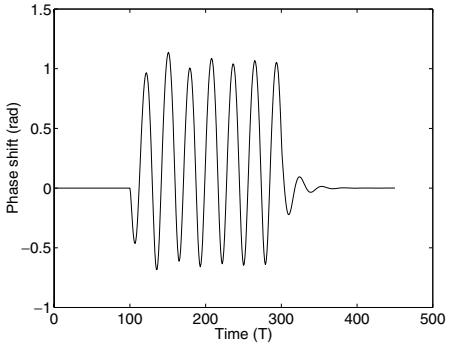
(b) Full simulation (injection amplitude = 0.02mA)



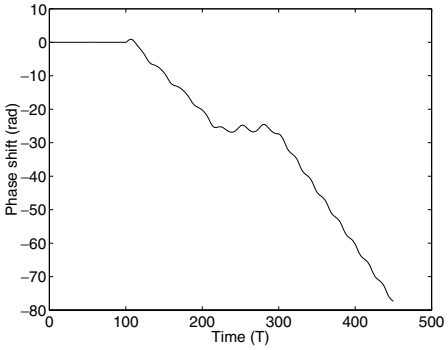
(c) Nonlinear Macromodel (injection amplitude = 0.01mA)



(d) Nonlinear Macromodel (injection amplitude = 0.02mA)



(e) Linear model (injection amplitude = 0.01mA)



(f) Linear model (injection amplitude = 0.02mA)

Fig. 10.24 PLL phase shifts for different injection amplitudes.

10.4 Biochemical and Nanoelectronic Applications of Nonlinear Oscillator Phase Macromodels

Coupled nonlinear oscillators arise in a wide variety of engineering and physical systems. For example, in microwave and optical systems, power combining can be achieved using coupled oscillators [68, 101]. In nanoelectronics, the tunneling phase logic (TPL) device [44, 77] makes use of the bistability of single-electron tunnelling oscillation to realize logic encoded in phase; this concept has been proposed [123] for implementing cellular nonlinear networks (CNNs) [8, 9] with ultra-high integration levels, far beyond DSM CMOS. Potential uses of such CNNs include large-scale high-performance image processing systems.

Large systems of coupled oscillators are particularly plentiful in biology. From the level of individual cells and below, to organ systems, all the way up to systems of many interacting organisms, coupled oscillators are ubiquitous. For example, all living cells contain endogenous biochemical oscillators, of period roughly 24 hours, coupling between which is fundamental to living organisms' daily sleep/wake cycles (*circadian rhythms*). Heart muscle tissue consists of millions of coupled heart muscle cells (*myocytes*), each of which is an electro-mechano-chemical oscillator; the combined dynamics of these coupled oscillators (in conjunction with coupling with other oscillators from the heart's internal neural system) lead to the heart's remarkable pumping properties. Swarms of flashing fireflies, which interact optically, can spontaneously synchronize their flashing; similar synchronization, but in sound, occurs in groups of chirping insects such as crickets and cicadas. Pattern formation in biology, a fascinating and extensively noted phenomenon (e.g., [43, 45, 63]), can be explained well using coupled oscillator models.

It has long been known empirically that in systems of coupled oscillating entities, self-organizing collective behavior begins to occur when coupling exceeds a certain threshold; some entities start to synchronize spontaneously while others remain incoherent. Such "co-operation" and "competition" engendered by oscillatory nonlinear dynamics can produce complicated and beautiful spatio-temporal patterns of collective behavior. These patterns can be explained by reaction diffusion theory [115], which shows that a system of reacting

and diffusing chemicals can evolve spontaneously, from an initially uniform or random state, into spatial patterns known as Turing structures [115]. Reaction–diffusion systems can be viewed as networks of oscillators interacting with each other. An example is the Brusselator reaction-diffusion system [87], pattern formation in which has been studied by experiments and simulation, including under periodic perturbations [58, 59].

Although the dynamics of coupled oscillating systems have been well studied experimentally, analytical understanding of the details of pattern formation remains a challenge. A fundamental difficulty is that while it is often possible to understand specific systems of a few coupled oscillators at an analytical level, collective self-organization typically manifests itself only when *large numbers* of oscillators are networked. Many important characteristics of such large networks have been essentially impossible to understand fully using hand analysis so far, in part because pattern generation is often very sensitive to small details of the nature of and coupling between individual network entities [85, 104]. Thus, analytical simplifications that may be justified for small systems are potentially inapplicable to much larger ones and can lead to egregious mispredictions. As a result, accurate and detailed computational techniques are particularly critical for developing "understanding" of such systems. Because of the huge sizes and complex dynamics of these oscillatory networks, however, the computational problem is very challenging. It is in this context that nonlinear phase macromodels of oscillators are of great value.

Indeed, empirically derived nonlinear phase macromodels for oscillators have been used in biology for decades. One of the earliest studies was made by the legendary Norbert Wiener [119, 120], who studied collective synchronization phenomena using a Fourier-integral-based method. Later, Winfree formulated an equation governing phase transitions in populations of coupled limit-cycle oscillators and presented the concept of a phase sensitivity function [121]. Winfree's approach was abstracted by Kuramoto and applied to systems of identical oscillators with equally weighted, all-to-all, purely sinusoidal couplings, resulting in a well-known phase-based nonlinear differential equation model for coupled oscillator systems, the *Kuramoto model* [50, 51].

Using his model, Kuramoto developed a steady-state "locking" (or "drifting") condition which has been shown to successfully predict bifurcation of phase transitions in coupled oscillator systems. Kuramoto's model has been extremely influential because it has provided a relatively simple means of understanding self-organizing phenomena in systems of coupled oscillators. For predicting detailed pattern formation in a variety of real systems, however, Kuramoto's approach has accuracy limitations that can compound, in large networks, to the extent that wrong patterns can emerge, as we show in this section.

The use of the PPV-based nonlinear phase macromodel (Equation (10.9)), however, alleviates the lack of applicability of Kuramoto's model, while retaining its advantages of relative simplicity and computational efficiency. In some situations (e.g., strong loading effects due to coupling, unlocked mutually pulling oscillators, *etc.*), amplitude variations are large and can couple significantly with phase effects, hence it is necessary to take them into account. In such situations, the additional amplitude macromodel components already mentioned are valuable. A key advantage of the PPV phase + amplitude macromodel is that it is extracted numerically, using automated abstraction algorithms; as a result, it is easily and conveniently applicable to a diverse variety of large and complex oscillator systems.

In the remainder of this chapter, we first provide a review of the Winfree and Kuramoto nonlinear phase macromodels. We then illustrate the use of PPV nonlinear phase macromodels on networks of biochemical and nanoelectronic oscillators: a large, coupled Brusselator chemical dynamics system [87] and a TPL-based cellular nonlinear network (CNN) [123]. We demonstrate formation of two important pattern families in biochemical systems: target patterns and spiral wave patterns, showing that they closely match measurements reported in the bio-chemical literature [63, 45, 43]. Furthermore, we show that Kuramoto's model is unable to predict the correct patterns. For the TPL-CNN network, we use PPV macromodels to confirm bistable network behavior and to demonstrate pattern formation and image processing abilities (e.g., edge detection), predicting patterns very close to measurements reported in [123]. Speedups of almost three orders of

magnitude over direct time-stepping simulation result from the use of PPV nonlinear phase macromodels.

10.4.1 Winfree's Coupled Oscillator Model

For the study of phase dynamics in coupled oscillator systems, a fruitful approach pioneered by Winfree in 1967 [121] is based on the intuition that if an oscillator is perturbed by an external input at a given moment, its phase change should equal to the product of the perturbation strength and a "phase sensitivity" of the oscillator at that moment. Based on this intuition, Winfree formulated a governing set of equations for a coupled oscillator systems with the assumptions of weak coupling and nearly identical oscillators. This system of equations is given by

$$\dot{\theta}_i = \omega_i + \left(\sum_{j=1}^{n} X(\theta_j) \right) Z(\theta_i), \quad i = 1, \ldots, n, \tag{10.36}$$

where n is the number of oscillators in the system, θ_i is the phase of oscillator i, θ_j is the phase of oscillator j and ω_i is the free-running frequency of oscillator i. $X(\theta_j)$ denotes the influence on oscillator i exerted by oscillator j, and $Z(\theta_i)$ is the phase sensitivity function of oscillator i. Each individual component of Winfree's equation is essentially equivalent to the PPV nonlinear phase macromodel Equation (10.9). The chief difficulty with using Winfree's system of equations is that no clear or convenient means is provided for obtaining the phase sensitivity functions; typically, these are measured empirically. The derivation behind the PPV macromodel puts such equation systems on a rigorous theoretical footing; in addition, it provides accurate and convenient computational techniques for finding the PPVs (or phase sensitivity functions).

10.4.2 Kuramoto's Coupled Oscillator Model

In an attempt to put Winfree's empirically-based model on a theoretical footing, Kuramoto [50] applied asymptotic expansion theory [113] and averaging to show that the long-term phase dynamics of weakly-coupled

simple-harmonic-type oscillators can be predicted using

$$\dot{\theta}_i = \omega_i + \sum_{j=1}^{n} \Gamma_{ij}(\theta_j - \theta_i), \quad i = 1, \ldots, n, \tag{10.37}$$

where n is the system size and Γ_{ij} are "interaction functions". Since asymptotic expansion theory applies to structural perturbations of simple harmonic oscillators, these interaction functions have a sinusoidal form. Further simplifications (identical oscillators with equally weighted, all-to-all, and purely sinusoidal coupling) result in

$$\Gamma_{ij}(\theta_j - \theta_i) = \frac{K}{n} \sin(\theta_j - \theta_i), \quad i = 1, \ldots, n, \tag{10.38}$$

where K is a coupling factor. Kuramoto's phase equation then becomes

$$\dot{\theta}_i = \omega_i + \frac{K}{n} \sum_{j=1}^{n} \sin(\theta_j - \theta_i), \quad i = 1, \ldots, n. \tag{10.39}$$

To visualize phase dynamics, Kuramoto introduced the complex "order parameter"

$$re^{i\psi} = \frac{1}{n} \sum_{j=1}^{n} e^{i\theta_j}, \tag{10.40}$$

where $r(t)$ is a radius which measures phase coherence, and $\psi(t)$ is an average phase. Equating the imaginary parts of Equation (10.40), the right-hand side of Equation (10.39) can be rewritten as:

$$\frac{K}{n} \sum_{j=1}^{n} \sin(\theta_j - \theta_i) = Kr \sin(\psi - \theta_i). \tag{10.41}$$

Thus, Equation (10.39) becomes

$$\dot{\theta}_i = \omega_i + Kr \sin(\psi - \theta_i). \tag{10.42}$$

Kuramoto showed that the locking condition of oscillator i in a coupled system is

$$|\Delta\theta_i| \leq Kr, \tag{10.43}$$

where $\Delta\theta_i$ is the difference between the free-running frequency of oscillator i and the mean frequency of the coupled system.

As already noted, and demonstrated later in this section, the sinusoidal simplifications inherent in Kuramoto's model make its predictions inaccurate when applied to large systems of oscillators that are non-sinusoidal.

10.4.3 PPV-Based Coupled Oscillator Phase Macromodels

Since $\vec{v}_1(t)$ in Equation (10.9) is a vector each element of which represents the phase sensitivity to perturbations of the corresponding state variable of the oscillator, and $\vec{b}(t)$ is also a vector of size n comprising separate perturbations, the PPV phase macromodel is capable of modelling any kind of coupling in a network of oscillators. For expositional simplicity, assume that a network consists of identical oscillators coupled through only one state variable, which has a PPV component of $v(t)$ and an unperturbed steady state waveform $x(t)$. This leads to the following system of phase equations for the network:

$$\dot{\alpha}_i(t) = v(t + \alpha_i(t)) \cdot \gamma_i(t), \qquad i = 1, \ldots, n, \tag{10.44}$$

where n represents the number of oscillators in the coupled system, $\alpha_i(t)$ the time shift of oscillator i due to perturbations from coupling and $\gamma_i(t)$ the perturbations to oscillator i that stem from coupling with other oscillators. If $\gamma_i(t)$ and $v(t)$ are purely sinusoidal, Equation (10.44) reduces to Kuramoto's model. However, when the coupling and the phase sensitivity functions are not purely sinusoidal, Equation (10.44) is typically far more accurate than Kuramoto's model.

10.4.4 Pattern Formation in a Brusselator Biochemical Network

Patterns (such as animal fur patterns and human fingerprints) are common in biology. Reaction-diffusion systems, involving oscillatory partial differential equations (PDEs), such as the Brusselator and Belouzov–Zhabotinsky systems, have been successful in explaining pattern formation. Solving reaction diffusion systems numerically involves spatial discretization of the PDEs, leading to very large networks of coupled

oscillators. Simulation of these large coupled oscillatory systems is a computational challenge, which the use of PPV phase macromodels can alleviate significantly. Here, we apply PPV macromodelling to a Brusselator biochemical system.

A Brusselator is an oscillating chemical system with two chemical "species" (e.g., molecules) whose concentrations are represented by $u(x,y,t)$ and $v(x,y,t)$. Their spatial and temporal evolution is governed by

$$\begin{aligned} \frac{\partial u}{\partial t} &= A - (B+1)u + (1 + \gamma \cdot \sin(2\pi f t))u^2 v + D_u \nabla^2 u, \\ \frac{\partial v}{\partial t} &= Bu - u^2 v + D_v \nabla^2 v. \end{aligned} \tag{10.45}$$

$A = 0.5$ and $B = 1.5$ are constant parameters corresponding to feed concentrations, $\gamma \sin(2\pi f t)$ is the external force applied to the oscillator; D_u and D_v are diffusion coefficients.

When the diffusion terms $D_u \nabla^2 u$ and $D_v \nabla^2 v$ are discretized in space, a large network of locally coupled oscillators results. We simulate a 400×400 network ($160,000$ oscillators) of such Brusselator oscillators. The diffusion terms correspond to resistive coupling. Interestingly, the coupled system spontaneously generates spatial patterns, the nature of which varies as coupling resistance R changed.

For coupling resistance $R = 15$, results generated by simulating the network using PPV phase macromodels are shown in Figure 10.25. In the figure, colors are used to represent the oscillator phase. Each oscillator in the network is initialized with a random phase pattern at time $t = 0$. After five nominal cycles of oscillation ($t = 5T$), collective synchronization can be clearly seen: neighbouring oscillators tend to synchronize phase with their neighbours and many coloured spots appear. After 20 cycles, several small target patterns develop, and a spiral wave pattern forms (boundary, center). From $t = 40T$ onwards, the patterns grow and merge. After 150 cycles, complex pattern results. This pattern, obtained using PPV phase macromodels, is qualitatively similar to experimental measurements reported in [59, 63]. When the coupling strength R is changed, different types of patterns result, as shown in Figure 10.26.

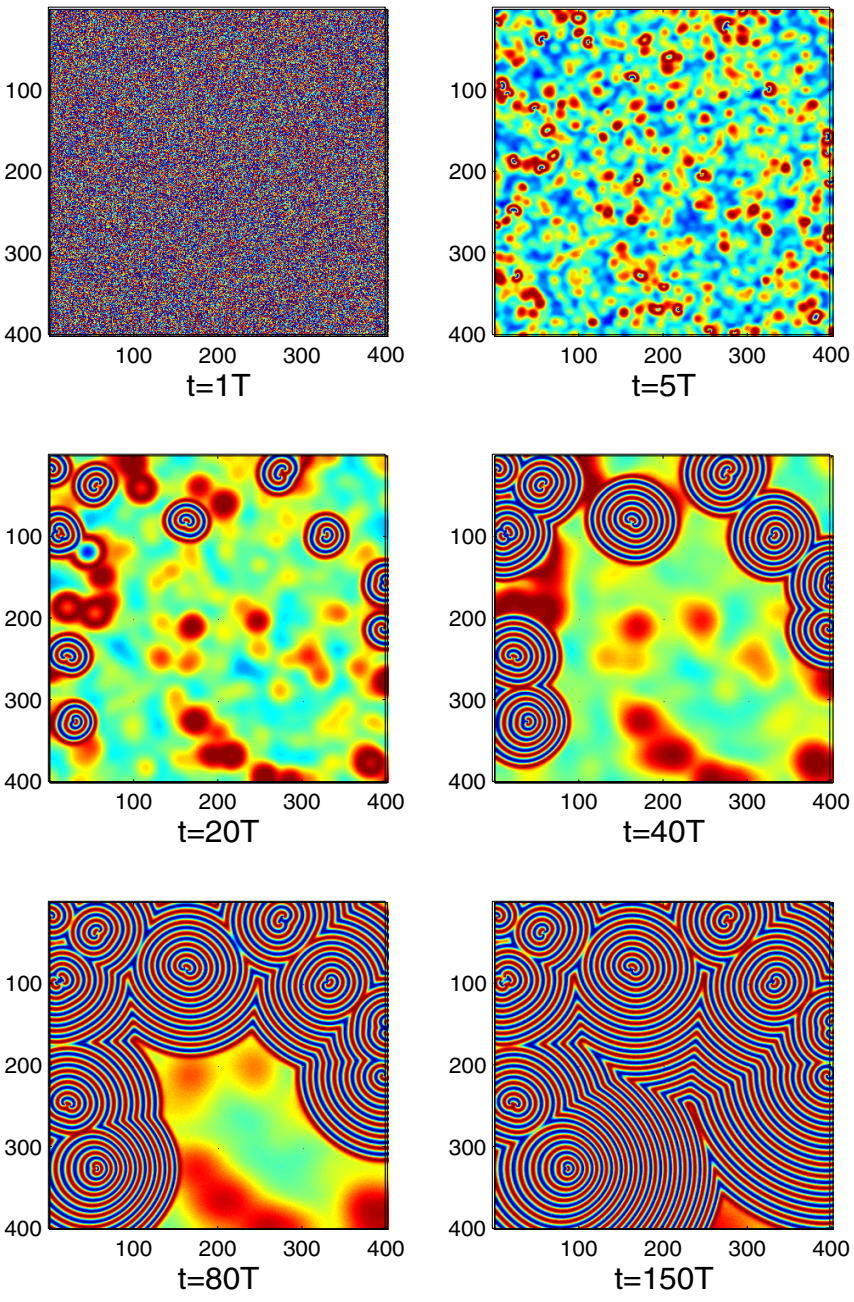


Fig. 10.25 Pattern formation in an unforced Brusselator system.

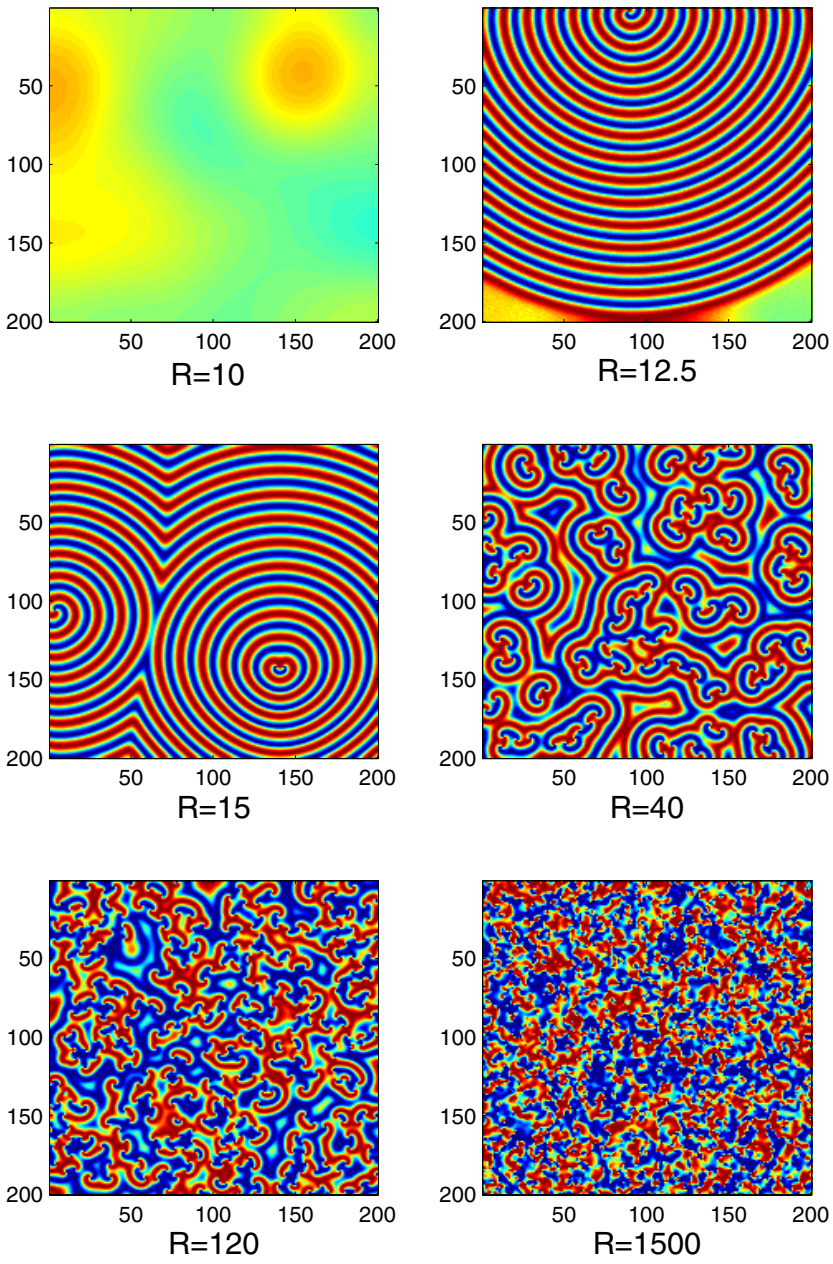


Fig. 10.26 Brusselator patterns for different coupling strengths.

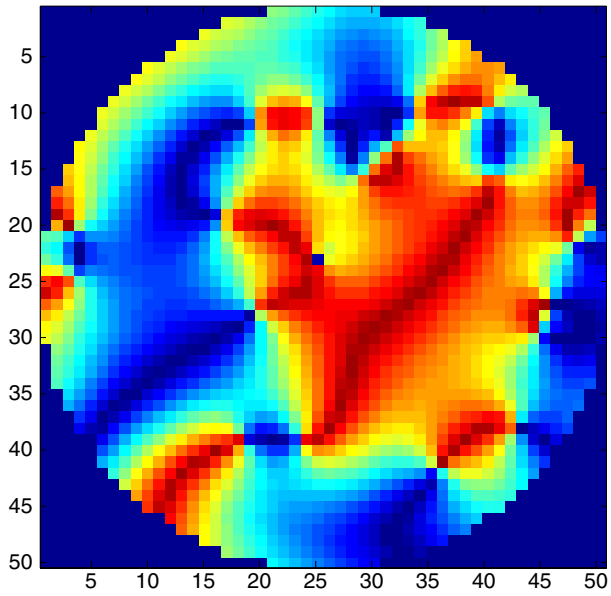


Fig. 10.27 Pattern resulting from Kuramoto's model.

Simulation of such pattern formation phenomena is important in understanding biological processes. Using PPV phase macromodels offers significant speedups: for the above simulations, speedups of about $3,000\times$ result, compared to simulation of the full oscillator ODEs. If individual oscillator sizes are larger, or the network is larger, speedups are larger.

Importantly, Kuramoto's sinusoidal model is unable to predict pattern formation correctly, underscoring the importance of accurate elemental models. Spiral waves or target patterns do not arise when Kuramoto's method is used, even for many different coupling strengths; patterns like the one in Figure 10.27 result instead.

10.4.5 Tunnelling Phase Logic-Based Nanoelectronic Cellular Nonlinear Network

Figure 10.28 depicts a basic tunneling phase logic (TPL) unit and its steady-state oscillation waveform. A TPL unit consists of an ultra-small SET junction with capacitance C_j, a DC bias V_{DC} and a pump

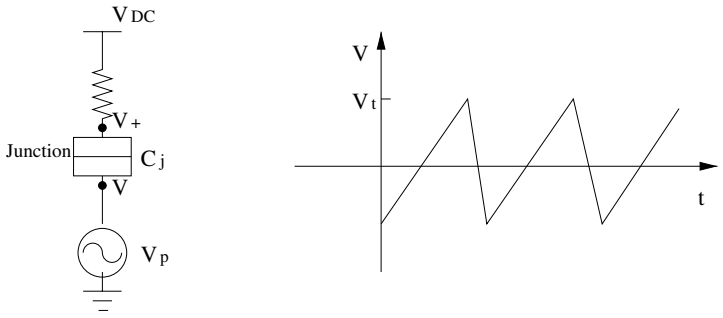


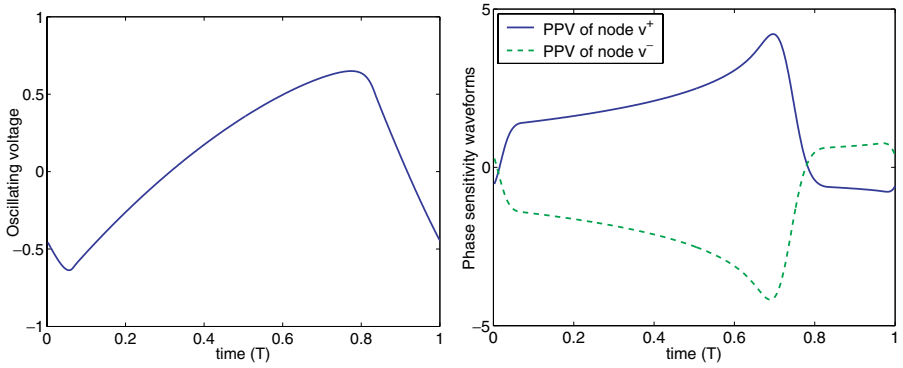
Fig. 10.28 TPL unit and its voltage–charge characteristic.

voltage V_p. The SET junction has the property that when its voltage increases to a threshold V_T, single-electron tunnelling occurs and the capacitor C_j is discharged. With the DC bias V_{DC} providing a bias current, the SET junction behaves as shown in Figure 10.28(b). The AC pump provides a sinusoidal voltage with amplitude V_p, which runs two times faster than the SET frequency. Therefore, if the SET is superharmonically locked by the pump voltage, it has two steady states separated by a phase difference of π. These two phases can be used to implement logic levels 0 and 1.

The characteristic in Figure 10.28(b) is a sawtooth idealization, best suited for analytical purposes. A more realistic model is to use a smooth characteristic for the SET. The steady-state oscillator solution of the smoothed TPL, and its PPV, is shown in Figure 10.29.

The PPV nonlinear phase macromodel (Equation (10.9)) is able to predict bistability in TPL. Four coupled TPL units, with different initial phases at $t = 0$, are simulated. After about 20 cycles, the phases of units 1 and 4 converge to one phase (about $\frac{3\pi}{2}$), as shown in Figure 10.30; the phases of the other units converge to a different phase (about $\frac{\pi}{2}$). It is apparent that PPV macromodels can replicate bistability in TPL systems.

PPV nonlinear phase macromodels are also able to replicate the dynamics of more complex networks of TPL oscillators, such as the TPL-based cellular nonlinear network (CNN) implementation of [123]. We use the nearest-neighbor coupling topology described in [123] and



(a) Smoothed TPL: steady state oscillation.

(b) Smoothed TPL PPV.

Fig. 10.29 Smoothed TPL: steady state and PPV.

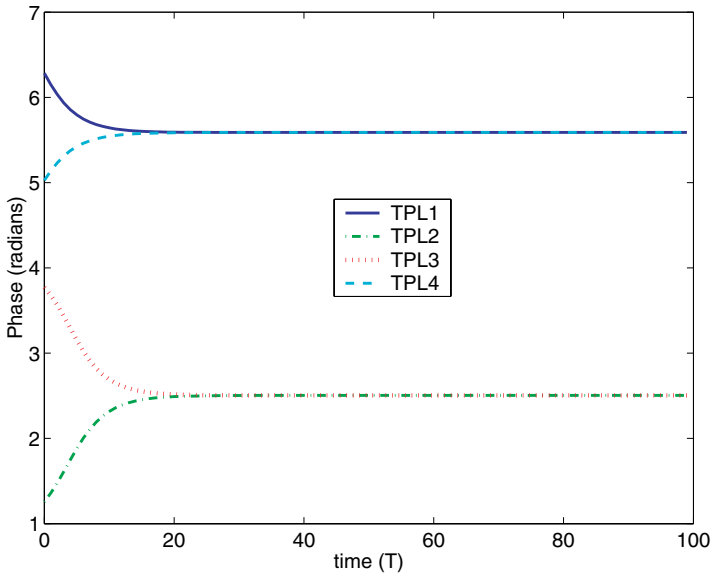


Fig. 10.30 TPL bistability.

define a "logic output" from each TPL unit to be

$$f_{i,j}(\phi_{i,j}) = \begin{cases} 1, & \pi < \phi_{i,j} < 2\pi \\ -1, & 0 < \phi_{i,j} < \pi, \end{cases} \tag{10.46}$$

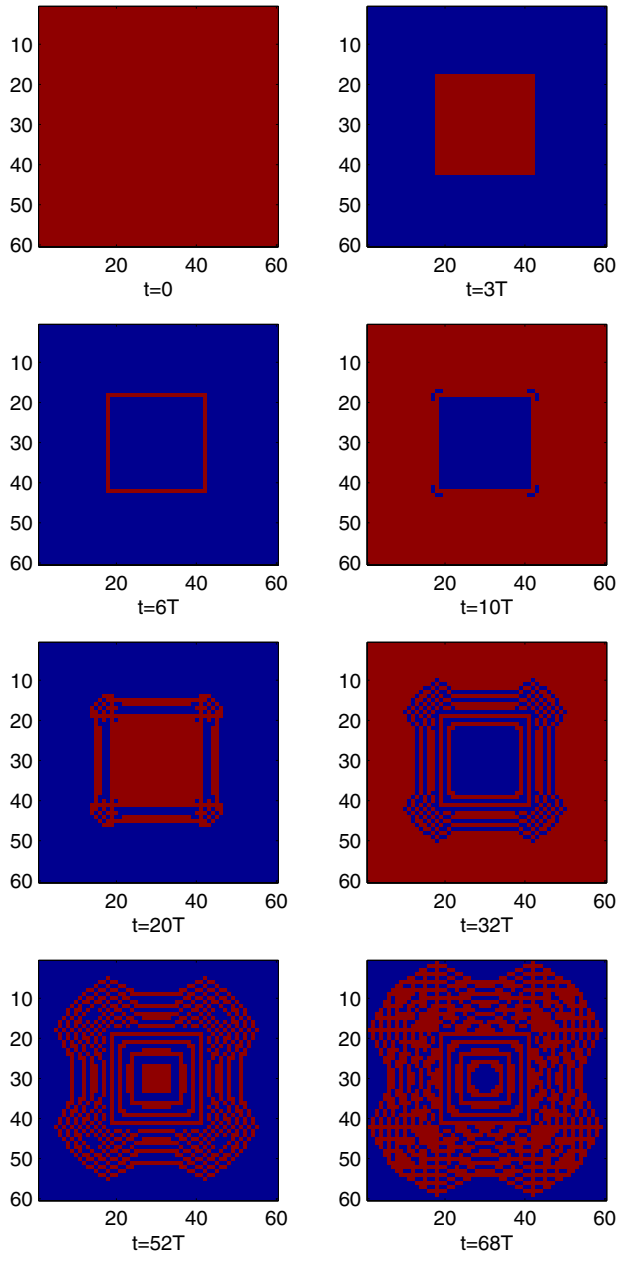
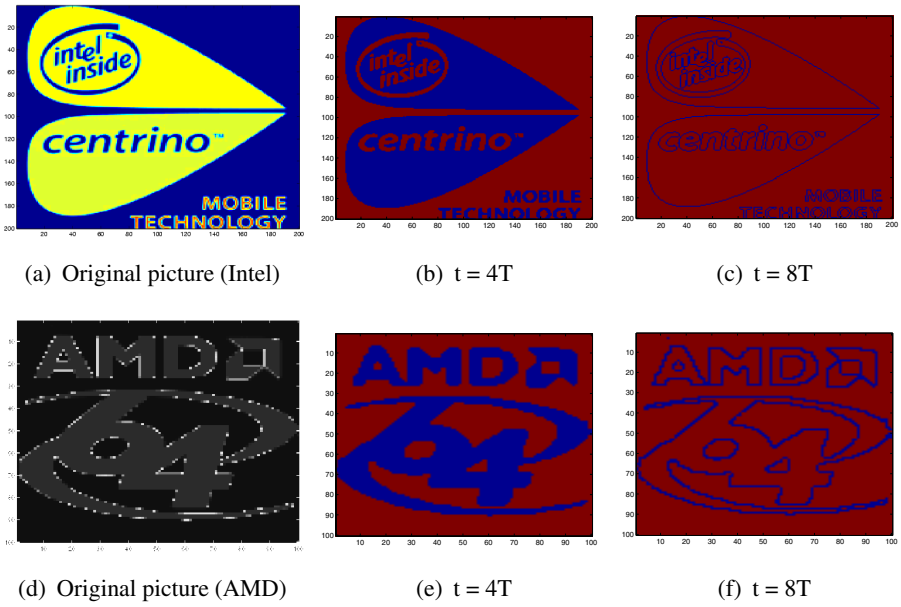


Fig. 10.31 TPL-CNN evolution, square pattern input.



(a) Original picture (Intel) (b) t = 4T (c) t = 8T

(d) Original picture (AMD) (e) t = 4T (f) t = 8T

Fig. 10.32 Edge detection capability of TPL-CNNs.

where $f_{i,j}$ is the output of the TPL cell on row i and column j, and $\phi_{i,j}$ is the cell's phase.

A simple square input pattern is applied to a 60×60 TPL-CNN and pattern formation is simulated. Figure 10.31 depicts the evolution of the input pattern over time as the TPL CNN "processes" it. At $t = 0$, all the TPLs have the same phase; no tunneling events have yet occurred. When $t = 3T$, tunneling occurs, and parts of the network move to two different phases due to the two-valued DC input. Interestingly, by $t = 6T$, the TPL CNN has performed edge detection, a key image processing operation. Further, along its evolution process, from $t = 10T$ to $t = 68T$, a number of complex geometrical patterns result. The same image processing feature is also apparent for other images, as shown in Figure 10.32.

References

[1] R. Adler, "A study of locking phenomena in oscillators," *Proceedings of the I.R.E. and Waves and Electrons*, vol. 34, pp. 351–357, June 1946.

[2] H.-T. Ahn and D. J. Allstot, "A low-jitter 1.9-v cmos pll for ultrasparc microprocessor applications," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 3, pp. 450–454, March 2000.

[3] T. J. Aprille and T. N. Trick, "Steady-state analysis of nonlinear circuits with periodicinputs," *Proceedings of the IEEE*, vol. 60, no. 1, pp. 108–114, January 1972.

[4] Z. Bai, R. Freund, and P. Feldmann, "How to make theoretically passive reduced-order models passive in practice," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 207–210, May 1998.

[5] H. G. Brachtendorf and R. Laur, "Transient simulation of oscillators," Technical Report ITD-98-34096K, Bell Laboratories, 1998.

[6] E. Chiprout and M. S. Nakhla, *Asymptotic Waveform Evaluation*. Norwell, MA: Kluwer, 1994.

[7] L. O. Chua and P.-M. Lin, *Computer-Aided Analysis of Electronic Circuits : Algorithms and Computational Techniques*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[8] L. O. Chua and L. Yang, "Cellular neural networks: Applications," *IEEE Transactions on Circuit and Systems*, pp. 1273–1290, 1988.

[9] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Transactions on Circuit and Systems*, pp. 1257–1272, 1988.

[10] W. Daems, G. Gielen, and W. Sansen, "A fitting approach to generate symbolic expressions for linear and nonlinear analog circuit performance

characteristics," in *Proceedings of the IEEE DATE Conference*, pp. 268–273, March 2002.

[11] G. Dahlquist and A. Björck, *Numerical Methods*. Dover, 2003.

[12] A. Demir, "Phase noise in oscillators: Daes and colored noise sources," in *IEEE/ACM International Conference on Computer-Aided Design*, November 1998.

[13] A. Demir, E. Liu, A. L. Sangiovanni-Vincentelli, and I. Vassiliou, "Behavioral simulation techniques for phase/delay-locked systems," in *Proceedings of the Custom Integrated Circuits Conference 1994*, pp. 453–456, May 1994.

[14] A. Demir, D. Long, and J. Roychowdhury, "Computing phase noise eigenfunctions directly from steady-state jacobian matrices," in *IEEE/ACM International Conference on Computer Aided Design*, pp. 283–288, November 2000.

[15] A. Demir, A. Mehrotra, and J. Roychowdhury, "Phase noise in oscillators: A unifying theory and numerical methods for characterization," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, pp. 655–674, May 2000.

[16] A. Demir, A. Mehrotra, and J. Roychowdhury, "Phase noise in oscillators: A unifying theory and numerical methods for characterization," *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 47, no. 5, pp. 655–674, May 2000.

[17] A. Demir and J. Roychowdhury, "A reliable and efficient procedure for oscillator PPV computation, with phase noise macromodelling applications," *IEEE Transactions on Computer-Aided Design*, pp. 188–197, February 2003.

[18] N. Dong and J. Roychowdhury, "Piecewise polynomial model order reduction," in *Proceedings of IEEE DAC*, pp. 484–489, June 2003.

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

[20] P. Feldmann, R. C. Melville, and D. Long, "Efficient frequency domain analysis of large nonlinear analog circuits," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1996.

[21] G. Fodor, *Laplace Transforms in Engineering*. Akademiai Kiado, Budapest, 1965.

[22] K. Francken, M. Vogels, E. Martens, and G. Gielen, "A behavioral simulation tool for continuous-time /spl Delta//spl Sigma/ modulators," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 229–233, November 2002.

[23] R. Freund, "Passive reduced-order models for interconnect simulation and their computation via Krylov-subspace algorithms," in *Proceedings of IEEE DAC*, pp. 195–200, June 1999.

[24] R. Freund and P. Feldmann, "Reduced-order modeling of large passive linear circuits by means of the SyPVL algorithm," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 280–287, November 1996.

[25] R. W. Freund, "Reduced-order modeling techniques based on Krylov subspaces and their use in circuit simulation," Technical Report 11273-980217-02TM, Bell Laboratories, 1998.

[26] R. W. Freund, "Reduced-order modeling techniques based on Krylov subspaces and their use in circuit simulation," *Applied and Computational Control, Signals, and Circuits*, vol. 1, pp. 435–498, 1999.

[27] R. W. Freund and P. Feldmann, "Efficient small-signal circuit analysis and sensitivity computations with the PVL algorithm," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 404–411, November 1995.

[28] K. Gallivan, E. Grimme, and P. Van Dooren, "Asymptotic waveform evaluation via a lanczos method," *Applied Mathematics Letters*, vol. 7, pp. 75–80, 1994.

[29] M. Gardner, *Phase-Lock Techniques*. Wiley, New York, 1979.

[30] W. A. Gardner, *Introduction to Random Processes with Applications to Signals & Systems*. McGraw-Hill, second ed., 1990.

[31] R. J. Gilmore and M. B. Steer, "Nonlinear circuit analysis using the method of harmonic balance — A review of the art. Part I. Introductory concepts," *International Journal on Microwave and Millimeter Wave CAE*, vol. 1, no. 1, 1991.

[32] E. J. Grimme, "Krylov projection methods for model reduction," PhD thesis, University of Illinois, EE Dept, Urbana-Champaign, 1997.

[33] R. Grimshaw, *Nonlinear Ordinary Differential Equations*. New York: Blackwell Scientific, 1990.

[34] A. Hajimiri and T. H. Lee, "A general theory of phase noise in electrical oscillators," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 2, February 1998.

[35] D. Hess, "Cycle slipping in a first-order phase-locked loop," *IEEE Transactions on Communications*, vol. 16, no. 2, pp. 255–260, April 1968.

[36] L. Hongzhou, A. Singhee, R. Rutenbar, and L. R. Carley, "Remembrance of circuits past: Macromodeling by data mining in large analog design spaces," in *Proceedings of IEEE DAC*, pp. 437–442, June 2002.

[37] X. Huang, C. S. Gathercole, and H. A. Mantooth, "Modeling nonlinear dynamics in analog circuits via root localization," *IEEE Transactions on Computer-Aided Design*, vol. 22, no. 7, pp. 895–907, July 2003.

[38] E. L. Ince, *Ordinary Differential Equations*. Forgotten Books, 2009.

[39] M. Kamon, F. Wang, and J. White, "Generating nearly optimally compact models from Krylov-subspacebased reduced-order models," *IEEE Transactions on Circuits and Systems II: Signal Processing*, pp. 239–248, April 2000.

[40] M. Kanehisa and S. Goto, "Kegg: Kyoto encyclopedia of genes and genomes," *Nucelie Acids Research*, vol. 28, no. 1, pp. 27–30, January 2000.

[41] J. Katzenelson and L. H. Seitelman, "An iterative method for solution of nonlinear resistive networks," Technical Report TM65-1375-3, AT&T Bell Laboratories.

[42] K. J. Kerns, I. L. Wemple, and A. T. Yang, "Stable and efficient reduction of substrate model networks using congruence transforms," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 207–214, November 1995.

[43] D. A. Kessler and H. Levine, "Pattern formation in dictyostelium via the dynamics of cooperative biological entities," *Physical Review E*, vol. 48, pp. 4801–4804, December 1993.

[44] R. A. Kiehl and T. Ohshima, "Bistable locking of single-electron tunneling elements for digital circuitry," *Applied Physics Letters*, vol. 67, no. 17, pp. 2494–2496, October 1995.

[45] A. J. Koch and H. Meinhardt, "Biological pattern formation: From basic mechanisms to complex structures," *Review of Modern Physics*, vol. 66, pp. 1481–1507, October 1994.

[46] K. Kundert, *Predicting the Phase Noise and Jitter of PLL-Based Frequency Synthesizers*. www.designers-guide.com, 2002.

[47] K. S. Kundert, J. K. White, and A. Sangiovanni-Vincentelli, *Steady-state Methods for Simulating Analog and Microwave Circuits*. Kluwer Academic Publishers, 1990.

[48] K. S. Kundert and A. Sangiovanni-Vincentelli, "Simulation of nonlinear circuits in the frequency domain," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-5, no. 4, pp. 521–535, October 1986.

[49] K. S. Kundert, G. B. Sorkin, and A. Sangiovanni-Vincentelli, "Applying harmonic balance to almost-periodic circuits," *IEEE Transactions on Microwave Theory and Techniques*, vol. MTT-36, no. 2, pp. 366–378, February 1988.

[50] Y. Kuramoto, *Chemical Oscillations, Waves, and Turbulence*. Springer, Berlin, 1984.

[51] Y. Kuramoto and I. Nishikawa, *Cooperative Dynamics in Complex Physical systems*. Springer, Berlin, 1989.

[52] X. Lai and J. Roychowdhury, "Capturing injection locking via nonlinear phase domain macromodels," *IEEE Transactions on Microwave Theory and Techniques*, vol. 52, no. 9, pp. 2251–2261, September 2004.

[53] X. Lai and J. Roychowdhury, "Fast, accurate prediction of PLL jitter induced by power grid noise," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 2004.

[54] X. Lai, Y. Wan, and J. Roychowdhury, "Fast PLL simulation using nonlinear VCO macromodels for accurate prediction of jitter and cycle-slipping due to loop non-idealities and supply noise," in *Proceedings of IEEE ASP-DAC*, January 2005. Best Paper.

[55] J. Lee, K. S. Kundert, and B. Razavi, "Modeling of jitter in bang-bang clock and data recovery circuits," in *Proceedings of the Custom Integrated Circuits Conference 2003*, pp. 711–714, September 2003.

[56] J.-R. Li and J. White, "Efficient model reduction of interconnect via approximate system gramians," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 380–383, November 1999.

[57] P. Li and L. Pileggi, "NORM: Compact model order reduction of weakly nonlinear systems," in *Proceedings of IEEE DAC*, pp. 472–477, June 2003.

[58] A. L. Lin, M. Bertram, K. Martinez, and H. L. Swinney, "Resonant phase patterns in a reaction-diffusion system," *Physical Review Letters*, vol. 84, pp. 4240–4243, May 2000.

[59] A. L. Lin, A. Hagberg, A. Ardelea, M. Bertram, H. L. Swinney, and E. Meron, "Four-phase patterns in forced oscillator systems," *Physical Review E*, vol. 62, pp. 3790–3798, 2000.

[60] L. Lin, L. Tee, and P. R. Gray, "A 1.4 ghz differential low-noise cmos frequency synthesizer using a wideband pll architecture," in *ISSCC 2000*, pp. 204–205, February 2000.

[61] D. Long, R. C. Melville, K. Ashby, and B. Horton, "Full chip harmonic balance," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 379–382, May 1997.

[62] S. A. Maas, *Nonlinear Microwave Circuits*. Artech House, Norwood, MA, 1988.

[63] P. K. Maini, K. J. Paintera, and H. N. P. Chaub, "Spatial pattern formation in chemical and biological systems," *Journal of Chemical Society Faraday Transactions*, pp. 3601–3610, 1997.

[64] A. Mehrotra, "Noise analysis of phase-locked loops," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 49, no. 9, pp. 1309–1316, September 2002.

[65] T. Mei and J. Roychowdhury, "Oscillator-AC: Restoring rigour to linearized small-signal analysis of oscillators," *IEEE Transactions on Computer-Aided Design*, vol. 26, no. 6, pp. 1054–1069, June 2007.

[66] R. C. Melville, P. Feldmann, and J. Roychowdhury, "Efficient multi-tone distortion analysis of analog integrated circuits," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 241–244, May 1995.

[67] R. Mickens, *Oscillations in Planar Dynamic Systems*. World Scientific, 1995.

[68] J. W. Mink, "Quasi-optical power combining of solid-state millimeter-wave sources," *IEEE Transactions on Microwave Theory and Techniques*, vol. 34, pp. 273–279, February 1986.

[69] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," PhD thesis, EECS department, University of California, Berkeley, Electronics Research Laboratory, Memorandum no. ERL-M520, 1975.

[70] L. W. Nagel and R. Rohrer, "Computer Analysis of Nonlinear Circuits, Excluding Radiation," *IEEE Journal of Solid-State Circuits*, vol. SC-6, pp. 166–182, August 1971.

[71] M. S. Nakhla and J. Vlach, "A piecewise harmonic balance technique for determination of periodic responses of nonlinear systems," *IEEE Transactions on Circuits and Systems*, vol. CAS-23, p. 85, 1976.

[72] O. Narayan and J. Roychowdhury, "Analysing oscillators using multitime PDEs," *IEEE Transactions on Circuits and Systems I: Fundamental Theory Applications*, vol. 50, no. 7, pp. 894–903, July 2003.

[73] A. Nayfeh and B. Balachandran, *Applied Nonlinear Dynamics*. Wiley, 1995.

[74] I. I. Novof, J. Austin, R. Kelkar, D. Strayer, and S. Wyatt, "Fully integrated cmos phase-locked loop with 15 to 240 mhz locking range and 50 ps jitter," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 11, pp. 1259–1266, November 1995.

[75] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: Passive reduced-order interconnect macromodelling algorithm," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 58–65, November 1997.

[76] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: Passive reduced-order interconnect macromodelling algorithm," *IEEE Transactions on Computer-Aided Design*, pp. 645–654, August 1998.

[77] T. Ohshima and R. A. Kiehl, "Operation of bistable phase-locked single-electron tunneling logic elements," *Journal of Applied Physics*, April 1996.

[78] T. Ohtsuki, T. Fujisawa, and S. Kumagai, "Existence theorems and a solution algorithm for piecewise-linear resistor networks," *SIAM Journal of Mathematical Analysis*, vol. 8, February 1977.

[79] A. Papoulis, *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, New York: McGraw-Hill Series in System Science, 1965.

[80] J. Phillips, "Model reduction of time-varying linear systems using approximate multipoint Krylov-subspace projectors," in *IEEE/ACM International Conference on Computer-Aided Design*, November 1998.

[81] J. Phillips, "Projection-based approaches for model reduction of weakly nonlinear, time-varying systems," *IEEE Transactions on Computer-Aided Design*, vol. 22, no. 2, pp. 171–187, February 2000.

[82] J. Phillips, "Projection frameworks for model reduction of weakly nonlinear systems," in *Proceedings of IEEE DAC*, June 2000.

[83] J. Phillips, L. Daniel, and L. M. Silveira, "Guaranteed passive balancing transformations for model order reduction," in *Proceedings of IEEE DAC*, pp. 52–57, June 2002.

[84] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *IEEE Transactions on Computer-Aided Design*, vol. 9, pp. 352–366, April 1990.

[85] T. Poston, *Catastrophe Theory And Its Applications*. Dover Publications, 1997.

[86] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes — The Art of Scientific Computing*. Cambridge University Press, 1989.

[87] I. Prigogine and R. Lefever, "Symmetry breaking instabilities in dissipative systems II," *Journal of Chemical Physics*, vol. 48, p. 1695, 1968.

[88] Y. Qicheng and C. Sechen, "A unified approach to the approximate symbolic analysis of large analog integrated circuits," *IEEE Transactions on Circuits and Systems I: Fundemental Theory Applications*, pp. 656–669, August 1996.

[89] T. L. Quarles, "Analysis of performance and convergence issues for circuit simulation," PhD thesis, EECS department, University of California, Berkeley, Electronics Research Laboratory, Memorandum no. UCB/ERL M89/42, April 1989.

[90] T. L. Quarles, *SPICE 3C.1 User's Guide*. University of California, Berkeley, EECS Industrial Liaison Program, University of California, Berkeley, California, 94720, April, 1989.

[91] S. A. Raghavan and H. K. Thapar, "On feed-forward and feedback timing recovery for digital magnetic recording systems," *IEEE Transactions on Magnetics*, vol. 27, no. 6, pp. 4810–4812, November 1991.

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

[93] V. Rizzoli and A. Neri, "State of the art and present trends in nonlinear microwave CAD techniques," *IEEE Transactions on Microwave Theory and Techniques*, vol. 36, no. 2, pp. 343–365, February 1988.

[94] M. Rösch, "Schnell simulation des stationären verhaltens nichtlinearer schaltungen," PhD thesis, Technischen Universität München, 1992.

[95] M. Rösch and K. J. Antreich, "Schnell stationäre simulation nichtlinearer schaltungen im frequenzbereich," *AEÜ*, vol. 46, no. 3, pp. 168–176, 1992.

[96] J. Roychowdhury, "MPDE methods for efficient analysis of wireless systems," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1998.

[97] J. Roychowdhury, "Reduced-order modelling of linear time-varying systems," in *IEEE/ACM International Conference on Computer-Aided Design*, November 1998.

[98] J. Roychowdhury, "Reduced-order modelling of time-varying systems," *IEEE Transactions on Circutis and System — II: Signal Processing*, vol. 46, no. 10, November 1999.

[99] J. Roychowdhury, "Analysing circuits with widely separated time scales using numerical PDE methods," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 48, no. 5, pp. 578–594, May 2001.

[100] W. Rugh, *Nonlinear System Theory — The Volterra-Wiener Approach*. Johns Hopkins University Press, 1981.

[101] D. B. Rutledge, Z. B. Popovic, R. M. Weikle II, M. Kim, K. A. Potter, R. C. Compton, and R. A. York, "Quasi-optical power-combining arrays," in *Microwave Symposium Digest, 1990, IEEE MTT-S International*, pp. 1201–1204, May 1990.

[102] Y. Saad, *Iterative Methods for Sparse Linear Systems*. PWS, Boston, 1996.

[103] M. Schetzen, *The Volterra and Wiener Theories of Nonlinear Systems*. John Wiley, 1980.

[104] R. L. Schiek and E. M. May, "Examining tissue differentiation stability through large scale, multi-cellular pathway modeling," in *Nanotech 2005*, May 2005.

[105] D. Schreurs, J. Wood, N. Tufillaro, D. Usikov, L. Barford, and D. E. Root, "The construction and evaluation of behavioral models for microwave devices based on time-domain large-signal measurements," in *Proceedings of IEEE IEDM*, pp. 819–822, December 2000.

[106] L. M. Silveira, M. Kamon, I. Elfadel, and J. White, "A coordinate-transformed Arnoldi algorithm for generating guaranteed stable reduced-order models of RLC circuits," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 288–294, November 1996.

[107] S. Skelboe, "Computation of the periodic steady-state response ofnonlinear networks by extrapolation methods," *IEEE Transactions on Circuits and Systems*, vol. CAS-27, no. 3, pp. 161–175, March 1980.

[108] J. L. Stensby, *Phase-Locked Loops: Theory and Applications*. New York: CRC Press, 1997.

[109] M. Takahashi, K. Ogawa, and K. S. Kundert, "VCO jitter simulation and its comparison with measurement," in *Proceedings of Design Automation Conference 1999*, pp. 85–88, June 1999.

[110] S. X.-D. Tan and C. J.-R. Shi, "Efficient DDD-based Term Generation Algorithm for Analog Circuit Behavioral Modeling," in *Proceedings of IEEE ASP-DAC*, pp. 789–794, January 2003.

[111] S. X.-D. Tan and C. J.-R. Shi, "Efficient DDD-based term generation algorithm for analog circuit behavioral modeling," in *Proceedings of IEEE DATE Conference*, pp. 1108–1009, March 2003.

[112] S. X.-D. Tan and C. J.-R. Shi, "Efficient very large scale integration power/ground network sizing based on equivalent circuit modeling," *IEEE Transactions on Computer-Aided Design*, vol. 22, no. 3, pp. 277–284, March 2003.

[113] T. Taniuti and N. Yajima, "Perturbation method for a nonlinear wave modulation," *Journal of Mathematical Physics*, vol. 10, pp. 1369–1372, August 1969.

[114] R. Telichevesky, K. Kundert, and J. White, "Efficient steady-state analysis based on matrix-free Krylov subspace methods," in *Proceedings of IEEE DAC*, pp. 480–484, 1995.

[115] A. M. Turing, "Philosophical transactions of the Royal Society of London. Series B," *Biological Sciences*, vol. 237, no. 641, pp. 37–72, August 14, 1952.

[116] P. Vanassche, G. Gielen, and W. Sansen, "Constructing symbolic models for the input/output behavior of periodically time-varying systems using harmonic transfer matrices," in *Proceedings of IEEE DATE Conference*, pp. 279–284, March 2002.

[117] P. Vanassche, G. G. E. Gielen, and W. Sansen, "Behavioral modeling of coupled harmonic oscillators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 8, pp. 1017–1026, August 2003.

[118] W. Weeks, A. Jimenez, G. Mahoney, D. Mehta, H. Qassemzadeh, and T. Scott, "Algorithms for ASTAP — A network-analysis program," *IEEE Transactions on Circuits and Systems*, vol. 20, no. 6, pp. 628–634, November 1973.

[119] N. Wiener, *Nonlinear Problems in Random Theory*. Cambridge, MA: MIT press, 1958.

[120] N. Wiener, *Cybernetics*. Cambridge, MA: MIT press, 1961.

[121] A. Winfree, "Biological rhythms and the behavior of populations of coupled oscillators," *Theoretical Biology*, vol. 16, pp. 15–42, 1967.

[122] J. Wood and D. E. Root, "The behavioral modeling of microwave/RF ICs using non-linear time series analysis," in *IEEE MTT-S Digest*, pp. 791–794, June 2003.

[123] T. Yang, R. A. Kiehl, and L. O. Chua, "Tunneling phase logic cellular nonlinear networks," *International Journal of Bifurcation and chaos*, vol. 11, pp. 2895–2911, 2001.

[124] L. A. Zadeh and C. A. Desoer, *Linear System Theory: The State-Space Approach*. McGraw-Hill, New York: McGraw-Hill Series in System Science, 1963.