

**THE DESIGNER'S GUIDE TO
SPICE AND SPECTRE®**

by

Kenneth S. Kundert

Cadence Design Systems



KLUWER ACADEMIC PUBLISHERS
Boston / Dordrecht / London

Distributors for North America:

Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA

Distributors for all other countries:

Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Consulting Editor: Jonathan Allen, Massachusetts Institute of Technology

Library of Congress Cataloging-in-Publication Data

A C. I. P. Catalogue record for this book is available
from the Library of Congress.

Spectre® is a registered trade mark of Cadence Design Systems, Inc.

Copyright © 1995 by Kluwer Academic Publishers

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher, Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061

Printed on acid-free paper.

Printed in the United States of America

Contents

Foreword**ix****Preface****xi****1 Circuit Simulation****1**

1.1 Historical Perspective

1

1.2 Algorithmic Perspective

3

1.3 Circuit Simulation

6

1.3.1 Equation Formulation

9

1.3.2 Differential Equations

10

1.3.3 Analyses

12

1.4 Modeling

13

1.5 Summary

14

2 DC Analysis**15**

2.1 Introduction

15

2.2 DC Analysis Theory

16

2.2.1 Solving Nonlinear Equations

18

2.2.2 Convergence Criteria

19

2.2.3 Convergence

23

2.3 DC Analysis Practice

35

2.3.1 Remedies for Convergence Problems

35

2.3.2 DC Analysis Accuracy

39

2.3.3	Remedies for Accuracy Problems	41
2.4	Applications of DC Analysis	42
2.4.1	Circuits with Multiple Operating Points	42
2.4.2	Large Circuits	45
2.4.3	Restarting from a Previous Solution	47
2.5	Summary	49
3	AC Analyses	51
3.1	Introduction	51
3.2	AC Analyses Theory	53
3.3	AC Analyses Practice	54
3.3.1	AC Analysis	54
3.3.2	XF Analysis	55
3.3.3	SP Analysis	56
3.3.4	Noise Analysis	57
3.4	Applications of the AC Analyses	67
3.4.1	Characterizing Feedback Amplifiers	67
3.4.2	Transfer Function Versus Bias	106
3.4.3	Capacitance Versus Bias	108
3.4.4	Non-Quiescent Operating Points	113
3.4.5	Differential Amplifiers	113
3.5	Summary	119
4	Transient Analysis	129
4.1	Introduction	129
4.2	Transient Analysis Theory	130
4.2.1	Integration Methods	131
4.2.2	Characteristics of the Integration Methods	133
4.2.3	Transient Analysis Accuracy	157
4.2.4	Truncation Error	160
4.2.5	Charge Conservation	167
4.3	Transient Analysis Practice	177
4.3.1	SPICE's Time-Step Control	177

4.3.2	Spectre's Time-Step Control	180
4.3.3	Break Points	189
4.3.4	Bypass	194
4.3.5	Initial Conditions	195
4.3.6	Remedies for Accuracy Problems	199
4.3.7	Convergence in Transient Analysis	201
4.3.8	Remedies for Convergence Problems	206
4.4	Applications of Transient Analysis	207
4.4.1	Computing the DC Operating Point	207
4.4.2	Oscillators	209
4.4.3	Unstable Circuits	219
4.4.4	Charge-Storage Circuits	224
4.4.5	Sinusoidal Circuits	226
4.4.6	Macromodels	227
4.4.7	Distribution Networks	234
4.4.8	Large Circuits	234
4.4.9	Accurate Current Measurements	235
4.4.10	Strobing	238
4.5	Summary	246
5	Fourier Analysis	251
5.1	Introduction	251
5.1.1	Distortion Metrics	251
5.2	Fourier Analysis Theory	257
5.2.1	The Fourier Series	257
5.2.2	The Discrete Fourier Transform	265
5.3	Fourier Analysis Practice	267
5.3.1	Errors Mechanisms	267
5.3.2	SPICE's Fourier Analysis	278
5.3.3	Spectre's Fourier Analysis	282
5.3.4	External Fourier Analysis	289
5.4	Applications of Fourier Analysis	299
5.4.1	THD of Low Distortion Amplifier	299

5.4.2	Resolution of a $\Sigma\Delta$ Modulator	303
5.4.3	Distortion of a Pulse-Width Modulator	308
5.4.4	Oscillators	310
5.4.5	Large-signal Transfer Functions	312
5.4.6	Clocked Analog Circuits	313
5.4.7	IMD of Narrow-Band Circuits	316
5.4.8	Distortion of a Mixer	328
5.5	Summary	332
A Simulator Options 335		
A.1	Introduction	335
A.2	SPICE Options	335
A.2.1	Global Options	335
A.2.2	DC Analysis Options	338
A.2.3	Transient Analysis Options	339
A.3	Spectre Options	342
A.3.1	Global Options	342
A.3.2	DC Analysis Options	347
A.3.3	Transient Analysis Options	350
B Spectre Netlist Language 357		
B.1	Introduction	357
B.2	The Language	357
Bibliography 367		
Index 371		

Foreword

Engineering productivity in integrated circuit product design and development today is limited largely by the effectiveness of the CAD tools used. For those domains of product design that are highly dependent on transistor-level circuit design and optimization, such as high-speed logic and memory, mixed-signal analog-digital interfaces, RF functions, power integrated circuits, and so forth, circuit simulation is perhaps the single most important tool.

As the complexity and performance of integrated electronic systems has increased with scaling of technology feature size, the capabilities and sophistication of the underlying circuit simulation tools have correspondingly increased. The absolute size of circuits requiring transistor-level simulation has increased dramatically, creating not only problems of computing power resources but also problems of task organization, complexity management, output representation, initial condition setup, and so forth. Also, as circuits of more complexity and mixed types of functionality are attacked with simulation, the spread between time constants or event time scales within the circuit has tended to become wider, requiring new strategies in simulators to deal with large time constant spreads.

As a result the typical circuit simulator today is a very powerful but very complex tool that is absolutely essential to the circuit designer in carrying out his task. More often than not, circuit designers do not fully understand the capabilities and limitations of this critical tool that is so important to their productivity. Often the time pressure of product development and the difficulty of obtaining accurate information on simulator capabilities and underlying functionality makes

it difficult for the design engineer to become a fully-informed user of the tool. The result can be longer design cycles and unnecessary product prototyping cycles.

This book provides an excellent new resource to the circuit designer on the internal operation and the application of circuit simulators. All facets of circuit simulation are discussed from the perspective of a sophisticated user who wants to understand what the simulator is actually doing but does not want to get involved in the details of the underlying code. The book should be required reading for design engineers making use of circuit simulators. Fundamental simulator problems such as convergence, charge conservation, and the like are covered extensively. Advanced applications such as techniques for efficiently working with large circuits and measurement techniques for loop gain or intermodulation distortion are also discussed at the end of each chapter.

It is clear that both the integrated circuits that are designed and manufactured as well as the CAD tools that provide the essential simulation capability to the designer will continue to increase in complexity and sophistication. This book makes a major contribution to increasing design engineering productivity by allowing more enlightened and efficient utilization of simulation tools.

Paul R. Gray
March 16, 1995
Berkeley, California

Preface

Circuit simulation is an important tool used when designing analog electronic circuits. Circuit simulators allow new designs to be evaluated quickly and at considerably less expense than the only other alternative, fabrication. Simulation is very heavily used, particularly in the design of analog integrated circuits, with almost all circuits being simulated before being fabricated. It is also rapidly becoming more popular for the design of board level analog circuits. However, even though circuit simulators have been available for over 20 years, getting a circuit simulator to converge and give an accurate answer is still considered an art. There is considerable folklore on how to use simulators successfully; however, much of it is based on the idiosyncrasies of particular simulators or tricks that are based largely on luck. For example, one large electronics company found that convergence difficulties sometimes disappeared when the input file was reorganized and wrote a program that designers could use to randomly shuffle the simulator input file.

The benefits and risks inherent in the use of a circuit simulator can be compared with that of an automobile. Cars are very useful, and some might say essential, and despite continuing improvements, they can still fail to get you where you are going, either because of collisions, break downs, or you simply get lost. Though, through knowledge of where you are going, how to safely operate the car, and how to properly maintain it, you can greatly increase your chances. As with automobiles, simulators are not completely reliable. Despite decades of improvements, simulators still occasionally fail to either converge or compute an accurate result.

It is impossible to write a circuit simulator that always computes accurate answers. Even attempting to write a simulator that produced accurate results in virtually all cases would result in the simulator being so conservative that it would be much too slow on circuits that do not have accuracy problems. Thus, circuit simulators are designed to work well in most cases, and it is up to the designer to be knowledgeable enough to recognize any problems that occur and correct them. Generally, circuit designers understand their circuits well enough to recognize inaccurate results, however they often do not know the best way to overcome problems. They usually rely on a trial-and-error approach to solve a problem, and often there are as many errors as trials.

How This Book is Unique

The motivation for writing this book came when I was asked to give a presentation to designers on these topics. I went to the bookstore to get material from the available simulation books. What I found was three types of books. Most books were targeted towards neophytes and were little more than users guides for various commercial simulators. They talked about simple things such as how to create a netlist and how to plot waveforms. The second type of book gave detailed descriptions of the semiconductor models provided in most simulators. The last type of book delved very deeply into simulation algorithms and was targeted towards people who write circuit simulators. None of the books addressed problems that an average designer might have with the simulator itself, which is my intention with this book.

This book is not an introduction to circuit simulators. I assume that you are already adept at the mechanics of operating a simulator and are interested in understanding how it works and increasing the skill and sophistication with which you use it. For an introduction to circuit simulators and SPICE, see Vladimirescu's *The SPICE Book* [vladimirescu94].

Why You Should Read This Book

This book is designed to take you from being a reactive user to being a proactive user. A reactive user is one that runs the simulator and hopes that nothing goes wrong. If something does go wrong, various remedies are pulled from a bag of tricks one at a time with little understanding with the hope that one solves the problem. If not, either the circuit is redesigned to avoid the problem or the simulator is not used. The proactive user anticipates problems. When one occurs, the user knows why it occurred and just what to do to resolve it. The reactive user is controlled by the simulator, whereas the proactive user controls the simulator.

I expect that by reading this book you will gain the following:

1. A basic understanding of how circuits simulators compute their results.
2. An understanding of what kind of errors occur with circuit simulators and how to recognize these errors.
3. General guidelines on how to improve the accuracy of the solution.
4. General guidelines on how to encourage the simulator to converge.
5. An understanding of what kind of errors are expected when simulating particular classes of circuits.
6. Know the meaning of the simulator's key convergence and error control parameters, such as `trtol`, `chgtol`, `gmin`, and etc.
7. Can relate simulator error messages to what actually when wrong and what should be done to fix it.

In addition, the book describes many nonobvious applications of circuit simulators.

What Is In This Book

This book presents one person's suggestions on getting a simulator to behave, based on 20 years of experience, both as a user and as a developer of circuit simulators. This book is intended to be a practical guide for circuit designers that routinely use circuit simulators. I describe problems that commonly occur and give suggestions on how to solve these problems. These issues are examined in the context of the circuit simulators SPICE2 [spice2] and SPICE3 [spice3], because they are so pervasive, and Spectre¹, a new simulator designed to address many of these problems. In fact, Spectre was modified several times in order to address issues that were uncovered while writing this book. Even though I focus on these three simulators, this book should be useful to the user of any circuit simulator.

In this book, I introduce the algorithms used by a circuit simulator with an emphasis on how convergence problems occur and how the algorithms influence the error in the results. Techniques for resolving convergence difficulties and controlling error are presented. In addition, I use example circuits, such as oscillators or switched-capacitor circuits, to illustrate the simulation problems that are inherent with certain types of circuits or to show how to make challenging measurements. The presentation is geared toward describing how designers can recognize and control common simulation problems.

Chapter 1 introduces circuit simulation, both from a historical and a computational perspective. Chapter 2 presents DC analysis and discusses the issue of convergence in depth. Convergence is important for both DC and transient analysis. Chapter 3 covers the small signals analyses, such as AC and noise analysis. Chapter 4 discusses transient analysis, with considerable attention being paid to the issue of accuracy. Chapter 5 talks about Fourier analysis. Finally in Appendix A the various simulator options are described.

The structure of each of the chapters on simulator analyses is similar. They start with an introduction, then delve into the theory or mathematical underpinnings of the analysis. Enough theory is

¹SpectreTM is a registered trademark of Cadence Design Systems of San Jose, California.

given for you to understand the basic operation of the analysis as well as its characteristics. Examples given in this section illustrate issues that result directly from the underlying mathematics. After the theory, the practical details or heuristics of the implementation are presented. Again, the focus is on presenting those details that directly result in accuracy or convergence issues. Examples given in this section illustrate the issues that result from the heuristics present in simulators. Finally, the chapter finishes up with a presentation of some of the various ways in which the analysis can be used. The intention is to present real-world nontrivial applications that illustrate how to make important measurements or how to resolve some thorny issues.

A Word About Netlists Forgive me for using Spectre netlists rather than SPICE netlists for the examples given in this book. Spectre's parameterized subcircuits makes these examples much more powerful and more self-explanatory. The Spectre language is similar to the SPICE language, so you should be able to translate these netlists without too much difficulty, as long as your simulator provides the needed components. Brief documentation for Spectre's netlist language is given in Appendix B.

Acknowledgements

I would like to deeply thank Jacob White and Alberto Sangiovanni-Vincentelli, who taught me the fine art of circuit simulation. I would also like to acknowledge the contributions of David Root of Hewlett-Packard and Mike Tu, Don Webber, and James Spoto of Cadence Design Systems. Dave helped me better understand the charge-conservation issue. Mike pointed out some of the subtle issues involved in loop-gain calculations. Don (along with Jacob) helped me work through and understand many of the issues presented in this book. And finally, James provided me with the environment and the freedom that allowed me research and write this book.

Thanks to Karin Freuler of Cadence and Mark Williams of Harris Semiconductor for volunteering to edit large pieces of this book. I

would also like to acknowledge the many circuit designers that contributed to this book, either intentionally or innocently.

Finally, special thanks go to my family, Mary, Kale, and Kara, for being so patient and supportive during the time it took to write this book.

One of the pleasures in writing a book is the opportunity afforded the author to improve both the breadth and the depth of his own understanding of a subject. I hope that in trying to convey the understanding of what I have learned while writing this book, that I am also able to share with you some of that pleasure.

Ken Kundert
March 16, 1995
Los Altos, California

THE DESIGNER'S GUIDE TO SPICE AND SPECTRE®

Chapter 1

Circuit Simulation

1.1 Historical Perspective

Circuit simulators, as we know them today, first began to appear in the late 1960's and early 70's. However, it was the explosive growth of the integrated circuit market in the 1970's that precipitated the rise of importance of circuit simulation. With integrated circuits, prototypes were expensive to build and difficult to troubleshoot. Circuit simulators were necessary to evaluate designs before they were fabricated. As designs became larger and more complicated, the need to use circuit simulators increased.

Two groups contributed significantly to the development of the modern circuit simulator. The ASTRAP group at IBM developed many of the numerical method's used. And the SPICE group at the University of California at Berkeley developed and propagated the de facto standard simulator.

The simulation effort at Berkeley started as a class project of Prof. Ron Rohrer. That modest beginning resulted in a flurry of simulation programs being developed and culminated in the release of SPICE in 1972 and then SPICE2 in 1975. SPICE was written by Larry Nagel, then under the guidance of Prof. Don Pederson. SPICE became very important for three reasons. First, SPICE was designed to be used to simulate integrated circuits. Unlike the simulators that preceded it, SPICE had all the models one needed to simulate integrated circuits built into it. As such, it was easier to use than earlier simulators.

Second, the source code for SPICE was made available to anyone who wanted it at a nominal cost. And third, Berkeley graduates took SPICE with them as they went to work at electronics companies throughout the country.

In the late 70's and early 80's, most versions of SPICE were proprietary and only used in-house by the integrated circuit manufacturers. At the time, it seemed as if every large electronics company in the country had developed their own version of SPICE. This occurred because circuit simulation was vital to IC manufactures and at least initially, there were few commercial simulators available that were suitable for IC design. Also, the IC manufacturers viewed having a good version of SPICE as a strategic advantage that allowed them to get designs to market quickly and reliably.

This situation began to change in the late 80's and early 90's as the commercial simulators began to surpass the internally developed simulators in terms of capabilities and performance. When this happened, the strategic value of an internally developed simulator disappeared. Commercial simulators starting replacing internal simulators, starting with the smaller companies and working up. Today, only the largest companies are still developing their own proprietary simulators.

In the late 80's, Berkeley upgraded SPICE by releasing SPICE3, which had a new architecture that made it considerably easier to add new component models and was written in C. While SPICE3 was architecturally a big step forward from SPICE2, algorithmically it was largely the same.

At the same time, Berkeley also released a new type of circuit simulator called Spectre. Spectre used harmonic balance to directly compute the steady-state solution of nonlinear circuits in the frequency domain. It was targeted for use on microwave circuits. Spectre was picked up by Hewlett-Packard, where it became known as their Microwave Nonlinear Simulator, or MNS, and by Cadence, where the harmonic balance algorithms were replaced by transient analysis algorithms.

Cadence took a slightly different approach with Spectre than is typ-

1.2. Algorithmic Perspective

ical. Rather than trying to increase the speed of the simulator by loosening the tolerances or employing faster, but less reliable, timing analysis algorithms, Cadence instead took a conservative approach. It started with the standard SPICE algorithms, discarded those that reduced reliability, such as bypass, and implemented each one from scratch with the goal of improving speed, as well as accuracy and reliability. By exploiting the 15 years of evolutionary improvements in simulation algorithms that had occurred since SPICE was written, Cadence was able to make Spectre 3-5 times faster than traditional versions of SPICE, while improving its accuracy and reliability.

It was during the process of developing Spectre than many of the issues that are discussed in this book were first encountered and explored. It is because Spectre was designed to address these issues that it plays a central part in this book. However, the book does not focus exclusively on Spectre. It discusses issues applicable to all simulators and so is useful for anyone that uses a circuit simulator.

A more comprehensive history of circuit simulation in general, and SPICE in particular, is available from Pederson [pederson84] and in Vladimirescu's *The SPICE Book* [vladimirescu94].

1.2 Algorithmic Perspective

The algorithms used in SPICE now define the traditional approach to circuit simulation. This approach is referred to as the *direct method* for simulating a circuit. With direct methods, the nonlinear ordinary differential equations that describe the circuit are first formulated and then converted to a system of difference equations by a multi-step integration method such as the trapezoidal rule. The nonlinear difference equations are solved using the Newton-Raphson algorithm, which generates a sequence of linear equations that are solved using sparse Gaussian elimination. Direct methods have proven to be the most reliable and general methods available.

In the late 70's and early 80's, attempts were made by several groups to develop alternate approaches that would provide better performance on the large digital circuits of the day. Two basic methods were explored, explicit integration methods and relaxation methods.

Using explicit integration methods, such as forward Euler, eliminates the need to actually solve the large system of differential equations that describe the circuit. Instead, the solution at a particular time step is extrapolated from the previous time point. It is assumed that there are no floating capacitors and there is at least one capacitor connecting every node in the circuit to ground. The extrapolation is performed by evaluating the circuit equations (not solving them) to determine the current into the grounded capacitors. The slope of the voltage waveforms are then computed directly from

$$\dot{v}(t) = C^{-1}i(t). \quad (1.1)$$

Explicit integration methods are very fast, especially if the grounded capacitors are linear and so are easily invertible. However, they have not gained wide acceptance because they are unstable and so generate results that blow-up if the circuit contains time constants that are shorter than the time step being used. This is a serious problem that makes explicit methods unstable on most circuits. Digital MOS circuits tend to have time constants that are all about the same size, so explicit methods can sometimes be used with great success.

Relaxation methods exploit latency in the circuit by breaking it into smaller pieces and solving each piece independently. If the signals in one or more pieces are latent, then it is not necessary to solve for them. The waveform relaxation methods take this idea one step further. Circuits are still partitioned into subcircuits, but the subcircuits are solved independently over an interval of time rather than for a single time point. This allows the simulator to exploit multi-rate behavior as well as latency. Multi-rate behavior is where the signals in subcircuits are changing, but where the signals in one are changing much more slowly than in another. In this case, the simulator is free to choose larger time steps in the subcircuit whose signals are changing slowly.

The drawback to relaxation methods is that because the entire circuit is not evaluated at once, it is sometimes necessary to make assumptions about signals before they have been computed, and if the assumptions turn out to be incorrect, the subcircuits that depend on those signals have to be reevaluated. Consider the circuit shown in Figure 1.1 and assume N_1 is evaluated before N_2 . Further assume

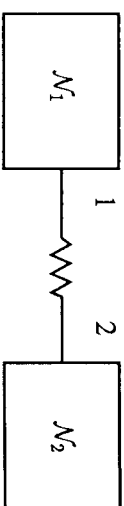


Figure 1.1: Example circuit used to illustrate relaxation methods.

that the signal on node 1 is found when solving N_1 and the signal on node 2 is found when solving N_2 . In order to solve N_1 , we must predict the values of the signal at node 2. If after solving N_2 it is decided that the signal on node 2 is significantly different from the prediction, then N_1 must be solved again. If the signal on node 1 changes, then N_2 needs to be solved again. This repeated solution of the blocks reduces the efficiency of relaxation methods. For relaxation methods to be attractive, this iteration must converge quickly, which happens if the back coupling from N_2 to N_1 is very small (if, in this case, the resistor is large). Thus, the performance of relaxation methods is critically dependent on the partitioning of the circuit into subcircuits and the ordering of the subcircuit evaluation. Unfortunately, automatic partitioning of circuit is only possible on the relatively simple circuit topologies found in digital MOS circuits. If relaxation methods are applied to analog or bipolar circuits, they become extremely inefficient and can even generate incorrect answers. Even more disturbing is that unlike with the explicit integration methods, the incorrect results generated by relaxation methods look plausible. Thus, users cannot easily tell when the simulator is generating correct results and when it is not, which severely undermines the trust that every designer must have in his or her simulator.

To further reduce the simulation time, some simulators provide one-step relaxation (OSR) methods. One-step relaxation methods do not reevaluate subcircuits once they have been evaluated, even if assumptions about signal values later prove to be incorrect. This clearly results in the simulator running faster, but of course, also makes it much more likely that it will compute the wrong answer.

Consider applying OSR to the circuit in Figure 1.2 on the facing page and assume that V_{in} steps to 1 V at $t = 0$. In the first iteration, V_2 is assumed to be zero and V_1 is computed to be 9.9 mV. Then V_1 is fixed and V_2 is computed to be 9.8 mV. At this point, the results are far from the correct value of $V_1 = 502$ mV and $V_2 = 498$ mV, but OSR assumes the values are correct, saves these incorrect results, increments time, and continues. On the second iteration, the process repeats with the results of $V_1 = 19.6$ mV and $V_2 = 19.4$ mV. As the simulation proceeds, the computed values eventually approach the correct values (this is not true in all cases). The final results are also shown in Figure 1.2. This waveform looks like a typical RC time constant, a wave shape that is so common when computing a step response that few designers would think of it as unusual and worthy of concern. However, in this circuit there are no capacitors and so it is clearly incorrect.

To continue the analogy begun in the Preface between simulators and automobiles: using relaxation methods is like driving without a seat belt, and using one-step relaxation methods is like driving without brakes. Such simulators should be avoided, or used with a great deal of skepticism.

Relaxation simulators have never achieved wide acceptance because they are not as reliable as direct method simulators, and because they are only applicable to digital MOS circuits and so must compete with faster logic and switch level simulators.

1.3 Circuit Simulation

A circuit simulator numerically computes the response of a particular circuit to a particular stimulus. To do so, the simulator simply formulates the circuit equations and then numerically solves them. The systems of equations generated by a circuit simulator cannot in general be solved explicitly, so other less direct methods must be used. In this section, the methods used to formulate and solve the circuit equations are briefly described.

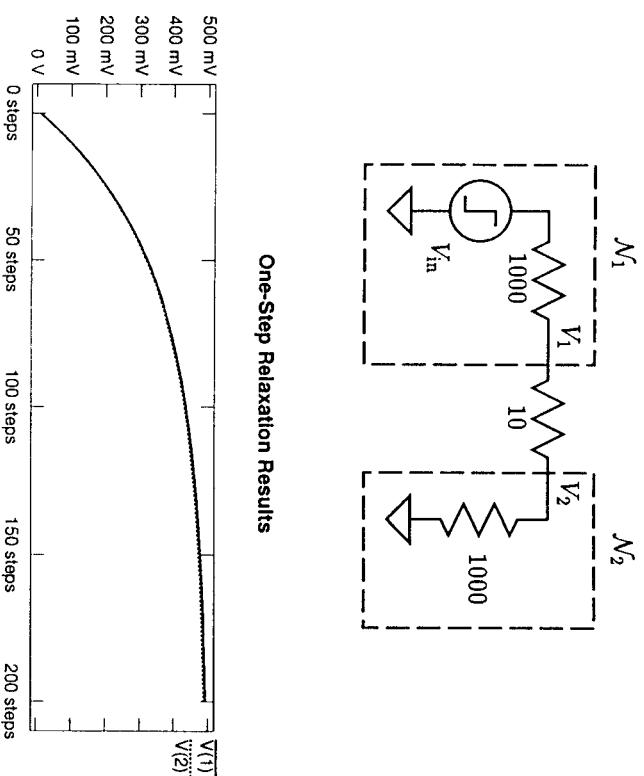


Figure 1.2: One-step relaxation incorrectly computes waveforms with long time constants on this purely resistive circuit.

Transient Analysis Transient analysis generates a system of nonlinear ordinary differential equations. There is no known method to directly solve these equations. The approach used in a circuit simulator is to discretize time, which converts the problem from the solution of a single system of nonlinear differential equations into that of solving a sequence of systems of nonlinear algebraic equations. It does so by replacing the time-derivative operator with a finite-difference approximation. One possible approximation is to

use an Euler formulation.

$$\frac{dq(t_i)}{dt} \approx \frac{q(t_i) - q(t_{i-1})}{t_i - t_{i-1}} \quad (1.2)$$

This is only one of many possible approximations available. Each approximation has its own advantages and disadvantages. However, any approximation is only accurate if the time step ($t_i - t_{i-1}$) is small relative to the time-constants present in the signals. If the time step is forced to be small everywhere to assure accuracy, the simulator is needlessly inefficient when signals are quiescent. Thus, an automatic time-step selection mechanism is needed in order to accurately and efficiently compute the transient behavior of most circuits.

Another extremely important aspect of transient analysis is that there is history in the calculations. The solution at every time point is built from the solution at the previous time point. As a result, an error made at one time point can degrade the accuracy of future time points. Whether the error accumulates or dissipates depends on the type of circuit being simulated. Certain circuits, such as high- Q circuits, circuits with long time constants, and chaotic circuits, are very sensitive to the build-up in error in transient analysis.

These issues and more are discussed in Chapter 4.

DC Analysis It happens that it is also not possible to explicitly solve a system of nonlinear algebraic equations. Such systems of equations are generated during DC analysis as well as every step of a transient analysis. To find the solution of a nonlinear system of equations, simulators formulate and solve a sequence of linear systems of equations in a process called Newton's method. Newton's method is an iterative process that continues until some criteria for stopping is satisfied. The accuracy of the solution depends directly on these stopping or convergence criteria. In some instances, Newton's method does not converge and so the criteria are never satisfied. In this case, the simulator fails. These issues are described in Chapter 2.

AC Analysis In AC analyses, the circuit is driven with 'small' sinusoidal signals and the steady-state, or eventual, solution is cal-

culated. Since the stimulus is small, the circuit can be linearized and all resulting signals will be sinusoids. AC analysis is an efficient way to compute transfer functions that does not have the accuracy problems of transient analysis or the convergence problems of DC analysis. AC analysis, along with several of its variations, are described in Chapter 3.

It is also not possible to directly solve a system of linear equations. Such systems are generated during AC analysis, as well as when solving the nonlinear equations present in DC and transient analysis. To find the solution of a system of N linear equations, circuit simulators use a process called Gaussian elimination or LU factorization, which converts the system of linear equations into a sequence of individual linear equations that can (finally) be solved directly. This process is not described in this book because it is generally reliable and accurate (in the context of circuit simulation) [duff86].

1.3.1 Equation Formulation

Circuit simulators compute the response of circuits by formulating a set of equations that represent the circuit and solving them. To form the equations, the simulator combines mathematical models of the individual components with equations that describe how the components are interconnected. The interconnection equations are derived from Kirchhoff's voltage and current laws.

1.3.1.1 Kirchhoff's Laws

Kirchhoff's Current Law (KCL)

The sum of all currents flowing out of a node at any instant is zero.

Kirchhoff's Voltage Law (KVL)

The algebraic sum of all branch voltages around a loop at any instant is zero.

Both of these laws imply that a node is infinitely small, or at least so small that there is negligible difference in potential anywhere on

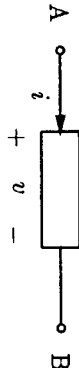


Figure 1.3: The associated reference directions for voltage v and current i for a branch with terminals A and B.

the node and negligible accumulation of current.

1.3.1.2 Associated Reference Direction

The reference directions for a particular branch are shown in Figure 1.3. The reference direction for voltage is indicated by the plus and minus symbols located near each terminal. Given the reference direction chosen, the branch voltage is positive whenever the voltage on the terminal marked with a plus sign (A) is larger than the voltage on the terminal marked with a minus sign (B). Similarly, the current is positive whenever it moves in the direction of the arrow (in this case from A to B). By convention, circuit simulators use *associated reference directions*. The reference directions for voltage and current are said to be associated if a positive current enters a branch by the terminal marked with the plus sign and exits the branch by the terminal marked with the minus sign. The reference directions shown in Figure 1.3 are associated.

1.3.2 Differential Equations

Circuit simulators generally have built-in parameterized mathematical models for common electrical components. Model parameters, along with circuit connectivity information are provided by a netlist. From these the simulator constructs a set of ordinary differential equations that describe the circuit. Given a stimulus and an initial condition, the equations are solved for the response of the circuit. Generally, modern circuit simulators use *modified nodal analysis* to

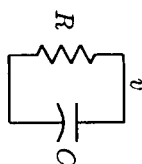


Figure 1.4: A simple RC circuit.

formulate the circuit equations. However, it is sufficient for our purposes to use the simpler *nodal analysis* [chu87]. The nodal analysis equation for a circuit containing resistors, capacitors, and current sources is simply

$$i(v(t)) + \frac{d}{dt}q(v(t)) + u(t) = 0 \quad (1.3)$$

$$v(0) = a \quad (1.4)$$

where $v(t)$ is the node voltages, $u(t)$ is the current entering the nodes from the current sources, $q(v(t))$ is the charges entering each node from capacitors, $i(v(t))$ is the currents entering each node from resistors, t is time, and a is the initial condition. Equation (1.3) simply says that at each node, the current from the resistors ($i(v(t))$), the capacitors ($\frac{d}{dt}q(v(t))$), and the current sources ($u(t)$) connected to the node must sum to zero. In other words, (1.3) is an explicit statement of Kirchhoff's current law.

Equation (1.4) is the initial condition. Equation (1.3) is not sufficient to define the solution of a circuit containing energy storage component such as capacitors and inductors. Consider the simple RC circuit shown in Figure 1.4. Use (1.3) to write KCL.

$$\frac{v(t)}{R} + \frac{d}{dt}Cv(t) = 0 \quad (1.5)$$

The solution takes the form

$$v(t) = V_0 e^{-\frac{t}{RC}} \quad (1.6)$$

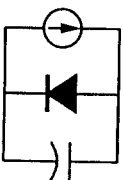


Figure 1.5: A simple circuit that can be analyzed with nodal analysis.

However, this is not one solution, but rather a continuum of solutions parameterized by V_0 . Including the initial condition (1.4) adds another constraint that eliminates all but one the solution given by $V_0 = a$.

Consider as an example the circuit of Figure 1.5. The circuit has only one node and so is represented with the single nodal analysis equation

$$i_d(v(t)) + \frac{d}{dt}Cv(t) + i_s(t) = 0 \quad (1.7)$$

where input current is i_s and the diode current is

$$i_d(v(t)) = I_s \left(e^{\frac{v(t)}{V_T}} - 1 \right) - \frac{C_{j0}}{\sqrt{1 - v(t)/\phi}} \frac{dv}{dt}. \quad (1.8)$$

Semiconductor devices such as diodes and transistors are modeled as the combination of nonlinear resistors and nonlinear capacitors.

1.3.3 Analyses

Transient analysis solves the nonlinear ordinary differential equations that describe the circuit over a user specified time interval. The user of the simulator must first specify the stimulus waveforms and the initial conditions, and transient analysis computes the voltage and current waveforms.

DC analysis solves for a equilibrium point, which is a solution that does not vary with time. The DC analysis equations are formulated

1.4. Modeling

13

from (1.3) by assuming that $\frac{d}{dt}v(t) = 0$ and $\frac{d}{dt}u(t) = 0$ for all t . Thus,

$$i(v_{dc}) + u_{dc} = 0 \quad (1.9)$$

This equation is solved to find the DC operating point.

Finally, AC analysis computes the steady-state response of the circuit to a small sinusoidal signal. Since the signal is assumed to be small, the circuit is linearized about the DC operating point before computing the response.

1.4 Modeling

Given the topology of a circuit, simulators construct the circuit equations from mathematical models of the components contained in the circuit. The models may either be built-in, as is generally the case for semiconductor models in SPICE-like simulators, or they may be specified by the user using some type of modeling language. An example of such a modeling language is SpectreHDL¹, which is short for Spectre's Hardware Description Language. Modeling is a difficult art. The quality of the models directly affects the accuracy, the efficiency, and the robustness of the simulator. For example, Spectre is typically 3–5 times faster than most versions of SPICE and is considerably better at converging. Both of these benefits are at least partially attributable to the very thoughtful and careful manner in which the models were implemented.

As important as good models are to the behavior of the simulator, they are not discussed in depth in this book. Modeling is a very large and difficult subject in its own right, with its own set of issues. In addition, few people that use circuit simulators actually model their own components. For these reasons I simply refer you to the excellent book that should be available soon from Jeng [jeng] and to the book currently available from Antognetti and Massobrio [antognetti93].

Also important, but not mentioned, is the whole topic of behavioral modeling. With electronic systems becoming more complicated,

¹ SpectreHDL™ is a comprehensive behavioral modeling capability and language that works with Spectre.

analog hardware description languages such as SpectreHDL are becoming much more prevalent and important. Behavioral models are used when simulating systems at high level. They are also combined with the simulator's built-in primitive models to support top-down and bottom-up design methodologies. Finally, behavioral models are written to implement test circuitry, such as for complex stimuli generation and measurements, which greatly amplifies the power of the base simulator. However, the topic of how to write good behavioral models is vast, and perhaps a task for another day.

1.5 Summary

A good introductory book on SPICE in the context of analog circuit design was written by Vladimirescu [vladimirescu94]. An interesting book that discusses both design and simulation issues for high-frequency circuits is available from Maas [maas88]. More information on the mechanics and numerics of circuit simulation from a simulator developer's perspective is found in the books by Sangiovanni-Vincentelli [sangiovanni81], McCalla [mccalla87], or Vlach and Singhal [vlach83]. Information on more esoteric simulation algorithms is found in the books by White and Kundert [white86, kunder90]. For information about the details of the more common semiconductor models available in SPICE and associated background material, there is the excellent book currently being written by Jeng [jeng] or the book edited by Antognetti and Massobrio [antognetti93]. Finally, information on macromodeling in SPICE is given in the book by Connelly and Choi [connelly92].

Chapter 2 DC Analysis

2.1 Introduction

Many of the analyses available for SPICE and Spectre compute operating points. For example, SPICE's .op analysis computes and outputs information about the operating point such as the voltage, current, power, etc. at each component. The .dc analysis computes the operating point as a function of some independent variable. The linear small signals analyses, such as .ac and .noise, first compute the operating point and then linearize the circuit about that operating point before computing the small-signal behavior of the circuit. Finally, the transient analysis computes an operating point for the initial state of the circuit. Clearly the ability to reliably compute an accurate operating point is very important.

In general operating points are simply snap-shots of some solution trajectory. In DC analysis, the operating points are also assumed to be equilibrium points. Equilibrium points are constant-valued operating points. In other words, equilibrium points are solutions that do not change with time. A circuit cannot reach an equilibrium point if the stimulus is still changing, so the first step of a DC analysis is to configure the independent sources so they are constant. In addition, since all waveforms are constant-valued at equilibrium points, $dv/dt = 0$ and $di/dt = 0$ and so capacitors act as open circuits and inductors act as short circuits. Therein lies the basic algorithm for computing an equilibrium point.

1. Configure all independent sources to be constant valued.
2. Replace all capacitors with open circuits.
3. Replace all inductors with short circuits.

Solving the equations that describe the resulting system gives an equilibrium point. This system of equations is nonlinear and algebraic (no time derivatives or integrals). Solving large nonlinear systems of algebraic equations is a difficult challenge. The only way to solve general nonlinear equations is to use iterative methods such as Newton's method. However, even these methods are not guaranteed to work. In fact, there is no practical algorithm that always works. When an iterative method fails, it does so by not converging to a solution. Convergence issues are the primary concern when using DC analysis because convergence is problematic for circuit simulators, especially on large circuits. The accuracy of the solution is a secondary issue, because once the simulator converges, the results are rarely inaccurate. In this chapter, the focus is mainly on convergence, though accuracy issues are also discussed.

2.2 DC Analysis Theory

As stated in Section 1.3.2 on page 10,

$$i(v(t)) + \frac{d}{dt}q(v(t)) + u(t) = 0 \quad (2.1)$$

$$v(0) = a \quad (2.2)$$

is solved to find the transient behavior of the circuit. Also of interest is the DC solution, or equilibrium point, which is defined as a solution to (2.1) that does not vary with time (the input $u(t)$ is assumed to be constant valued). The DC equations are formulated from (2.1) by assuming that $\frac{d}{dt}v(t) = 0$ for all t , and so $\frac{d}{dt}q(v(t)) = 0$ (this constraint replaces the initial condition constraint of (2.2)).

$$i(v_{dc}) + u_{dc} = 0 \quad (2.3)$$

Circuit simulators solve this equation to compute the DC solution or operating point. However, it must be stressed that the solution

2.2. DC Analysis Theory

17

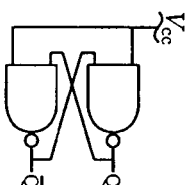


Figure 2.1: A latch, a circuit with 3 equilibrium points, one of which is unstable.

computed is not necessarily unique, nor is it required to be stable. For example, consider the latch shown in Figure 2.1. This circuit has three equilibrium points, $Q = V_{cc}$, $Q = 0$, and $Q = \frac{1}{2}V_{cc}$. The first two solutions are stable. In other words, if the circuit is at one of these solutions, and is perturbed slightly, it eventually returns to the same solution. The last equilibrium point is unstable, meaning any perturbation causes the voltages to drift away from this solution and eventually ends up at one of the other two solutions. The simulator is just as likely to find an unstable solution as it is to find a stable one.

It is important to understand that:

1. Circuits sometimes have more than one DC solution.
2. The DC solution computed by the circuit simulator may be unstable.

The fact that circuit simulators do not really distinguish between stable and unstable solutions allows a circuit simulator to compute a DC solution for oscillators, which do not generally have stable DC equilibrium points.

2.2.1 Solving Nonlinear Equations

The set of equations that result from DC analysis (2.3) and on every step of a transient analysis (4.12) are nonlinear algebraic systems and so in general cannot be solved directly. These equations can be solved by the Newton-Raphson algorithm (also referred to as Newton's method), which converts the solution of a nonlinear equation into the solution of a sequence of linear equations. Newton-Raphson starts with an initial guess. It then linearizes the circuit about that guess, and solves the linear circuit. The circuit is then re-linearized about the new point and the procedure repeats until the process converges.

Newton's method solves equations of the form

$$f(\hat{v}) = 0 \quad (2.4)$$

for $\hat{v}$ by starting with an initial guess called $v^{(0)}$ and repeatedly solving the Newton-Raphson iteration equation

$$J(v^{(k)})(v^{(k+1)} - v^{(k)}) = -f(v^{(k)}) \quad (2.5)$$

or

$$v^{(k+1)} = v^{(k)} - J^{-1}(v^{(k)})f(v^{(k)}) \quad (2.6)$$

for $v^{(k+1)}$ (the value of v on the $k + 1^{\text{th}}$ iteration) until some convergence criteria are met. $J(v) = \frac{d}{dv}f(v)$ is called the Jacobian of f at v . Since both $f(v)$ and v are N -dimensional vectors, $J(v)$ is an $N \times N$ matrix. It represents the circuit linearized about v . This process is explicitly stated in Algorithm 2.1 and is illustrated graphically with a simple scalar example in Figure 2.2.

The sequence generated by (2.6) is guaranteed to converge to $\hat{v}$ if f is continuously differentiable, if the solution is isolated (this concept is discussed in Section 2.2.3.1 on page 25), and if $v^{(0)}$ is sufficiently close to $\hat{v}$. In circuit simulation, none of these three conditions are guaranteed, and so neither is convergence. Failure to converge is probably the biggest complaint designers have with circuit simulators.

Newton-Raphson has the very desirable property of quadratic convergence, meaning that once it is close to the solution, it reduces the error by squaring it on each iteration. The end result is that once

Step 0: Initialize

set $k \leftarrow 0$
choose $v^{(0)}$

Step 1: Linearize about $v^{(k)}$

$J_f(v^{(k)}) = \partial f(v^{(k)})/\partial v$
where J_f is the Jacobian of f .

Step 2: Solve the linearized system

$v^{(k+1)} \leftarrow v^{(k)} - J_f^{-1}(v^{(k)})f(v^{(k)})$

Step 3: Iterate

set $k \leftarrow k + 1$
if not converged, go to step 1.

Algorithm 2.1: Newton-Raphson algorithm for finding $\hat{v}$ such that $f(\hat{v}) = 0$:

Newton-Raphson is close to the solution, the solution is found very accurately with only a few more iterations.

2.2.2 Convergence Criteria

Newton-Raphson is a method that takes an initial guess of the solution of a system of nonlinear equations, and refines it making it more and more accurate on each iteration. However, in its pure form, Newton's method never terminates. A way of deciding when the iteration should be terminated is needed.

2.2.2.1 Absolute Convergence Criteria

The Newton-Raphson iteration is considered to have converged, and therefore can be terminated, only after the approximate solution satisfies two convergence criteria. These two convergence criteria are given in simplified form first, with a more practical form given later.

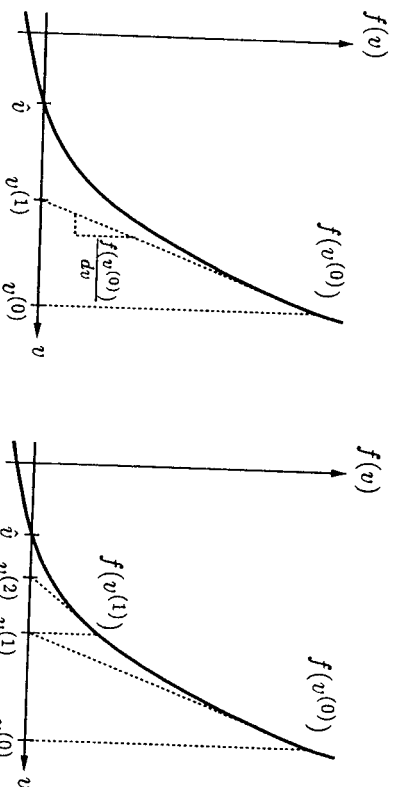


Figure 2.2: Newton's method applied to find the value $\hat{v}$ such that $f(\hat{v}) = 0$ (the value of v where the curve crosses the horizontal axis). The process starts by guessing the value $v^{(0)}$. The function is linearized about $v^{(0)}$ and solved for the next guess $v^{(1)}$. If all goes well, $v^{(1)}$ is closer to the solution $\hat{v}$ than was $v^{(0)}$, and $v^{(k)} \rightarrow \hat{v}$ as $k \rightarrow \infty$. The process terminates when $v^{(k)}$ is sufficiently close to $\hat{v}$.

The first criterion specifies that KCL should be satisfied to a given degree,

$$|f_n(v^{(k)})| < \epsilon_f \quad (2.7)$$

where ϵ_f is some small positive number. The second tries to control the error in the solution by asserting that the difference between the last two iterations must be small,

$$|v_n^{(k)} - v_n^{(k-1)}| < \epsilon_x \quad (2.8)$$

where ϵ_x is some small positive number.

A simulator considers $v^{(k)}$ a solution if (2.7) and (2.8) are both satisfied. It is necessary to assure that both conditions are satisfied to be certain that the solution computed by Newton's method is correct. However, most simulators (in particular, those that are descendants

of SPICE) only check condition (2.8). This occasionally results in a condition called false convergence, which occurs when the iteration is terminated prematurely with (2.4) far from being satisfied because progress on one iteration is slow (and therefore $v^{(k)} - v^{(k-1)}$ is small and so (2.8) is satisfied). These simulators try to avoid this problem by using heuristics (generally that $|f_n(v^{(k)}) - f_n(v^{(k-1)})| < \epsilon_f$), but these sometimes fail. The reason they fail is that neither condition actually verifies that the equations are being solved (that KCL is being honored). Instead they are satisfied when the current iteration is close to the previous one. Thus, if the rate of convergence becomes slow, these conditions are satisfied without the iteration being close to the solution. The progress on an iteration in both (2.7) and (2.8) would be slow if the Jacobian is wrong because of an error in the implementation of a model. On the other hand, conditions (2.7) and (2.8), which are used in Spectre, do not exhibit false convergence. Condition (2.7) assures that KCL is approximately satisfied, and (2.8) bounds the error in the solution.

Why the Residue Criterion is Needed In general, the residue criterion (2.7) is important when the impedance at a node is small. For example, consider a strongly forward-biased pn -junction. Even small changes in voltage across the junction result in large changes in the current through the junction. In such circuits, the residue criterion is more important than the update criterion (2.8) for maintaining the accuracy of the solution.

Why the Update Criterion is Needed The update criterion (2.8) is important when the impedance at a node is large. Consider a node that is isolated from others by a reverse-biased pn -junction. There is a large range of voltages that result in the current through the junction being less than the absolute current tolerance. In this situation, the update criterion is more important than the residue criterion for maintaining the accuracy of the solution.

2.2.2.2 Relative Convergence Criteria

While conceptually simple, the convergence conditions given by (2.7) and (2.8) are not used as given in practice because the criteria do not tolerate changes in scale well. Consider condition (2.8), a better criterion is:

Newton Update Convergence Criterion

The solution updates are said to have converged if

$$|v_n^{(k)} - v_n^{(k-1)}| < \text{reltol } v_{n\max} + \text{vntol} \quad (2.9)$$

where typically $v_{n\max} = \max(|v_n^{(k)}|, |v_n^{(k-1)}|)$.

By default, **reltol** is 0.001 and **vntol** (called **vabstol** in Spectre) is $1\mu\text{V}$.

reltol is called the relative convergence tolerance because it specifies how small the update must be relative to the node voltage. **reltol** allows you to simulate high voltage circuits and low voltage circuits without adjusting the convergence criteria. **vntol** is referred to as an absolute tolerance. It becomes important when the solution on a particular node is near zero. In this case, just using the **reltol** criterion would force the update to be microscopic before convergence was allowed. In some cases the required update would be smaller than the computer round-off error, in which case convergence would never occur. **vntol** prevents these problems from occurring by causing any update smaller than **vntol** to be accepted.

Criterion (2.9) overcomes several problems that plague (2.8). First, (2.8) does not automatically scale itself to the problem, and so ϵ_x would have to be manually adjusted to fit the problem. Second, if the solution at node n is large and at node m is small, then ϵ_x must be chosen fairly large so that convergence is not precluded at node n , which results in convergence being checked very loosely at node m .

The residue convergence criterion of Equation (2.7) can also be improved.

Newton Residue Convergence Criterion

The residue is said to have converged if

$$|f_n(v^{(k)})| < \text{reltol } f_{n\max} + \text{abstol} \quad (2.10)$$

where typically $f_{n\max}$ is the absolute value of the largest current entering node n from any one branch.

Typically, **abstol** (called **labstol** in Spectre) is 1pA .

2.2.3 Convergence

Failure of circuit simulators to converge is a serious problem. One large electronics company estimated that their circuit designers spent an average of two hours a day trying to cajole their simulators into converging. Also, convergence issues, more than anything else, limits the size of circuits that can be simulated. Because convergence problems get worse on larger circuits, designers generally find that as circuits increase in size, a simulator generally fails to converge before its needs for memory and time exceed what is available.

Circuit simulators employ Newton's method to solve the nonlinear systems of equations that are formed during DC and transient analyses. Newton's method is an iterative procedure that starts with a guess or estimate of the solution and refines it on each iteration until it converges to the solution. However, there is no guarantee that Newton's method actually converges to the solution.

On each Newton-Raphson iteration, the circuit is linearized and the solution to this linear circuit is computed and used in the next iteration. If Newton's method starts from a point sufficiently close to the solution, it is guaranteed to find that solution if the component model equations are continuously differentiable and if the solution is isolated. A solution is isolated if changing it slightly in any direction would cause Kirchhoff's laws to be violated. A great deal of energy has been expended to assure that model equations in any reputable simulator are continuously differentiable. However, convergence problems can still occur due to poor starting points. Avoiding isolated solutions is largely the responsibility of the user.

(though large g_{min} resistors are added by most simulators to reduce the likelihood of nonisolated solutions).

As a practical matter, it has been observed that errors in specifying circuit connectivity, component values, or model parameter values often cause convergence problems. Circuit simulators provide a topology checker that finds many connectivity problems before Newton's method is applied. However, if convergence problems occur, the netlist should be carefully checked both for topological errors and unreasonable model or instance parameters. Spectre has parameter value sanity checking, which prints a warning if the value of any parameter is outside a reasonable range. This feature generally fixes roughly a third of all convergence problems.

Another thing that causes convergence difficulties is very small floating resistors. These generally occur as parasitic resistors in semiconductors or as current-sense resistors. Any error in the voltages computed for nodes with small resistors attached results in large error currents. Consider a $1\mu\Omega$ floating resistor embedded in a typical semiconductor circuit with 1V on each node. If there were 1pV error on one node, it would result in a $1\mu A$ error current, which would often not satisfy the KCL convergence criterion. In fact, the error in the voltage on either node would have to be less than 10–18V in order to satisfy an absolute current convergence criterion of $abstol=1pA$. This is generally smaller than the numerical resolution of the computer.

In Spectre, this problem would result in convergence problems because the convergence criterion for KCL is never satisfied. However, several steps have been taken in Spectre to solve or avoid the problem. In SPICE this particular problem has been avoided by not using a convergence criterion for KCL. However, the disadvantage of this approach is that it makes the simulator subject to false convergence.

Another problem occurs when using very small floating resistors. When the Jacobian matrix is formed by summing of all conductances attached to a node, the very large conductance of the small resistor can dominate over the small conductances of semiconductors attached to the same node. If the large and small conductances connected to a node are of vastly different size, the small conduc-

tances would be lost due to numerical round-off. This results in the factorization of the Jacobian being ill-conditioned and increases the likelihood that there will be small errors in the node voltages.

You should try to avoid small floating resistors. Use 0-volt voltage sources as current probes rather than small-valued resistors and discard overly small parasitic resistors in semiconductors. Spectre automatically deletes all parasitic resistors smaller than a particular value, given by the model parameter `minr`. If it is necessary to use small valued parasitic resistors, increase `iabstol` to avoid convergence problems. Spectre automatically reformulates small-valued non-parasitic resistors to avoid the large conductance.

2.2.3.1 Isolated Solutions

Newton's method requires that the solution be isolated. Many circuits have solutions that are not isolated. If you attempt to simulate a circuit that does not have isolated solutions, the simulator usually fails with an obscure complaint about the matrix or the Jacobian being singular. If the nonisolated solutions result from a structural property of the circuit, the topology checker usually identifies the problem. In particular, it searches for nodes that do not have paths to ground for DC currents. For example, consider the DC analysis of a circuit containing a subcircuit that is completely isolated from ground except possibly for capacitors, such as N_2 in Figure 2.3. While the node voltages within the subcircuit are well defined with respect to other nodes in the subcircuit, they may all be raised or lowered in unison with respect to ground without violating Kirchhoff's laws. Thus there is an infinite continuum of solutions.

Another scenario involves a loop of ideal inductors. Since ideal inductors have zero resistance at DC, a loop of inductors supports a nonzero current even with no applied voltage. Thus, an arbitrary amount of current may be circulating in the loop without affecting the validity of the solution.

The topology checker will not find situations that cause nonisolated solutions that depend on component parameter values. A common example of this situation is when a CMOS inverter, as shown in

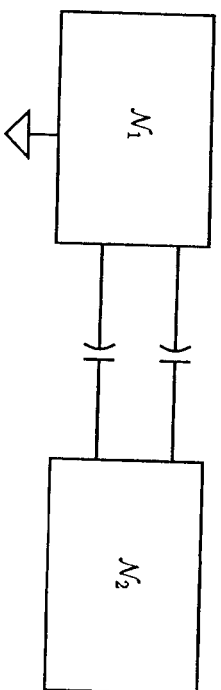


Figure 2.3: This circuit does not have an isolated solution because the DC voltages on the nodes in N_2 can be raised or lowered in unison without violating Kirchhoff's laws.

Figure 2.4, is constructed with FETs that have their model parameters set such that they have infinite output impedance in saturation. When either the N- or P-type device is in the ohmic region, the solution is unique, but when both devices are saturated, there exists a finite range of output voltages that all satisfy Kirchhoff's current law. In this situation, the linearized circuit that Newton's method forms on each iteration (the Jacobian) is singular. This is one of several situations that cause problems when using overly simplified models.

The topology checker also cannot find the situations that cause non-isolated solutions when they result from the behavior of nonlinear components. For example, consider the simple CMOS nand gate shown in Figure 2.5.

SPICE tries to avoid this problem by adding a resistor with conductance g_{min} across every nonlinear component. This generally solves the problem. If the problem still exists, Spectre prints an error message indicating that the Jacobian is singular and gives the names of the nodes or components where the problem was detected. The search to determine why the circuit has a nonisolated solution should be focused on the given nodes or loops containing the given components.

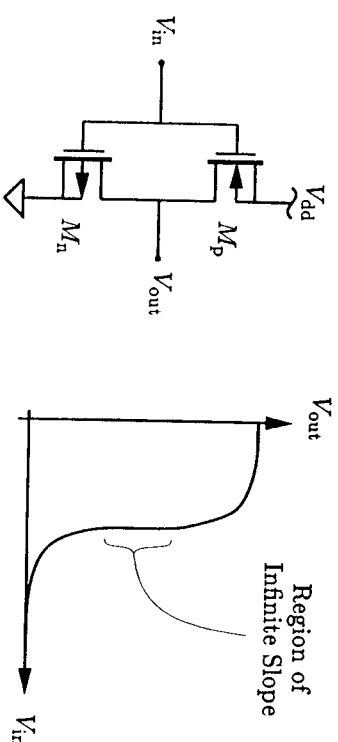


Figure 2.4: If $V_{in} = \frac{1}{2}V_{dd}$ and if both M_n and M_p have infinite output resistance, then any value for V_{out} that results in both M_n and M_p remaining saturated is valid and therefore the circuit has a nonisolated solution.

2.2.3.2 Providing a Good Starting Point

You can specify the starting point for Newton's method to SPICE using the `nodeset` statement. If your simulator fails to converge and you have already eliminated topology and parameter errors as the source of the problem, then using the `nodeset` statement to specify a reasonable estimate of the solution is your best hope for getting convergence. Providing a complete set of node voltages and branch currents is best, but if that is not possible, an incomplete set often helps. If given an incomplete set, SPICE tries to compute the remaining unknowns by performing an initial DC solution with the given voltages and currents forced to their specified values. Node voltages are forced by connecting a voltage source in series with a 1 Ohm resistor (the resistance is given by the `rforce` option in Spectre). Once the solution to this modified circuit is found, it is used as a starting point for Newton's method applied to the original circuit. The `nodeset` statement is used not only to aid convergence, but also to bias the simulator towards finding a particular solution when more

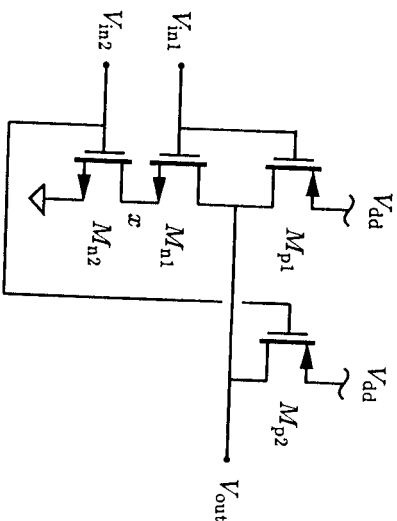


Figure 2.5: Node x floats when both inputs are low. Thus any voltage on node x that results in both M_{n1} and M_{n2} remaining off is valid and therefore the solution is not isolated, which causes convergence failure unless g_{min} is employed.

than one exist.

Once a solution has been found, Spectre can save it in to a file and recall it for use later in the simulation session, or even in subsequent simulation sessions.

2.2.3.3 Forcing a Solution

If a circuit has a non-isolated solution, nodesets alone are not sufficient for convergence. Usually, this means the circuit itself should be modified to eliminate the problem. Spectre provides two mechanisms to overcome these problems. First, if Spectre's topology checker identifies floating nodes, it automatically installs g_{min} from the floating node to ground. This is usually sufficient to allow the simulation to continue. However, in some rare cases either it is not sufficient, or the topology checker cannot identify the cause of the non-isolated so-

lution. For these cases, Spectre provides the *node force*. Node forces differ from nodesets in that the voltage source and resistor used to force the node remain connected to the circuit during the full DC analysis, and are only removed for other analyses. The operating point computed by the DC analysis with a node force is in effect is *not* an equilibrium point.

2.2.3.4 Forces Versus Nodesets

Nodesets and node forces are implemented in a similar manner but their effect is quite different. Node forces actually define a part of the solution, whereas nodesets only influence it. Also, while nodesets affect both the DC and transient analyses, node forces only affect the DC analysis (in transient analysis, initial conditions act analogously to node forces). Nodesets are usually used only as a convergence aid and do not affect the final results (unless the circuit has more than one solution, such as a latch, in which case nodesets does bias the simulator towards finding the solution closest to the nodeset values). In contrast, node forces directly affect the results. Finally, if only nodesets are used, then the operating point found is an equilibrium point, which is not the case if any node forces are used.

To make the situation more confusing, some simulators confuse the concept of nodesets and node forces. They provide node forces, but call them nodesets or initial conditions. SPICE2 and SPICE3 correctly implement nodesets and do not provide node forces.

2.2.3.5 Continuation Methods

Given an initial guess from one of the above methods, SPICE applies Newton's method attempting to find a solution. If it is unsuccessful, it attempts to use continuation or homotopy methods to find the solution (you must explicitly tell SPICE2 to use continuation methods by setting the `it16` option nonzero). Continuation methods are slower but more robust than plain Newton's method. They start by modifying the circuit in such a way that the solution to the modified circuit is known or easy to compute, and such that a parameter controls the amount of modification. Once the solution has been

found for the modified circuit, the parameter is slowly returned to the original value, which causes the circuit to return to its original form. As the parameter is changed, the solution is computed at each step, using the solution from the previous step as the starting point. As long as the solution changes continuously as a function of the parameter and the steps are small enough, the previous solution is always a good starting point and Newton's method always converges.

In other words, the circuit equations are written in the form

$$f(v(\lambda), \lambda) = 0 \quad (2.11)$$

where λ is chosen as either a natural or contrived parameter of the circuit such that

1. For $\lambda = 0$ the solution $v(0)$ is known in advance or is easy to compute.
2. For $\lambda = 1$, $f(v(1), 1) = f(v)$.
3. The trajectory $v(\lambda)$ is a continuous function of λ for $0 \leq \lambda \leq 1$.

This results in a continuous curve between the solution of the initial system ($v(0)$, which is easy to find) and the desired solution of the final system ($v(1) = \hat{v}$), as shown in Figure 2.6. The next step is to simply follow the curve from $v(0)$ to $v(1)$. The procedure for doing so is to slowly step λ from 0 to 1, computing $v(\lambda)$ at every step. Use $v(\lambda_k)$ as the initial guess for $v(\lambda_{k+1})$. If $|v(\lambda_{k+1}) - v(\lambda_k)|$ is small, and if $v(\lambda)$ is continuous, then $v(\lambda_k)$ is in the region of convergence for Newton's method, and the iteration is guaranteed to converge.

There are three continuation methods in common use in simulators today, source stepping, **gmin** stepping, and pseudo-transient. SPICE2 provides source stepping, SPICE3 provides **gmin** stepping, ASTAP provides pseudo-transient, and Spectre provides all three. Spectre also provides an additional proprietary method that is referred to as damped-pseudo-transient or **dpttran**. It is similar to pseudo-transient, but is more likely to converge. MetaSoftware also claims to offer damped pseudo-transient in HSPICE, but in fact it is nothing more than regular pseudo-transient. The continuation methods are

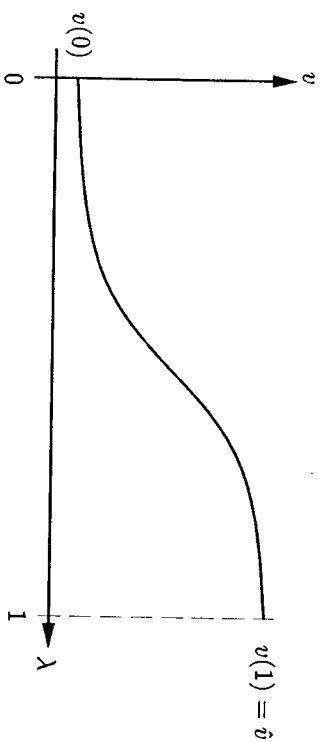


Figure 2.6: The trajectory of the solution $v(\lambda)$ in a typical continuation method.

considerably slower than the base DC solution algorithm. However, in Spectre once a DC solution has been found it can be written to a file and recalled for later use. This speeds subsequent analyses considerably.

gmin-stepping starts by placing small resistors in parallel with all nonlinear devices, and a solution is computed. The solution is easy to compute because the nonlinear behavior of the devices is swamped out by the resistors. The size of the resistors is slowly increased, and the solution is computed each time the resistors are changed. Eventually the resistors are so large that they no longer affect the circuit. They are then removed completely and the DC solution computed.

Source stepping starts by setting all source voltages and currents to zero, and slowly ramping them to their full value. The solution is recomputed each time the source level is changed, with all but the final solution discarded.

With both **gmin**- and source-stepping there are three types of discontinuities in $v(\lambda)$ that causes them to fail, simple discontinuities, folds and bifurcations. Simple discontinuities, as shown in Figure 2.7,

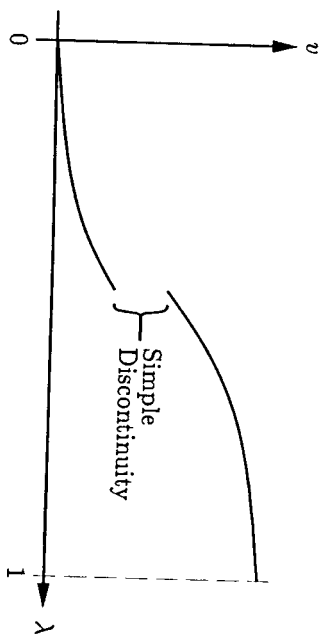


Figure 2.7: Simple discontinuity in homotopy due to discontinuous models causes g_{min} - and source-stepping to fail.

are caused by discontinuous component constitutive relations. Folds, shown in Figure 2.8, occur when the solution curve doubles back on itself and result in the circuit having multiple solutions for at least some values of $v(\lambda)$. Folds are remarkably common. Bifurcations, as shown in Figure 2.9, occur when the solution curves cross each other. Bifurcations result when the circuit and the starting point $v(0)$ is symmetric, and when there exists a non-symmetric solution for some value of λ . For example, the simple latch shown in Figure 2.10 can result in bifurcation in the homotopy because it is symmetric and has at least one non-symmetric solution. To avoid bifurcations, use nonsymmetric nodesets for symmetric circuits. Figure 2.11 shows the homotopy that results by applying nonsymmetric nodesets to a symmetric circuit.

As a rule, folds and bifurcations plague source-stepping more than g_{min} -stepping.

With the pseudo-transient method, a capacitor is installed in parallel with each nonlinear device with a zero initial condition (the capacitors and inductors naturally in the circuit are ignored by setting them to zero). Time (rather, the “pseudo-time”) is swept from zero

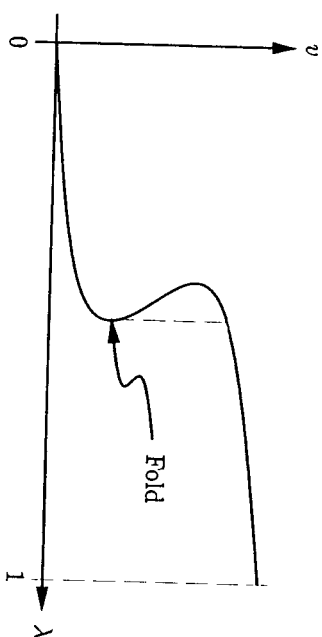


Figure 2.8: Fold in homotopy causes g_{min} - and source-stepping to fail.

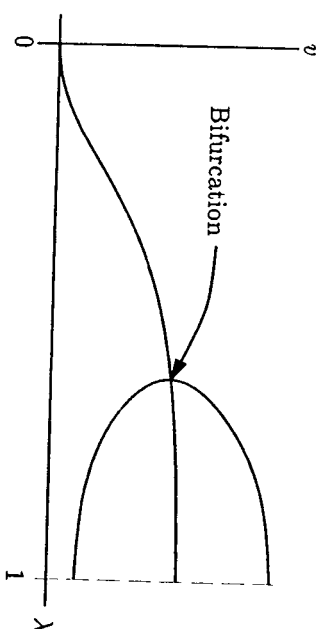


Figure 2.9: Bifurcation in homotopy caused by symmetry causes g_{min} - and source-stepping to fail.

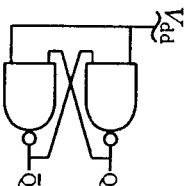


Figure 2.10: The symmetry of this latch causes continuation methods to fail unless the starting point is non-symmetric.

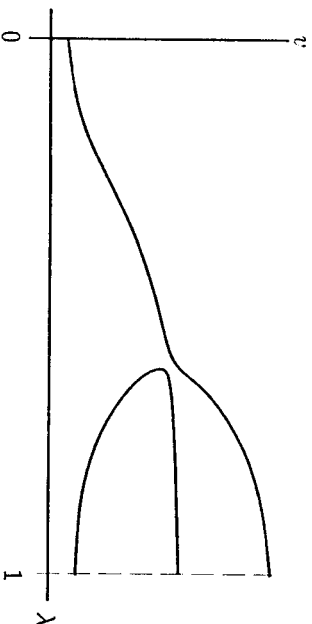


Figure 2.11: Bifurcation in Figure 2.9 is eliminated with non-symmetric nodesets.

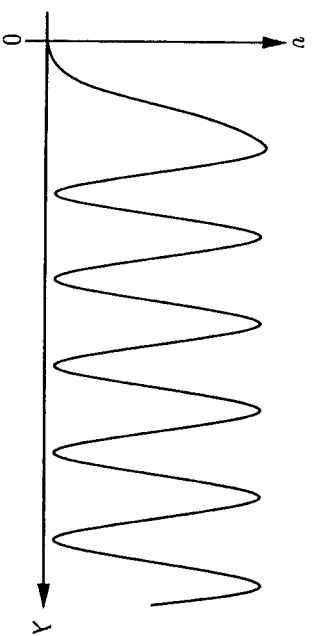


Figure 2.12: The pseudo-transient continuation method does not suffer from discontinuities, but fails if the circuit oscillates.

to infinity in an attempt to find the DC solution. The capacitors should cause the solution waveform to be a continuous function of time, the continuation parameter. This continuation method works well as long as adding the capacitors does not convert the circuit into an oscillator, as shown in Figure 2.12.

2.3 DC Analysis Practice

2.3.1 Remedies for Convergence Problems

Here are some suggestions to try if you are having convergence problems.

1. Heed any warnings the simulator gives. This may seem obvious, but I have found that it is not so obvious that people always do it. Whenever a problem occurs with a simulation, go back and scan the messages generated by the simulator and look for clues to why the problem occurs. With Spectre, enable the diagnostics by setting `diagnose=yes` on the options statement.

2. Carefully check all parameter values to assure they are reasonable. Use the `list` and delete the `nomod` options with SPICE to get a complete listing of the parameter values. Spectre automatically checks the parameter values for you if you use the `+param` command line option. The list of 'reasonable' values are kept in a file provided with Spectre, and so can be modified to suit your situation. Use the Spectre's info statement with `what=all` for a list of parameter values if you would like to check the parameter values by hand. This is done quickly if `extremes=only` is specified, in which case Spectre only prints the largest and smallest values specified for each parameter.
3. Check the circuit connectivity by using the `node` option with SPICE and using the info statement with Spectre (`what=nodes` or `what=terminals`).
4. Eliminate small floating resistors. These typically show up as parasitic resistors in the semiconductor or current-sense resistors. Use 0-volt voltage source as current probes. Discard very small parasitic resistors or increase `abstol` (`iabstol` in Spectre).
5. Check the direction of both independent and dependent current sources. Convergence problems result if current sources are connected such that they force current backward through diodes or *pn*-junctions.
6. Use `nodeset` statements to provide estimates for as many nodes as possible.
7. Enable whatever continuation methods the simulator supports (set `itl6` in SPICE to 20-100, or more if problems persist).
8. Increase the iteration limit (`itl1` in SPICE up to 500). When computing DC transfer curves, consider increasing `itl2` up to 500.
9. Consider increasing `gmin` above its default value of 10^{-12} S. Be careful not to increase it to a degree that it interferes with the proper operation of the circuit. However, be aware that if you increase `gmin` to the point where convergence is achieved, then

you must have changed the circuit in a significant way and it is likely that the results are incorrect.

10. Spectre allows many analyses to be run sequentially. If the problem is not with the first analysis, perhaps the solution from the previous analysis is far from the solution for this analysis. This is likely if a component parameter has been changed, the temperature has been changed, or the input sources have changed their value (DC and transient analysis often have different input source values). In these cases, recall a previous solution from a file or tell Spectre to start fresh (using `restart=yes`).
11. Set the region parameter (`off` is SPICE, `region` in Spectre) on all transistors and diodes correctly.
12. If you have convergence difficulties in the middle of a DC sweep, try using a smaller step size.
13. If your simulator fails to compute the DC solution to a circuit at an extreme temperature, but is able to compute it at room temperature and is able to sweep temperature, then try adding a DC analysis that sweeps temperature. Start at room temperature, sweep to the extreme temperature, and use the final solution to generate nodesets for the entire circuit.
14. Loosen tolerances, particularly `abstol` (`iabstol` in Spectre). If tolerances are set too tight, they might preclude convergence. The absolute tolerance should generally be set to around 10^{-6} , but no smaller than 10^{-9} , times the largest similar quantity present in the circuit. For example, in integrated circuits, the largest voltages typically range from 5 to 15 volts, so `vntol` (`vabstol` with Spectre) is set to $1\ \mu\text{V}$, and the largest currents typically range around $1\ \mu\text{A}$, and so `abstol` (`iabstol` with Spectre) is set to 1pA .
15. Try simplifying models. In particular, eliminate second-order effects and parasitic resistors (especially the nonlinear base resistors). Once the solution for the simplified solution is known, use it to generate nodesets for the original circuit.

16. If convergence difficulties occur when using nodesets or initial conditions, try changing the forcing resistance if possible (use the `rforce` option in Spectre).
17. Divide the circuit into smaller pieces and simulate them individually, but be careful to assure that the results will be close to what they would be if the rest of the circuit was present. Use the results to generate nodesets for the entire circuit.
18. Replace the DC analysis with a transient analysis and modify all the independent sources to start at zero and ramp to their DC values. Run the transient analysis well beyond the time when all the sources have reached their final value (remember that transient analysis is inexpensive when all of the signals in the circuit are not changing) and use the final solution to generate nodesets. To make the transient analysis more efficient, set the integration method to backward Euler if possible and eliminate the local-truncation error criterion by setting the option `lv1tim=1` (this is the *only* time you should use this option). Spectre does not allow you to disable the local-truncation error criterion, but you can de-emphasize it by increasing `lteratio`, say to 50. Occasionally, this approach fails or is very slow because the circuit is or contains an oscillator. Try to disable the oscillator before using this approach. For more information on this approach, see Section 4.4.1 on page 207.
19. The previous suggestion is simply a do-it-yourself continuation method. In Spectre, since you may sweep model parameters, you may try many other possible continuation methods. For example, in a circuit that consists of a large number of bipolar junction transistors, try starting at a high temperature and sweeping it down to the temperature of interest. Or you can replace all NPN models with one common model and perform a DC analysis while using a model parameter such as `bf`, `vaf`, or `imax` as a continuation parameter. In other words, choose some value for which it is easy to achieve DC convergence (`bf=10-6` for example), and perform a DC analysis that sweeps the parameter to its final value. Or, convert all independent sources to controlled sources dependent on a single source, and sweep

that source. Use the solution at the final step as a nodeset vector on the circuit with the original models.

20. If you cannot get the simulator to converge when computing the initial point of a transient, skip the initial point computation by specifying `uic` on the transient analysis statement.

Long amplifier chains (inverter chains or ring oscillators) may have convergence problems because their high gain nature causes the matrix package to overflow. Carefully choose an initial guess with `nodeset` statements. If the solution is in a low gain region (such as with an inverter chain), use nodesets to initially bias the circuit into the low gain region.

Many of the above suggestions require that you approach the DC solution gradually using DC sweeps or transient analyses, and once you have it, use the final solution to generate nodesets. This is a very difficult and time consuming procedure with many simulators. Spectre provides a very desirable feature from a usability perspective: the ability to automatically write final solutions to a file and read them in as a nodeset vector on any subsequent analysis (see Section A.3.2.7 on page 349).

2.3.2 DC Analysis Accuracy

In a DC or operating point analysis, there are three things that contribute error in the solution. First, the models may not be completely accurate [Jeng]. Second, the simulator may add components to the circuit that the user did not explicitly specify, such as `gmin`. Lastly, the convergence criteria contribute error because they stop Newton's method before the nonlinear equations are solved exactly.

2.3.2.1 The Minimum Conductance: `Gmin`

Most simulators add a very small conductance of `gmin` across nonlinear devices to prevent nodes from floating if the nonlinear devices are turned completely off. By default, `gmin=10-12Ω`. The manner

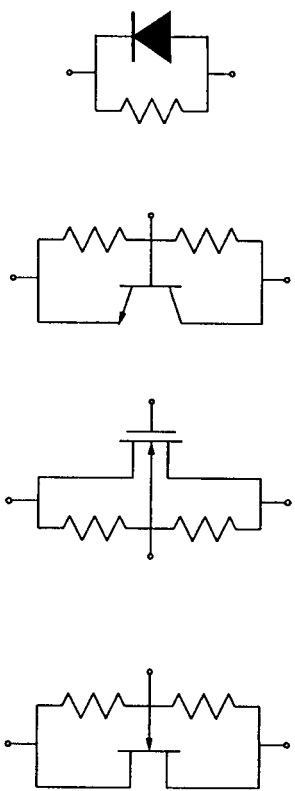


Figure 2.13: How the g_{min} conductors are added to each nonlinear device type in SPICE.

in which SPICE and Spectre add the g_{min} conductors to the various nonlinear devices is shown in Figures 2.13 and 2.14.

The g_{min} conductors affect the accuracy of the solution because they change the circuit being solved by the simulator. Most circuits are tolerant of the small currents that flow through the g_{min} conductors, however some circuits are not. For example, sample-and-hold circuits or other circuits that try to hold charge on a capacitor for a long period of time are sensitive to the small currents that flow through the g_{min} conductors. Spectre warns you if the current through the g_{min} conductors is adversely affecting the accuracy of the solution and allows you to set g_{min} to zero. However, setting g_{min} to zero may result in the simulation failing due to a singular Jacobian.

2.3.2.2 Convergence Criteria

Newton's method continues to iterate until the convergence criteria are satisfied. The first criterion (2.7) allows Kirchhoff's current law to be violated slightly. In other words, the currents at each node do not quite sum to zero. This is equivalent to connecting small current sources to every node, randomly assigning a current to each

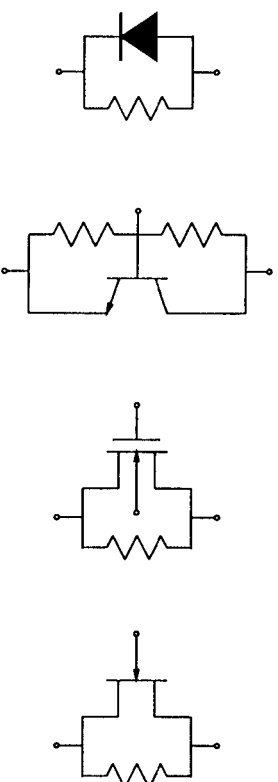


Figure 2.14: How the g_{min} conductors are added to each nonlinear device type in Spectre.

source (with the proviso that the assigned current is smaller than that allowed by the convergence criterion), and solving this modified circuit exactly. This would be a problem with high-impedance nodes because even a small current injected into a high impedance contributes a significant voltage error. However, the second convergence criterion (2.8) limits the voltage error by insisting that the voltages converge. The second criterion does not dictate the accuracy of the solution directly, because it compares the proposed solution against the value on the previous iteration, not the true solution. Thus, setting $reltol$ to 0.1% does not imply the solution is accurate to 0.1%.

2.3.3 Remedies for Accuracy Problems

In general, if you would like your simulator to compute a more accurate solution, tighten $reltol$. Also, make sure $abstol$ and $vnrtol$ are reasonable. If your circuit is sensitive to simulator errors, it is a good idea to tighten $reltol$ before you even start. In some situations tightening $reltol$ may not help, or it may slow the simulator down more than necessary. So you might also consider the following suggestions.

2.3.3.1 DC Analysis

1. First assume that the simulator has computed the correct solution to the wrong circuit. Use your knowledge of the circuit to debug it using the computed DC solution and the operating point. Look for errors in the topology, the component parameters, the models, or the power supplies. The simulator often gives clues in the form of warning and error messages.
2. Assume that models you are using are appropriate and that the model parameters are consistent and correct.
3. Consider that your circuit might have more than one solution, which is fairly common, and the simulator found one you did not anticipate. Try using `nodeset` statements to encourage the simulator to compute the solution you desire.
4. Tighten `reltol`. Also make sure `abstol` and `vntol` are reasonable.
5. Assume `gmin` is not affecting the solution. If possible, set `gmin` to zero.

2.4 Applications of DC Analysis

This last section discusses circuit issues that arise when using DC analysis. The discussion starts with an example of a common circuit that exhibits multiple operating points and ends by presenting suggestions on how approach the simulation of large circuits.

2.4.1 Circuits with Multiple Operating Points

Consider the supply-independent current reference of Figure 2.15 on the next page [Gray84]. The transfer function of the circuit consisting of Q_1 , Q_2 , and R is shown in Figure 2.16 as a solid line. The transfer function of the current mirror composed of Q_4 and Q_5 is shown with a dashed line. The operating point of the circuit falls at the intersection of the two curves. Since there are two intersections,

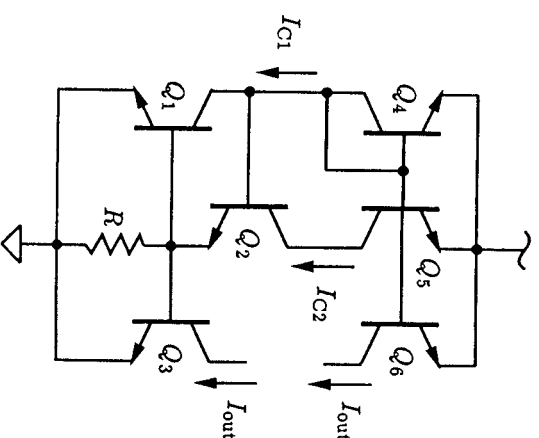


Figure 2.15: Supply independent current reference that has startup problems because of multiple solutions.

there are two operating points. The operating point far from the origin is the desired one. Normally, the operating point at the origin is unstable because of positive feedback that acts to drive the circuit away from this state. In practice, the operating point near the origin tends to be stable because the very low currents at the origin act to reduce the current gain of the transistors, often to the point where the circuit is unable to drive itself away from the origin. A startup circuit that eliminates the undesirable operating point or assures that it is unstable is needed for this circuit to reliably perform as intended.

Regardless of whether the operating point at the origin is stable, it does represent a solution of the DC circuit equations. The simulator is likely to compute and report on either operating point. If a

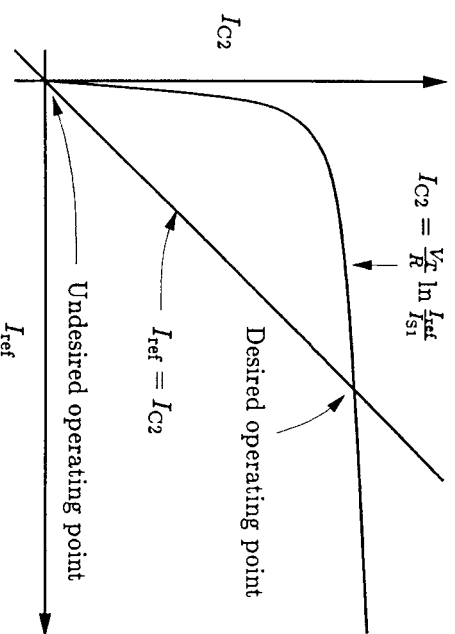


Figure 2.16: The supply independent current reference of Figure 2.15 on the page before has two operating points. The one at the origin is undesirable because it results in no current flowing in the outputs.

simulator reports on an undesirable operating point, such as the one near the origin, you should try to determine whether it is stable or unstable by simulating the circuit with a transient analysis. Perturb the circuit slightly and use the techniques for nurturing oscillations described in Section 4.4.2 on page 209. If the circuit drifts away from the undesired operating point due to the small perturbation, then it is likely unstable. If it returns, it is likely stable. Using a perturbation that is too large may result in the circuit drifting away from a stable operating point.

You can use `nodesets` to encourage the simulator to find an operating point if you know where it is. However, there is no reliable approach usable with currently available commercial simulators to find unexpected multiple operating points.

2.4.2 Large Circuits

Large circuits pose additional challenges to both the simulator and the user. Of course simulation time is greater on large circuits, but you will probably also find that DC convergence is harder to achieve. Once a simulator has converged, the size of the circuit makes it difficult to determine if the results are correct. A quick way to find some problems on large circuits is to inspect composite figures of merit such as power dissipation or supply currents. For example, when evaluating Spectre one large semiconductor company gave us a netlist for a 1000 transistor circuit that they had already simulated with their own simulator. The designers were convinced that the circuit worked properly and so had submitted it for fabrication. When Spectre simulated the circuit, it emitted a warning indicating that two transistors had unusually high collector currents. A closer look at those transistors showed that the transistors were configured as diodes that were shorting the 5 V power supply to ground. When the designers went back to the original simulations, they found that the fault had existed all along and they had simply not noticed it. Had they glanced at the power dissipation, which was greater than 100 W, they would have been alerted to the problem, though they would still need to find it.

Spectre provides another tool for finding problems in large circuits. The info statement prints information about the circuit and its components. Based on the `what` parameter, the info statement prints the following information:

input

Prints the value of each parameter on every component in the circuit. For example, the resistance specified for each resistor, etc.

output

Prints computed quantities associated with every component that *do not* depend on the operating point. For example, the true channel length of a MOSFET, which equals the length as drawn minus the encroachment of the source and drain from lateral diffusion. Another example is the effective resistance of a temperature dependent resistor at the current temperature.

oppoint

Prints computed quantities associated with every component that *do* depend on the operating point. For example, a bipolar transistor may compute the terminal currents, power dissipation, and the small signal model. It also prints out the node voltages and selected branch currents.

More interesting for large circuits is that the **info** statement can be directed to print only the extreme values. The extreme values include the most positive, the most negative, and the closest to zero. In other words, rather than printing the resistance for every resistor, the **info** statement prints only the most positive resistance, most negative resistance, and the resistance with the smallest absolute value. In addition, it also prints the name of the resistor where the extreme value was found. Extreme values can be printed for input, output, and operating point parameters.

When simulating large circuits, it is helpful to use the **info** statement to print the extreme input, output and operating point parameters to make sure everything is reasonable. One important advantage to using the extreme value feature of the **info** statement rather than just inspecting the power dissipation and supply currents is that the **info** statement points you directly to the components with the unusual conditions.

To further automate the process when validating large circuits, Spectre provides soft parameter limits and the **check** statement. Soft parameter limits are user specifiable ranges for parameter values that when violated cause Spectre to generate a warning message. For example, in practice, very little DC current should flow into the bulk terminal of a MOSFET. If significant current does flow, it is generally due to an error, such as a when a P-channel MOS is mistakenly used instead of an N-channel MOS. As shipped from Cadence, Spectre is configured to assume that $|I_{\text{bulk}}| < 1 \text{ nA}$ is normal, and any current outside that range will generate a warning. If you routinely work with power MOSFETs, then you might want to change the 1 nA limit to 1 μA to account for the larger leakage current.

Soft limits can be placed on any input, output, or operating point

parameter. If Spectre is directed to use the soft limits, then it checks all input and output parameters after the circuit has been read in and before any analyses are run. It also checks the operating point parameters after it performs a DC analysis. You can use the **check** statement to tell Spectre to check the soft limits at other times, such as after a set of parameters have been altered, or after a transient analysis.

2.4.3 Restarting from a Previous Solution

One way that Spectre differs from SPICE is that in SPICE you can only run one analysis of each type and the analyses are run in a fixed order. In Spectre, you can run an analysis any number of times and the analyses are run in the order you specify them. The benefit is that with Spectre one can run a few analyses, make small modifications to the circuit, run more analyses, etc, until a circuit is completely characterized. However, to get the best performance, you should carefully choose the order in which you specify the analyses. The goal is to minimize the effort Spectre must put into recomputing the operating point.

Consider the situation where you would like to calculate the DC operating point of a circuit, perform a transient analysis, and compute the frequency response. The frequency response is computed at the DC operating point and does not modify the operating point. Transient analyses generally start from the operating point, but does modify the operating point. Thus, the best order would be to compute the operating point, compute the frequency response, and then perform the transient analysis. If the frequency response were computed after the transient analysis, the operating point would have to be recomputed. Furthermore, since the final point of the transient analysis may be far from the operating point, computing the operating point starting from the final transient point is often more expensive than starting fresh.

Spectre internally keeps the most recently computed solution and makes it available as a starting point for further computation. The next analysis starts from it automatically if no parameter that affects the DC solution has changed value. If such a parameter has changed

value, then normally Spectre would not start from this saved solution unless explicitly told to do so by specifying `restart=no` on the subsequent analysis.

There are two different ways to cause Spectre to restart from a previous solution. First, if the solution you want to start from is the solution from the last step of the previous analysis, use `restart=no`. An example of where that might be useful is if you were having convergence difficulties at high temperature using:

```
SetTempTo80 alter param=temp value=80
OpAt80 dc
```

Instead, you could compute the solution at room temperature, sweep temperature to the higher temperature, and compute the new solution using `restart=no`.

```
OpAt27 dc
SweepTemp dc param=temp start=27 stop=80 lin=80
SetTempTo80 alter param=temp value=80
OpAt80 dc restart=no
```

Since `SetTempTo80` changes the temperature, which does affect the operating point, the previously computed solution would not normally be used by `OpAt80`. The `restart=no` tells `OpAt80` to start from the solution computed at the final step of `SweepTemp`. A `restart=no` is not needed on the `SweepTemp` analysis, because nothing changes from `OpAt27`, and so Spectre automatically starts from the previous solution.

The second situation is when you want to restart from something other than the last point of the previous analysis. In this situation, you must write the solution to a file when you first compute it, and recall it later when it is needed. Consider the case where you would like to perform two swept DC analyses

```
OpAt27 dc
SweepTemp dc param=temp start=27 stop=80 lin=80
SweepVcc dc param=dc dev=Vcc start=3 stop=7 lin=80
```

As before, a `restart=no` is not needed on the `SweepTemp` analysis, because nothing has changed from `OpAt27`. However, Spectre resets the temperature to 27 at the end of `SweepTemp`. Because of this, `SweepVcc` starts from scratch, whereas it would be faster if it started from the solution computed by `OpAt27`. That is arranged as follows:

```
OpAt27 dc write="OpAt27"
SweepTemp dc param=temp start=27 stop=80 lin=80
SweepVcc dc param=dc dev=Vcc start=3 stop=7 lin=80 \
    reads="OpAt27"
```

After Spectre computes the dc solution for `OpAt27`, it writes it to the file "OpAt27". Before starting `SweepVcc`, Spectre reads file "OpAt27" and uses its contents as a nodeset.

Since Spectre saves the operating point in a file, it is possible to use the data on later invocations of Spectre. In this case, adding `reads="OpAt27"` to `OpAt27` causes it to start from the contents of file "OpAt27" if it exists. This generally results in Spectre starting up faster, even if the circuit has been changed somewhat.

2.5 Summary

This is a list of the key points presented in this chapter along with the section and page numbers where they were presented.

- Circuits often have more than one DC solution and any DC solution computed by the circuit simulator may be unstable. Section 2.2 on page 17.
- Newton's method is guaranteed to converge if the equations are continuously differentiable, if the solution is isolated, and if the initial starting point is sufficiently close to the solution. In circuit simulation, none of these three conditions are guaranteed, and so neither is convergence. Section 2.2.1 on page 18.
- SPICE-like simulators do not assure that Kirchhoff's current law is satisfied, and so are subject to false convergence. Section 2.2.2.1 on page 21.

- Errors in specifying circuit connectivity, component values, or model parameter values often cause convergence problems. Section 2.2.3 on page 24.
- Very small floating resistors can cause convergence difficulties. These generally occur as parasitic resistors in semiconductors or as current-sense resistors. Either eliminate the small resistor, or increase `abstol`. Section 2.2.3 on page 24.
- Newton's method requires that the solution be isolated. The topology checker will alert you to many, but not all situations that generate nonisolated solutions. Section 2.2.3.1 on page 25.
- The starting point for Newton's method is specified using the `nodeset` statement. Specifying the starting point can avoid convergence problems or can encourage the simulator to find a particular solution when more than one exists. Section 2.2.3.2 on page 27.
- Continuation methods are employed by the simulator to overcome convergence problems. Section 2.2.3.5 on page 29.
- Suggested remedies for convergence problems that occur while computing a DC operating point or the initial time initial condition for a transient analysis are given in Section 2.3.1 on page 35. Suggested remedies for accuracy problems that occur while computing a DC operating point or the initial time initial condition for a transient analysis are given in Section 2.3.3 on page 41.

Chapter 3

AC Analyses

3.1 Introduction

The AC analyses are a family of frequency-domain analyses that include AC analysis, transfer function (XF) analysis, scattering parameter (SP, TDR) analyses, and noise analysis. All of these analyses are based on the same underlying mathematical technique: phasor analysis [chu87]. As such, they all share common assumptions and characteristics. Phasor analysis computes the small-signal sinusoidal steady-state response of the circuit. This implies that the solutions computed by the AC analysis contain only sinusoids at the same frequency as the input signal. Thus, each signal is represented with only two numbers, one that gives the magnitude, and another that gives the phase. With such a simple representation for a signal, it is not possible to represent transient behavior. Indeed, the AC analyses do not compute the transient response of a network. Thus, unlike with transient analysis, it is difficult to judge the stability of a circuit by a simple inspection of the results. For example, if one used AC analysis to compute the response of a simple RC circuit, and then made the circuit unstable by replacing the R with $-R$, the response of the two circuits would be the same, except for the phase, which would be negated.

AC analysis computes the small-signal behavior of a circuit by first linearizing the circuit about a DC operating point. Since the AC analyses operate on a linear time-invariant representation, the re-

sults computed by the AC analyses cannot exhibit the effects normally associated with nonlinear and time-varying circuits: distortion and frequency translation. However, the AC analyses do provide a wealth of information about the linearized circuit and so are invaluable in certain applications. They are also, on the whole, much less temperamental than DC or transient analysis. The AC analyses are not subject to the convergence problems of DC, and the accuracy problems of transient. If the AC analyses are inaccurate, it is almost always because the component models are incorrect.

While the mathematical basis of all of the AC analyses is the same, the results they compute are different and they are applied in different situations. Both AC and XF analyses are used to compute transfer functions. Which one you used is decided solely on the basis of efficiency. AC analysis is used if the number of independent outputs is greater than the number of independent inputs. A typical example of when AC analysis is called for is when you want to debug a circuit by applying a single stimulus and inspecting the response by probing arbitrary node voltages and terminal currents at will. XF analysis is used if the number of independent inputs is greater than the number of independent outputs. An example of when XF analysis is preferred is when you want to compute several transfer functions, such as differential- and common-mode gains as well as positive and negative power supply rejection, to a single output.

S-parameter analysis is used to create an S-parameter N-port description of the circuit. This analysis is most commonly used on high-frequency analog circuits. Time-domain reflectometry (TDR) analysis is the time-domain equivalent of S-parameter analysis. Converting S-parameters into the time domain sometimes provides insight not available from frequency-domain results.

Finally, noise analysis is used to predict the steady-state response of the linear time-invariant circuit to the small random noise signals generated by the components. It does so in the frequency-domain where such calculations are very efficient.

3.2 AC Analyses Theory

One question engineers often ask about their circuits is “What is its frequency response?” or “What is its transfer function?”. This is its frequency response?” or “What is its transfer function?”. This is equivalent to asking “How will it respond if I stimulate it with a small sinusoidal signal?”. To answer such a question, circuit simulators formulate a new set of equations by linearizing the circuit about the DC solution using a Taylor series expansion and solving them using phasor analysis. Given a circuit described with (1.3) with a DC operating point of v_{DC} , the Taylor series expansion is used to calculate the change in the solution (Δv) as a function of the change in the stimulus (Δu). The Taylor series expansion for

$$f(v) + u = 0 \quad (3.1)$$

is

$$f(v + \Delta v) + u + \Delta u = f(v) + u + \Delta u + \frac{\partial f}{\partial v} \Delta v + \frac{\partial^2 f}{\partial v^2} \Delta v^2 + \dots \quad (3.2)$$

Since $v + \Delta v$ is the solution to the perturbed circuit,

$$f(v + \Delta v) + u + \Delta u = 0. \quad (3.3)$$

The equation is simplified by substituting (3.1) and (3.3) to (3.2) and by assuming the perturbation in the stimulus (Δu) is small, which makes the higher order derivatives negligible.

$$\frac{\partial f}{\partial v} \Delta v + \Delta u = 0 \quad (3.4)$$

Applying these arguments to the circuit equation (1.3) and replacing $f(v)$ with $i(v) + \frac{d}{dt}q(v(t))$ results in

$$\frac{\partial}{\partial v} i(v_{DC}) \Delta v(t) + \frac{\partial}{\partial v} \frac{d}{dt} q(v_{DC}) \Delta v(t) + \Delta u = 0 \quad (3.5)$$

Let

$$G = \frac{\partial}{\partial v} i(v_{DC}) \quad (3.6)$$

be the small-signal conductance at the operating point, and

$$C = \frac{\partial}{\partial v} q(v_{DC}) \quad (3.7)$$

be the small-signal capacitance. Then (3.5) becomes

$$G\Delta v(t) + C\frac{d}{dt}\Delta v(t) + \Delta u = 0 \quad (3.8)$$

In phasor analysis one computes the small-signal sinusoidal steady-state solution by assuming that all signals take the form of complex exponentials. Let $\Delta v(t) = V e^{j\omega t}$ and $\Delta u(t) = U e^{j\omega t}$ where V and U are complex, then

$$GV e^{j\omega t} + C\frac{d}{dt}(V e^{j\omega t}) + U e^{j\omega t} = 0 \quad (3.9)$$

$$GV e^{j\omega t} + j\omega C(V e^{j\omega t}) + U e^{j\omega t} = 0 \quad (3.10)$$

$$GV + j\omega CV + U = 0 \quad (3.11)$$

$$(G + j\omega C)V = -U \quad (3.12)$$

Equation (3.12) is a complex linear system of equations that is solved for V . V is a complex number (or vector) that gives both the magnitude and the phase of the solution.

3.3 AC Analyses Practice

In this section, the practical details of the various AC analyses are presented.

3.3.1 AC Analysis

With AC analysis, one specifies the stimulus for the circuit and computes the sinusoidal steady-state response. The stimulus may consist of signals on several sources, but they must all be sinusoidal and have the same frequency. For example, simulating a differential amplifier often involves applying two sinusoidal sources to the inputs, one shifted 180° from the other. It is not possible to simulate a mixer because for mixers the two input signals must be at different frequencies.

Prior to performing the AC analysis, the circuit is linearized. Thus, the absolute magnitude of the stimulus loses some of its significance.

Since the circuit is linearized, large signal effects such as distortion or clipping are not modeled. Furthermore, since the circuit is linearized about a constant operating point, effects from the time-varying nature of a circuit are not modeled, such as energy being converted from one frequency to another (as in mixers). For this reason, most AC analyses are performed with the stimulus magnitude set to unity. Doing so is a convenience that results in the simulator directly computing the transfer function rather than computing the actual signal level. For example, consider the simulation of an open-loop opamp with a low frequency gain of 100 kV/V using AC analysis. By specifying the stimulus magnitude to be one and stimulus phase to be zero, the simulator directly computes the gain of the opamp.

3.3.2 XF Analysis

Spectre provides an AC-like analysis that is referred to as the transfer function or XF analysis. Like AC analysis, it is a small-signal analysis (in other words, it operates on the linearized circuit equations) that is based on phasors. Unlike AC analysis, which allows you to simultaneously compute the transfer function from a single stimulus to every node in the circuit, XF analysis simultaneously computes individual transfer functions from every independent source to a single output. Consider the amplifier shown in Figure 3.1 on the following page. Typically, a designer might be interested in transfer functions between several sources in the circuit to the output. For example, the gain is the transfer function from V_{in} to $V(out)$. The positive power supply rejection is the inverse of the transfer function from V_{cc} to $V(out)$. Similarly the negative power supply rejection is the inverse of the transfer function from V_{ee} to $V(out)$. Finally, the output impedance is the transfer function from I_{out} to $V(out)$. The XF analysis computes all of these transfer functions at once.

By default Spectre computes the transfer function from every independent source in the circuit to the chosen output. However, it is also possible to have Spectre compute all possible transfer functions. This is useful when it is not known in advance which transfer functions are interesting. Transfer functions for nodes are computed assuming that a unit magnitude current source is connected from

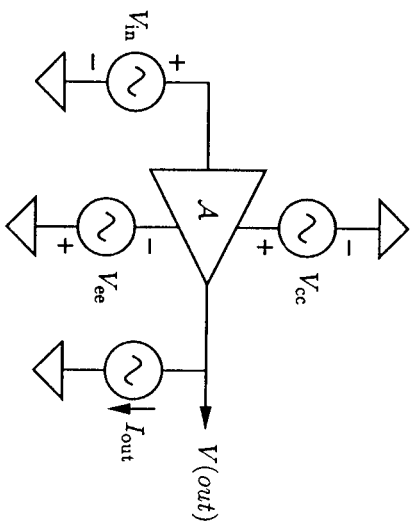


Figure 3.1: Xf analysis simultaneously computes the transfer function from all sources in the circuit to the output. In this circuit, it allows the gain, the power supply rejection from both supplies, and the output impedance to be computed with one analysis.

the node to ground. Transfer functions for terminals are computed assuming that a unit magnitude voltage source is connected in series with the terminal.

3.3.3 SP Analysis

S -parameters are used by high-frequency circuit designers to characterize linear or near-linear circuits and components. Spectre's `sp` analysis directly computes S -parameters in the frequency domain and its `tdr` analysis directly computes them in the time domain. Both analyses are small-signal analyses based on phasor analysis, and so both start by linearizing the circuit about the DC operating point. To measure S -parameters the circuit must be configured as an N -port. Spectre provides a port component that is used to define the ports of the N -port. Every port component defines one

port of the network. A port component acts like a voltage source in series with a resistor. The value of the resistor is referred to as the reference resistance of the port. The default value is $50\ \Omega$, though it can be set to any value. When the S -parameters are computed as a function of frequency, they may be saved for plotting and they may be written to an ASCII file that is readable by Spectre's `nport` component. In this way, the characteristics of a linear component or circuit are encapsulated in a file by the `sp` analysis, and can be read into a different larger circuit by the `nport` component and analyzed using any analysis supported by Spectre, including transient analysis.

It is possible to compute S -parameters as a function of frequency with SPICE, though it takes some work. Start by modeling each port by a resistor in series with a voltage source. The value of the resistor is set to the reference resistance of the port, and the AC magnitude of the voltage sources is initially set to zero on all ports. Assume for simplicity that the S -parameters for a two port are being computed. To compute s_{11} and s_{21} , set the AC magnitude of the voltage source on the first port to 2 V and run an AC analysis over the frequency range of interest. Then, s_{11} is equal to the voltage across port one minus 1, and s_{21} is equal to the voltage across port two. To compute s_{12} and s_{22} , return the AC magnitude on the first port to zero and set the AC magnitude of the voltage source on the second port to 2 V. Again, run AC analysis. s_{22} is equal to the voltage across port two minus 1, and s_{12} is equal to the voltage across port one. This procedure is diagrammed in Figure 3.2 on the next page. For clarity, the disabled voltage sources have been removed from the figure. To avoid the subtraction by one when computing s_{11} and s_{22} , install 1 V voltage sources in the circuit to automatically perform the subtraction.

3.3.4 Noise Analysis

Noise analysis predicts the noise performance of a linearized circuit in the frequency domain. The noise is caused by stochastic fluctuations in certain types of components. Examples of noise sources in components include thermal noise in resistors (also referred to as

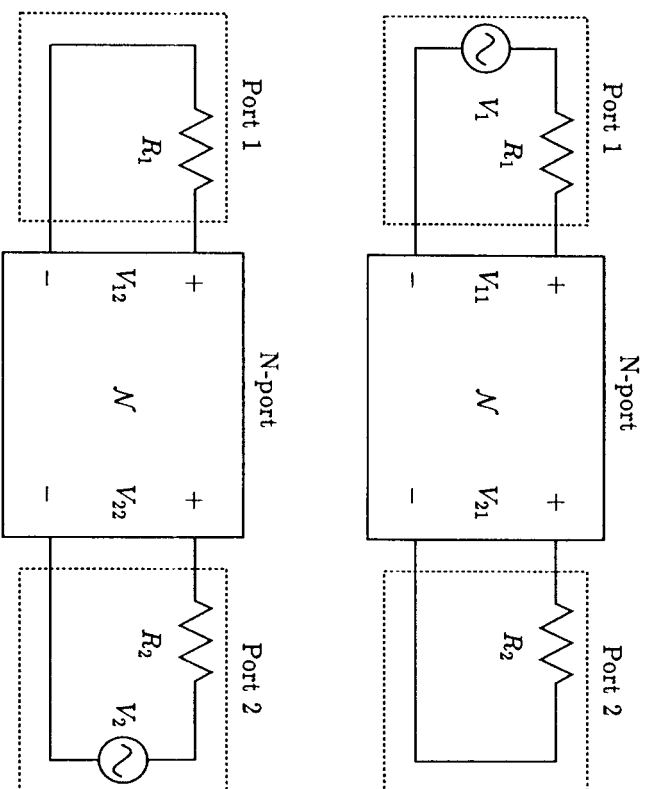


Figure 3.2: Illustration of how S -parameters are computed with SPICE. The top circuit computes $s_{11} = V_{11} - 1$ and $s_{21} = V_{21}$, where $V_1 = 2$ V. The bottom circuit computes $s_{22} = V_{22} - 1$ and $s_{12} = V_{12}$, where $V_2 = 2$ V.

Johnson noise or Nyquist noise), shot noise in semiconductor junctions, and flicker or $1/f$ noise in a variety of components [gray84]. In all cases the distribution of the noise is assumed to be Gaussian.

The noise is computed as a spectral or noise density or spot noise. When referring to voltage noise, the noise density is specified as $\overline{v^2}/\Delta f$ with units of Volts squared per Hertz. When referring to current noise, the noise density is specified as $\overline{i^2}/\Delta f$ with units of Amperes squared per Hertz. Thermal noise and shot noise are considered to be ‘white’ noise processes, which implies that the spectral density is flat with frequency and the noise is not self correlated. The spectral density of flicker noise is inversely proportional to the frequency raised to some, possibly non-integer, power usually near one.

$$\overline{n^2(f)} \sim \frac{1}{f^\alpha} \quad (3.13)$$

where typically $0.5 < \alpha < 2$. Noise is considered to have a ‘pink’ spectrum if $\alpha = 1$ and to have a ‘red’ spectrum if $\alpha = 2$. When embedded in a circuit, the circuit further ‘colors’ the noise.

The noise analysis computes the individual contribution of every noise source in the circuit to the output noise density. It also computes the composite noise density. All noise sources are assumed to be independent Gaussian noise processes with a given spectral density and random phase. As such, the composite output noise is computed as the *mean-square* sum of the contributions from each noise source individually.

3.3.4.1 Noise of Common Components

This section describes simple noise models for common components. Your simulator generally uses more detailed models.

Resistor The thermal noise of resistors is given as either a noise voltage in series with the resistor

$$\overline{v^2} = 4kTR\Delta f \quad (3.14)$$

or as a noise current in parallel with the resistor

$$\overline{i^2} = 4kTG\Delta f \quad (3.15)$$

where $k = 1.3806226 \times 10^{-23}$ is Boltzmann's constant, T is the absolute temperature in Kelvins, R is the resistance of the resistor, and $G = 1/R$ is its conductance. Carbon resistors also exhibit flicker noise if the resistor carries a DC current.

Junction Diode The noise model for an intrinsic semiconductor junction is given as a noise current

$$\overline{i^2} = \underbrace{2qI_f\Delta f}_{\text{shot noise}} + K_f \underbrace{\frac{I_f^{\alpha_f}}{f}}_{\text{flicker noise}} \Delta f \quad (3.16)$$

in parallel with the junction. This equation predicts the noise of a diode if the noise model for the series resistor is added.

BJT Similarly, the noise model of the intrinsic BJT is

$$\overline{i_c^2} = \underbrace{2qI_c\Delta f}_{\text{shot noise}} \quad (3.17)$$

$$\overline{i_b^2} = \underbrace{2qI_b\Delta f}_{\text{shot noise}} + K_f \underbrace{\frac{I_b^{\alpha_f}}{f}}_{\text{flicker noise}} \Delta f \quad (3.18)$$

Again, it is necessary to add noise models for the parasitic resistors, particularly the for the base resistor.

JFET The noise model for an intrinsic JFET is given with a shot current noise generator connected from the gate to the source, and thermal and flicker noise current generators connected from drain to source

$$\overline{i_g^2} = \underbrace{2qI_g\Delta f}_{\text{shot noise}} \quad (3.19)$$

$$\overline{i_d^2} = 4kT \underbrace{\left(\frac{2}{3}g_m\right)\Delta f}_{\text{thermal noise}} + K_f \underbrace{\frac{I_d^{\alpha_f}}{f}}_{\text{flicker noise}} \Delta f \quad (3.20)$$

where g_m is the transconductance of at the operating point. This model is applicable only when the device is operating in the saturation region. One can also add noise models for the parasitic resistors at the terminals (including the gate).

MOSFET Finally, the noise model for an intrinsic MOSFET is similar to the model for the JFET, except the insulated gate eliminates the need for the shot noise model.

$$\overline{i_d^2} = 4kT \underbrace{\left(\frac{2}{3}g_m\right)\Delta f}_{\text{thermal noise}} + K_f \underbrace{\frac{I_d^{\alpha_f}}{f}}_{\text{flicker noise}} \Delta f \quad (3.21)$$

Again, this model only applies to the saturation region, and does not include noise due to the parasitic resistors.

3.3.4.2 Total Noise

To compute the total noise over a range of frequencies, one must first specify the frequency response of the measurement itself (noise is always assumed to be measured over a finite bandwidth, because the total noise power of any white noise process over an infinite bandwidth is infinite). Assume that frequency response of the measurement is given by $H(s)$ where $H(s) \rightarrow 0$ as $s \rightarrow \infty$. Then the total noise is calculated by integrating the product of the noise spectral density ($S(f)$) and the magnitude-squared of the measurement transfer function $H(j2\pi f)$.

$$\overline{v_T^2} = \int_0^\infty |H^2(j2\pi f)S(f)|df. \quad (3.22)$$

It is most convenient to compute the total noise while assuming that $H(j2\pi f)$ is unity over a range of frequencies and zero otherwise. To measure the total noise in this situation, simply perform the noise analysis while sweeping the frequency over the given range. Then

the total noise is simply the sum over each frequency

$$\overline{v_T^2} = \sum_f S(f) \Delta f \quad (3.23)$$

where Δf is assumed small. Be sure to perform the computation using the mean-square noise density (measured in Volts or Amperes squared per Hertz) for $S(f)$ rather than the root-mean-square noise.

If a non-flat measurement frequency response is desired, a noiseless filter could be synthesized that implements the desired shape and placed after the circuit under test, but before the noise probe.

3.3.4.3 Input Referred Noise

In some cases it is more useful to model a noisy two port with a single noise source at the input. Doing so allows one to easily compare the noise generated by the circuit to its expected input signals.

To compute the spectral density of this equivalent input referred noise, simply divide the output noise density by the square of the two-port gain at each frequency (the simulator can usually be configured to do this for you). The interpretation of the input referred noise is such that the total noise density at the output of a noisy two port is the same as the noise at the output of a noiseless, but otherwise equivalent, two port with the input referred noise applied to the input.

3.3.4.4 Noise Figure

The noise figure is a commonly used method of specifying the noise performance of a circuit or a component. It is popular in communication systems.

The *noise factor* F is defined as [motchenbacher73]

$$F = \frac{\text{input } S/N \text{ ratio}}{\text{output } S/N \text{ ratio}} \quad (3.24)$$

It can be rewritten as

$$F = \frac{\text{output noise power}}{\text{noise of source resistance} \times \text{gain}^2} \quad (3.25)$$

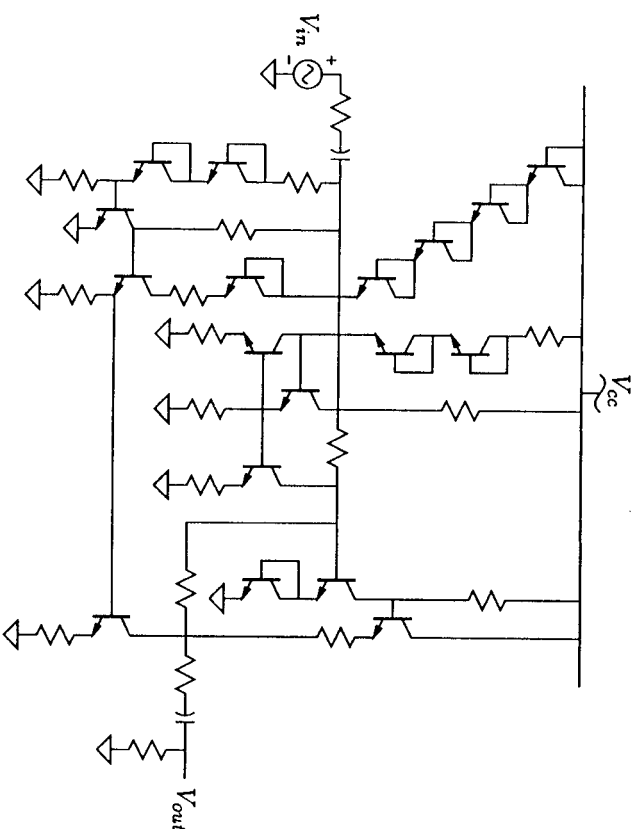


Figure 3.3: Class AB power amplifier of Meyer and Mack [meyer89].

It is common to specify noise factor in decibels, in which case it is referred to as the *noise figure*

$$NF = 10 \log_{10} F \quad (3.26)$$

Example For the circuit shown in Figure 3.3, the total output spot noise at 100 MHz, including a contribution from the source resistance, is 107 aV²/Hz. The output spot noise due only to the source resistance is 7.71 aV²/Hz. The spot noise figure is

$$NF = 10 \log_{10} \frac{1.07 \times 10^{-16}}{7.71 \times 10^{-18}} = 11.4 \text{ dB} \quad (3.27)$$

Another way to compute the noise figure is to divide the total output noise by the product of the noise of the source ($0.83 \text{ aV}^2/\text{Hz}$) and the square of the gain (3.05).

$$NF = 10 \log_{10} \frac{1.07 \times 10^{-16}}{(3.28 \times 10^{-19})(3.05)^2} = 11.4 \text{ dB} \quad (3.28)$$

One thing to be careful of with this calculation is to use the right gain. Normally one thinks of the gain being measured from the input of the amplifier to its output. However, in this case the gain must be measured from the noise source to the output. In the case where the amplifier significantly loads the input source these gains can be quite different. In this case, the source impedance is 50Ω , and the input impedance of the amplifier is also about 50Ω , so there is factor of two difference between the gain of the amplifier ($A_v = 6.28$) and the gain used in Equation 3.28.¹

3.3.4.5 Modeling Noisy and Noiseless Components

In macromodeling, it is sometimes desired to use resistors that are noiseless. This is easily done using voltage-controlled sources and connecting the inputs directly to the outputs. Two noiseless resistors implemented using Spectre's parameterized subcircuits are shown in Netlist 3.1.

To create a noisy voltage or current source, use a controlled voltage or current source whose input is only connected to a noisy resistor as shown in Figure 3.4 on the next page. With SPICE, one is limited to white thermal noise. Red noise is implemented easily by paralleling the resistor with a capacitor. Implementing pink or $1/f$ noise is much more difficult because it requires a one-half pole roll-off, which would have to be implemented with a long RC ladder network. With Spectre, one uses the flicker noise of the resistor to generate pink noise. To do so, use a current source to bias the resistor so that it creates flicker noise. Put a voltage source in series with the resistor

¹In their paper, Meyer and Mack indicate that the measured gain was 20 dB, as was the gain predicted by SPICE. However, on the netlist I have, both SPICE and Spectre predict that the gain is about 14 dB. I suspect that the discrepancy results from a difference between the netlist used for their final simulations and the netlist given to me.

```
// Subcircuits that implement noiseless resistors
simulator lang=spectre

subckt noiseless_resistor1 (p n)
  parameters r=1
  G1 (p n p n) vccs gm=1/r
ends noiseless_resistor1

subckt noiseless_resistor2 (p n)
  parameters r=1
  Vt (p i) vsource
  F1 (i n) ccvs rm=r probe=Vt
ends noiseless_resistor2
```

Netlist 3.1: Parameterized subcircuits that implement noiseless resistors. The first implements $I = V/R$. It is preferred in most cases because it runs slightly faster in most simulators than does the second. The second, which implements $V = IR$ is best when R is very small.

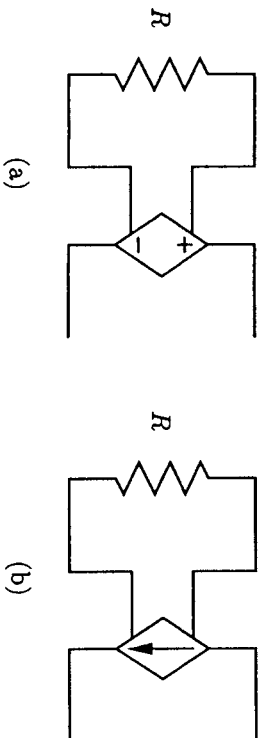


Figure 3.4: Noisy voltage and current sources are implemented by combining a noisy resistor with a controlled voltage or current source.

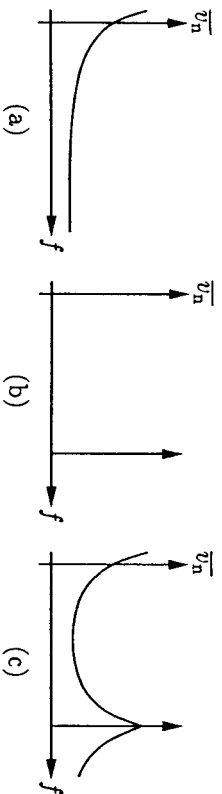


Figure 3.5: Phase noise of oscillator. The noise of the quiescent oscillator (a), the output of a noiseless oscillator (b), and the output of a noisy oscillator (c). The skirt near the carrier is the phase noise. It results from the flicker noise be mixed up by the oscillation.

to subtract off the bias voltage. As an alternative to using resistors to implement noisy voltage and current sources in Spectre, one can also create a file that describes the desired noise density as a function of frequency, and use it with the voltage source, current source, or port components.

3.3.4.6 Limitations of Noise Analysis

The noise analysis is performed on the circuit as linearized about the DC operating point. This is an approximation that is not accurate in all situations. For example, the noise performance of mixers, switched-capacitor and switched-current filters, sample-and-holds, ADCs and DACs, samplers, and oscillators cannot accurately be predicted using this type of noise analysis. All these circuits exhibit a large amount of frequency conversion when operating normally, and the linear time invariant circuit used during the noise analysis cannot exhibit frequency conversion. Consider an oscillator. When it is not oscillating, standard noise analysis accurately predicts the noise behavior, which might look like Figure 3.5a. The increase in noise at low frequencies is due to flicker noise in the active devices. When oscillating, the noise free output signal might look like Figure 3.5b. However, the noisy output of the oscillator, shown in Figure 3.5c, is

not the simple sum of the first two. The flicker noise is modulated by the oscillation, which causes it to be mixed up to the fundamental frequency and harmonics of the oscillation. This forms a noise skirt on the fundamental frequency that is referred to as phase noise. It is *not* possible to compute phase noise with the standard noise analysis in SPICE and Spectre because it linearizes the circuit about a constant operating point, and so the modulation of the flicker noise by the carrier is not modeled.

Even with simple amplifiers, the assumption of a linear time-invariant circuit for noise analysis is sometimes inaccurate. Consider the noise generated in the tail current of a differential pair. If the noise analysis is performed with the differential pair biased such that it is balanced, then the effect of the tail current noise is lost on the differential output noise because it acts as a common-mode input and so is rejected by the differential pair. However, during normal operation an input signal is applied and so most of the time the differential pair is not balanced. In this case the tail current noise does affect the output. As before, this effect cannot be modeled with the standard noise analysis.

3.4 Applications of the AC Analyses

This section discusses applications of the AC analyses. It starts with a comprehensive presentation of the difficult topic of characterizing feedback systems and finishes with a description of how to characterize differential amplifiers.

3.4.1 Characterizing Feedback Amplifiers

There are four important parameters that characterize a feedback system: open-loop gain, closed-loop gain, loop gain, and feedback factor. These feedback parameters are used to understand how the feedback affects the performance of the base amplifier in terms of the distortion, stability, noise, input and output impedance,² and sensi-

²Impedance is an abbreviated way of referring to both impedance and admittance.

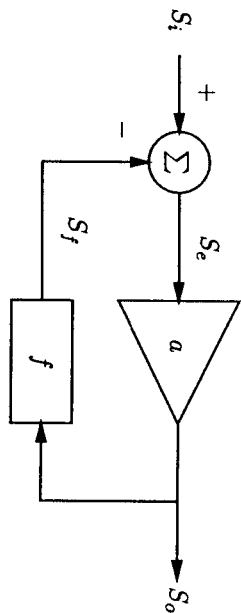


Figure 3.6: Ideal feedback amplifier. Blocks are assumed to be unidirectional and such that they do not load each other.

tivity to changes in components. One must be able to accurately measure these parameters to predict the performance of a feedback amplifier. Unfortunately, when non-ideal components are combined to form a feedback system, the feedback parameters become quite difficult to measure. This section starts by describing the four feedback parameters and then gives a number of ways that they can be measured that are appropriate for feedback systems made up of ideal blocks. Then, the interpretation of the feedback parameters in the presence of loading and bilateral coupling is described. These effects act to degrade the accuracy of the simple measurements techniques that are described first. Finally, more sophisticated measurements techniques are given that are accurate in the presence of loading and bilateral coupling.

3.4.1.1 Feedback Systems Constructed from Ideal Blocks

Consider the idealized feedback system shown in Figure 3.6. Each block is assumed to be unidirectional and such that it does not load the other blocks. S_i and S_o are the input and output signals of the system and may be either voltages or currents. The input signal is subtracted from the feedback signal S_f to form the error signal S_e . The error signal passes through the amplifier where it is multiplied by a , the *open-loop gain*, to form S_o . Finally, the output signal passes

3.4. Applications of the AC Analyses

69

through the feedback network where it is multiplied by f , the *feedback factor*, to form the feedback signal S_f .

With relatively simple manipulations it is easy to show that the gain of the feedback system, also referred to as the *closed-loop gain* A , is

$$A = \frac{S_o}{S_i} = \frac{a}{1 + af} \quad (3.29)$$

It is also useful to define product of the gains around the loop as the *loop gain* T , where

$$T = \frac{S_f}{S_e} = af \quad (3.30)$$

The loop gain is used when studying the effect of feedback on imperfections such as finite input and output impedance, distortion, and noise. More information on this topic is found in most texts on analog circuit design [gray84].

In summary, the four parameters used to characterize feedback systems are:

$$A = S_o/S_i \text{ — closed-loop gain} \quad (3.31)$$

$$a = S_o/S_e \text{ — open-loop gain} \quad (3.32)$$

$$T = S_f/S_e \text{ — loop gain} \quad (3.33)$$

$$f = S_f/S_o \text{ — feedback factor} \quad (3.34)$$

3.4.1.2 Feedback Configurations

As mentioned before, the input and output signals may be either voltage or current. In practice, designers distinguish between the four possible configurations, which are shown in Figure 3.7 on the next page.

Series-Shunt Feedback A feedback amplifier that has voltage as both the input and output is referred to as being in the series-shunt configuration because the feedback network must be connected in series with the input to feedback a voltage and in shunt with the output to sense a voltage. If the blocks are near ideal (unilateral

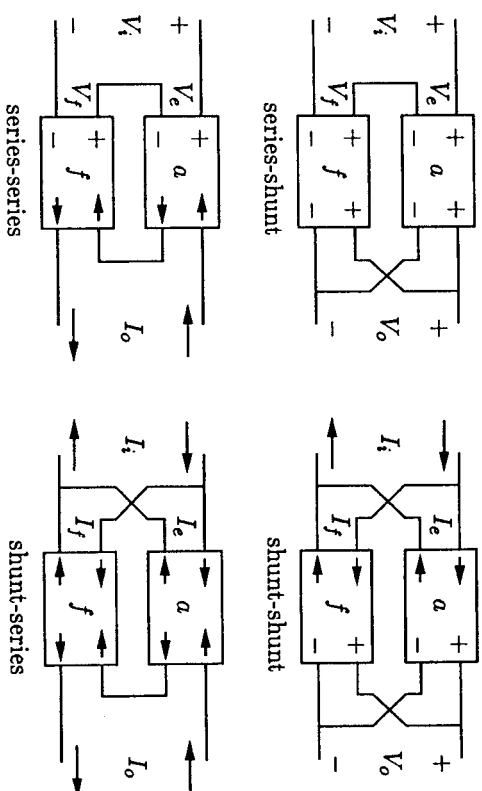


Figure 3.7: The four possible feedback configurations.

blocks, no loading, and no common-mode gain) then the following formulas directly compute the feedback parameters.

$$A = V_o/V_i \quad (3.35)$$

$$a = V_o/V_e \quad (3.36)$$

$$T = V_f/V_e \quad (3.37)$$

$$f = V_f/V_o \quad (3.38)$$

A typical example of a series-shunt feedback amplifier constructed from an opamp and two resistors is shown in Figure 3.8 on the facing page.

Shunt-Shunt Feedback A feedback amplifier that has current as an input and voltage as an output is referred to as being in the shunt-shunt configuration because the feedback network must be connected

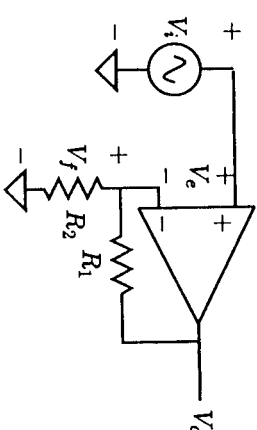


Figure 3.8: A common example of a series-shunt feedback amplifier. This is the standard non-inverting voltage amplifier opamp configuration.

in shunt with the input to feedback a current as well as the output to sense a voltage. If the blocks are near ideal (unilateral blocks with no loading) then the following formulas directly compute the feedback parameters.

$$A = V_o/I_i \quad (3.39)$$

$$a = V_o/I_e \quad (3.40)$$

$$T = I_f/I_e \quad (3.41)$$

$$f = I_f/V_o \quad (3.42)$$

A typical example of a shunt-shunt feedback amplifier constructed from an opamp and a resistor is shown in Figure 3.9 on the next page.

Series-Series Feedback A feedback amplifier that has voltage as the input and current as the output is referred to as being in the series-series configuration because the feedback network must be connected in series with the input to feedback a voltage as well as the output to sense a current. If the blocks are near ideal (unilateral blocks with no loading) then the following formulas directly compute

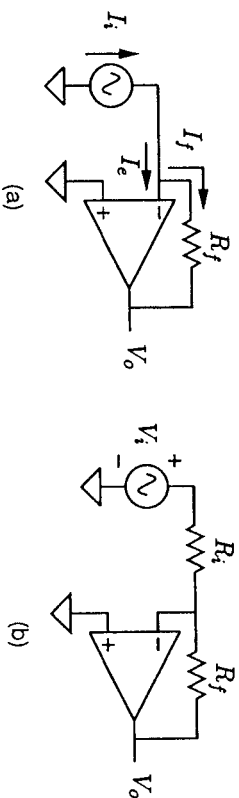


Figure 3.9: A common example of a shunt-shunt feedback amplifier (a). It is a transresistance amplifier, it converts a current to a voltage. If a resistor is connected to the input, as in (b), this circuit becomes the standard inverting voltage amplifier opamp configuration. The input resistor acts as a voltage-to-current converter.

the feedback parameters.

$$A = I_o/V_i \quad (3.43)$$

$$a = I_o/V_e \quad (3.44)$$

$$T = V_f/V_e \quad (3.45)$$

$$f = V_f/I_o \quad (3.46)$$

Shunt-Series Feedback A feedback amplifier that has current as both an the input and the output is referred to as being in the shunt-series configuration because the feedback network must be connected in shunt with the input to feedback a current and in series with the output to sense a current. If the blocks are near ideal (unilateral blocks with no loading) then the following formulas directly compute the feedback parameters.

$$A = I_o/I_i \quad (3.47)$$

$$a = I_o/I_e \quad (3.48)$$

$$T = I_f/I_e \quad (3.49)$$

$$f = I_f/I_o \quad (3.50)$$

3.4.1.3 Measurement Techniques for Abstract Feedback Systems

When first taught about feedback systems, undoubtedly you were told that the loop gain is the gain around the loop once the loop has been broken. It is mentioned much less often that the loop gain is also the gain about the loop even when the loop is not broken. The same is true for the open loop gain of the amplifier. As a result, a large number of designers feel they must break the loop in order to measure a , f , and T . This is the basis of several of the methods commonly used to measure loop gain. These methods are illustrated in Figure 3.10 on the following page for a unity-gain series-shunt feedback amplifier. However, breaking the loop in a real feedback circuit causes the loading on the blocks to change, which changes the characteristics of the feedback loop.

The methods presented in this section, only half of which actually break the loop, are approximate if the components that make up the feedback loop are not ideal. Methods for characterizing non-ideal feedback circuits are given later.

Closed-Loop Measurements If a feedback system is made up of unilateral blocks that do not load each other, then measuring the four feedback parameters is very easy. Simply apply an input signal to the circuit while its feedback loop is operating normally (closed-loop) and perform an AC analysis over a range of frequencies. Measure the four feedback parameters using the equations already given for the appropriate feedback configuration. For example, if we assume that the opamp in Figure 3.8 on page 71 is ideal (zero input admittance, output impedance, reverse coupling, and common-mode gain), then the feedback parameters are directly found using (3.35–3.38). Accuracy of this approach degrades when it is applied to real circuits. As shown in Figure 3.14 on page 80, loading effects, reverse coupling through the opamp, and forward coupling through the feedback circuit cause the computed results to be inaccurate at high frequencies.

In practice, this method is also inaccurate at low frequencies due to nonzero common-mode gain in the amplifier. Consider an amplifier

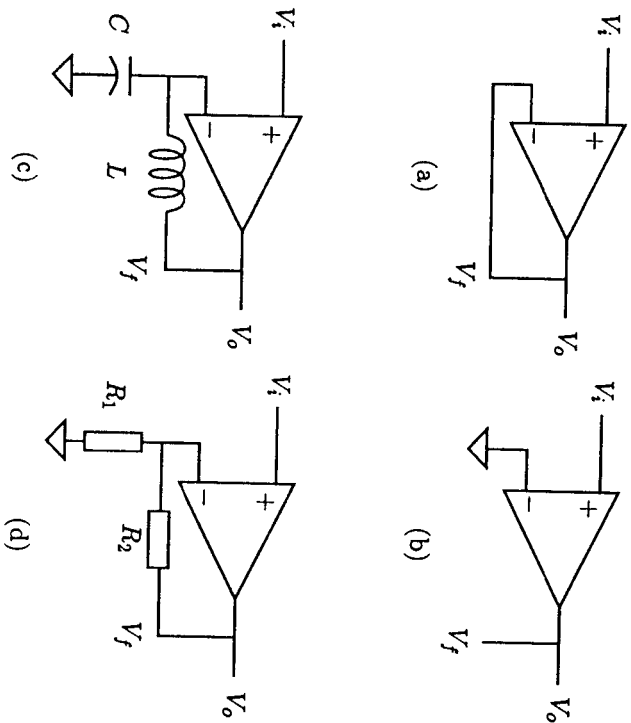


Figure 3.10: Three approaches for measuring the loop gain of a feedback amplifier that involve breaking the loop, as demonstrated on the unity-gain series-shunt configuration (a). In the first method (b) the loop is physically broken. In the second (c), L and C are extremely large and act as a low pass filter with a very low cutoff frequency. As a result, the loop is closed at very low frequencies, and open at all other frequencies. Finally, in the last method (d), R_1 and R_2 are special resistors that take different values in AC and DC analyses. In DC, R_1 is large and R_2 is small, which closes the loop. In AC, R_1 is small and R_2 is large, which opens the loop.

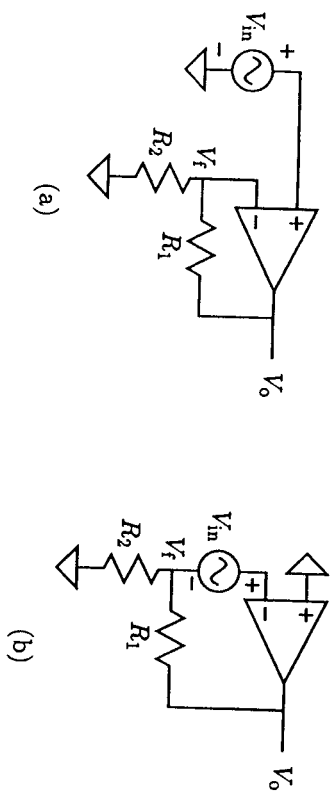


Figure 3.11: When measuring the feedback parameters for the non-inverting amplifier configuration, the error due to large common-mode signals is reduced greatly by moving the input source from the positive input (a) to the negative input (b).

with a differential mode gain of $a_{dm} = 100k$, a common mode gain of $a_{cm} = 2$ and a feedback factor of $f = 1$.

$$V_o = a_{dm}(V_p - V_n) - a_{cm}(V_p + V_n)/2 \quad (3.51)$$

$$V_n = fV_o = V_o \quad (3.52)$$

$$(1 + a_{dm} + a_{cm}/2)V_o = (a_{dm} - a_{cm}/2)V_p \quad (3.53)$$

$$V_o = \frac{(a_{dm} - a_{cm}/2)}{(1 + a_{dm} + a_{cm}/2)} V_p \quad (3.54)$$

The composite open loop gain is

$$a = \frac{V_o}{V_p - V_n} = \frac{V_o}{V_p - V_o} = \frac{(a_{dm} - a_{cm}/2)}{(1 + a_{cm})} \quad (3.55)$$

Thus, the open loop gain a is not a_{dm} as usually desired. For the example given, $a = 33k$, which is off by a factor of 3. This problem is corrected by rearranging the circuit slightly as shown in Figure 3.11. In this case

$$V_o = a_{dm}(V_p - V_n) - a_{cm}(V_p + V_n)/2 \quad (3.56)$$

$$V_p = 0 \quad (3.57)$$

$$V_o = (a_{cm}/2 - a_{dm})V_n \quad (3.58)$$

$$a = a_{cm}/2 - a_{dm} \quad (3.59)$$

For the example given, the error is negligible.

Breaking the Loop The most obvious approach to measuring the loop gain is to simply break the loop as shown in Figure 3.10b. This approach is generally only used by naive designers because it suffers from a serious problem. Breaking the loop disturbs the DC operating point. This is a serious problem with high gain amplifiers such as opamps. Even a small change in the DC levels at the input causes the output to clip at the rails when the feedback loop is broken.

Using a Low-Pass Filter The next two approaches try to close the loop at DC while keeping it open at the frequencies of interest in order to avoid the problem of the first approach. Figure 3.10c shows how a low pass filter is inserted in the feedback loop to attenuate the feedback signal to negligible levels at all but the lowest frequencies. This approach is rather cumbersome because if the band edge of the filter is chosen too high, the effect of the filter interferes with the measurement of the feedback parameters. If the band edge is too low, then the capacitors and inductors that make up the filter have values that are so large that they cause numerical problems in the simulator, with the result that it acts strangely or fails completely. Also, the filter loads the amplifier, which changes the characteristics of the loop.

Opening the Loop Only During the AC Analysis A refinement of the low-pass filter method that is available with some simulators is shown in Figure 3.10d. In this method, the low-pass filter components are replaced with either resistors that change their values (HSPICE and others), or a switch that changes its position (Spectre), as a function of the analysis being run by the simulator. In

DC analysis, the resistors/switch are configured to pass the feedback signal and so close the loop. In AC analysis, they are reconfigured to block the feedback signal and so open the loop. This approach is similar to the low-pass filter approach in that the loop is maintained at DC to avoid any shift in the DC operating point, but if the effective cutoff frequency of the filter becomes 0^+ Hz and it is less likely that the component values will be set to extreme values and cause numerical problems in the simulator. While this is the most accurate of the methods that break the loop, it still has accuracy problems, especially at high frequencies, because the loading of the components changes when the loop is opened. However, common mode signals are not a problem when breaking the loop, even for the standard non-inverting opamp configuration, because the act of breaking the loop grounds the negative input and greatly reduces the common mode signal.

Middlebrook's Method An approach that does not involve opening the loop and does not require “magic” components was developed by Middlebrook [middlebrook75]. The measurements are made on series-input configurations with the test circuit shown in Figure 3.12 on the following page. Simply insert a test voltage source in the feedback loop, set $V_{in} = 0$ and $V_i = 1$ and perform an AC analysis. The two gains are computed as follows:

$$a_v = \frac{V_o}{V_p - V_n} \quad (3.60)$$

$$T_v = -\frac{V_f}{V_n} \quad (3.61)$$

The quantity T_v is called the voltage loop gain. It equals the true loop gain T of the opamp if the impedance looking into the negative input of the opamp is much larger than the impedance looking into the feedback network from the input of the opamp for all frequencies.

It is similarly possible to measure shunt-input configurations by inserting a test current source as shown in Figure 3.13. Simply set $I_n = 0$ and $I_i = 1$ and perform an AC analysis. The two gains are computed as follows:

$$a_i = \frac{V_o}{I_p - I_n} \quad (3.62)$$

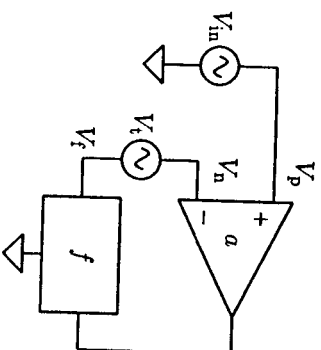


Figure 3.12: Test circuit for measuring open-loop gain and voltage loop gain of a series-input opamp using Middlebrook's method. The magnitude of V_{in} is set to zero and the magnitude of V_i is set to 1.

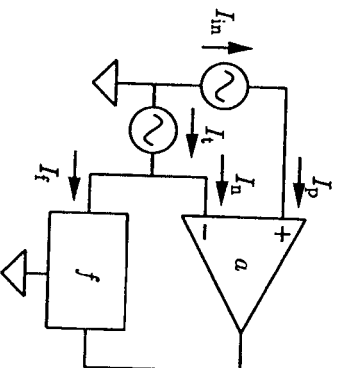


Figure 3.13: Test circuit for measuring current loop gain of a shunt-input opamp using Middlebrook's method. The magnitude of I_{in} is set to zero and the magnitude of I_i is set to 1.

$$T_i = \frac{I_f}{I_n} \quad (3.63)$$

The current loop gain T_i equals the true loop gain T of the opamp if the impedance looking into the negative input of the opamp is much smaller than the impedance looking into the feedback network from the input of the opamp.

So far there is nothing different here than what was presented in the section on closed-loop measurements. However, Middlebrook goes farther by developing a more accurate formula for loop gain. If you cannot find a point in the loop that satisfies either assumption, then the accuracy of the loop gain measurement is improved by measuring both T_v and T_i and combining them as follows:

$$T = \frac{T_v T_i - 1}{2 + T_v + T_i} \quad (3.64)$$

This equation becomes inaccurate if $T \ll 1$.

Characterizing Just the Opamp All of these methods make assumptions about the components in the feedback loop. The two most common assumptions is that the blocks do not load each other and that the both the amplifier and the feedback circuit are unidirectional. Often when working with opamps these assumptions hold well enough at low frequencies. Of course, most people are also very interested in the characteristics of the feedback system at high frequencies. Capacitive and inductive loading effects usually play an important role in the behavior of real feedback circuits. In addition, feedback networks are almost never unidirectional and forward coupling through the feedback network can strongly affect stability by adding right-hand plane zeros. So for both of these reasons, all of these methods are not accurate enough at high frequencies to be applied to most real circuits, as is clear from Figures 3.14 and 3.15.

If you are only interested in the open-loop gain of the amplifier it is often possible to improve the accuracy of your measurements by replacing the feedback circuit with one that is ideal. For example, the opamp of Figure 3.16a is in the standard non-inverting configuration. In this circuit, the input of the amplifier loads the feed-

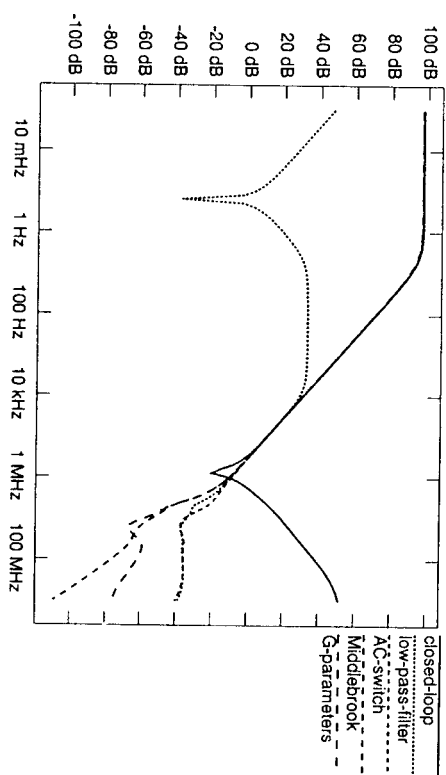
Loop Gain of ua741 in NonInverting Configuration ($A_v = 2$)

Figure 3.14: Loop gain for a $\mu a741$ opamp in a series-shunt configuration (shown in Figure 3.11 on page 75) as measured by the methods described in this section. At high frequencies all of the approximate methods become inaccurate. The G-parameter method is the reference of comparison. It is an approach to computing the feedback parameters that is accurate for non-ideal circuits (presented in next section). The Middlebrook method (3.64) is the next most accurate, followed by AC-switch (opening the loop only during AC analysis). The low-pass filter method is similar to AC-switch at high frequencies but in this case is inaccurate at low frequencies. Finally, the closed-loop techniques produce terrible results at high frequencies.

3.4. Applications of the AC Analyses

81

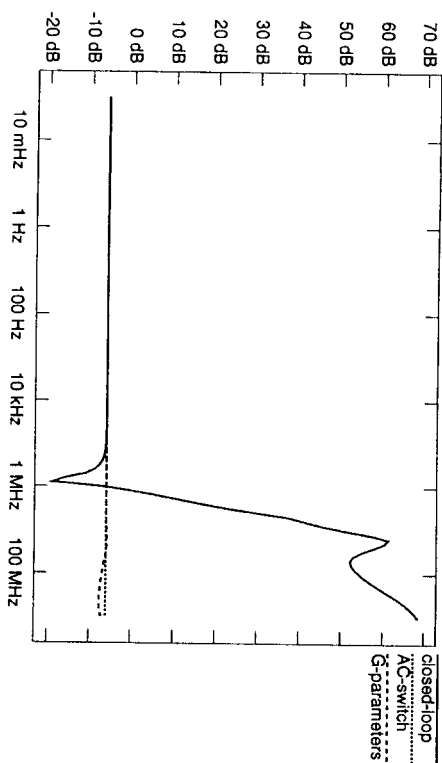
Feedback Factor of ua741 in NonInverting Configuration ($A_v = 2$)

Figure 3.15: Feedback factor for a $\mu a741$ opamp in a series-shunt configuration (shown in Figure 3.11 on page 75) as measured by the methods described in this section. Notice that errors in the closed-loop measure causes anomalous results that suggest the feedback network has gain.

back network of R_1 and R_2 . In addition, the output of the opamp is loaded by the feedback network, and some of the input signal bypasses the opamp by traveling the wrong way through the feedback network. These problems are eliminated by replacing the feedback network constructed from resistors with one constructed with a voltage-controlled voltage source as shown in Figure 3.16b. The advantage of this approach is that it is easier to accurately compute the open-loop gain of a circuit with an ideal feedback network. However, the disadvantage is that the circuit is unrealistic. In real circuits, the feedback network affects the behavior of the amplifier, which is not modeled when ideal feedback is used.

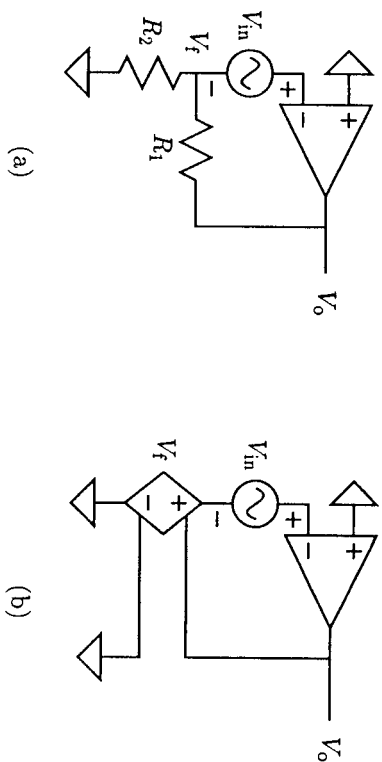


Figure 3.16: An opamp in the non-inverting amplifier configuration. The resistors that form the feedback network in (a) are loaded by the input of the amplifier, load the output of the amplifier, and provide a undesirable path for the input signal to reach the output. These non-idealities are eliminated by using a voltage-controlled voltage source as the feedback network as in (b). Notice that these circuits configure the input to avoid large common-mode signals, as shown in Figure 3.11 on page 75.

Once the feedback circuit is replaced by an ideal controlled source, as in Figure 3.16b, one can apply the simple closed-loop formulas (3.35–3.50). If the formulas can be applied directly at the controlled source, the results are accurate because the controlled source is not subject to loading. This is shown in Figure 3.17 on the next page, which shows the open-loop gain computed by applying (3.35–3.38) to a $\mu A741$ in series-shunt configuration as shown in Figure 3.16b.

The effect of loading is even more dramatic when measuring the open-loop gain of an opamp using the inverting amplifier configuration. In this setting, the feedback system is in shunt-shunt configuration, and so the input is a current and the output is a voltage. Thus, the amplifier is acting as a transresistance amplifier. The transresistance for

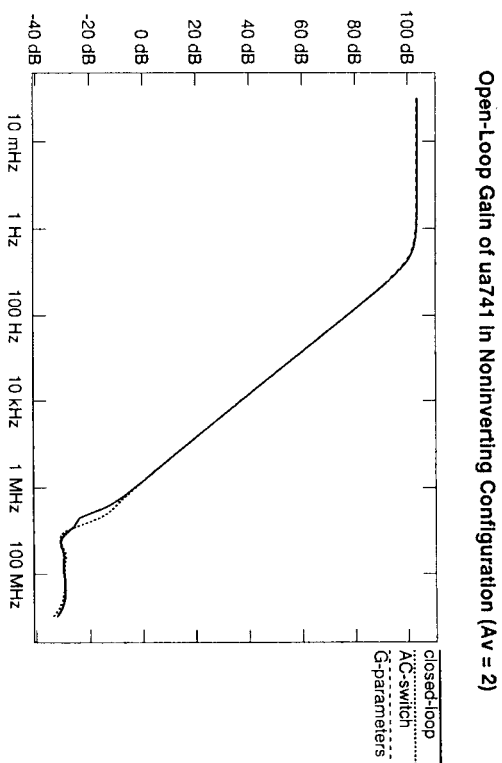


Figure 3.17: Open-loop gain for a $\mu A741$ opamp in a series-shunt configuration as measured with a voltage-controlled voltage source for the feedback network as shown in Figure 3.16 on the preceding page. Using a controlled source for the feedback network greatly reduces the affect of loading.

high input-impedance opamps is very high because it takes very little input current to affect a large change in the output voltage. When a resistor is used as the feedback network, as shown in Figure 3.18a, the resistor loads the input of the opamp, greatly reducing its effective transresistance. If instead, a voltage-controlled current source is used, as shown in Figure 3.18b, the effective transresistance is much higher because the feedback network does not load the inputs of the opamp. The open-loop transresistance of a $\mu A741$ was measured with both test circuits, and the results are shown in Figure 3.19.

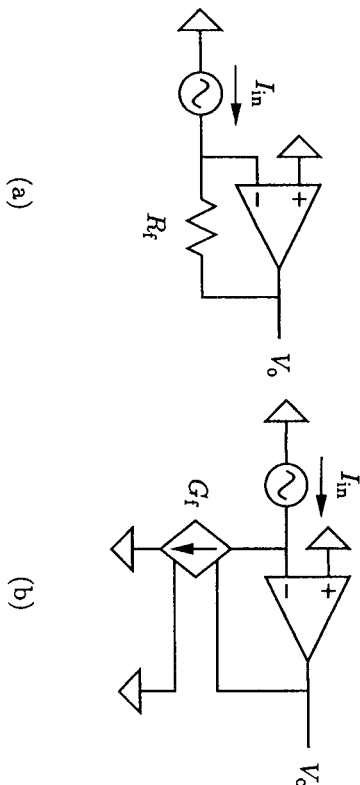


Figure 3.18: An opamp in the inverting amplifier configuration. The resistor that form the feedback network in (a) strongly loads the input of the amplifier. The loading is eliminated by replacing the resistor with a voltage-controlled current source as in (b).

3.4.1.4 Feedback Parameters of Real Circuits

When measuring the feedback parameters of a circuit, it is important to realize that unless the individual components are ideal you should not base your conclusions on measurements of the individual components. The effective open-loop gain is dependent on the source, the load, and the feedback circuitry. Similarly, the effective feedback factor is dependent on the amplifier. To accurately measure the feedback parameters, the circuit must be operated as a whole. In particular the common belief that one should break the feedback loop to measure loop gain is only appropriate on ideal circuits. This means that in practice, you should almost never break the feedback loop (if your components are so ideal, then why are you simulating them?) Even circuits that contain opamps should be simulated with the feedback loop intact because loading and bilateral coupling are significant at high frequencies.

The methods given in the previous section give acceptable answers

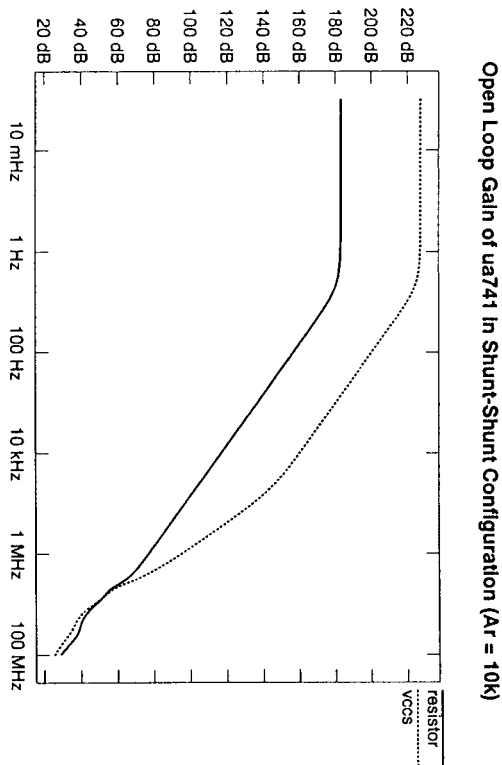


Figure 3.19: Open-loop gain for a $\mu A741$ opamp in a shunt-shunt configuration. The gain marked 'resistor' was measured using Figure 3.18a, and the gain marked 'vccs' was measured using Figure 3.18b. The large discrepancy results from the resistor loading the high-impedance input of the opamp, which dramatically reduces its effective transresistance.

on opamp circuits at low frequencies. However it is also important to accurately characterize the feedback loop at high frequencies. For example, when exploring the stability of the feedback loop, the magnitude and phase of the loop gain must be accurate well above the unity gain frequency. In addition, loading effects often play an important role at all frequencies in feedback circuits that do not contain opamps, such as the one in Figure 3.20. In these situations, the methods presented earlier are not appropriate.

To accurately handle imperfections in the blocks such as loading and bilateral coupling, we must repartition the circuit into new blocks

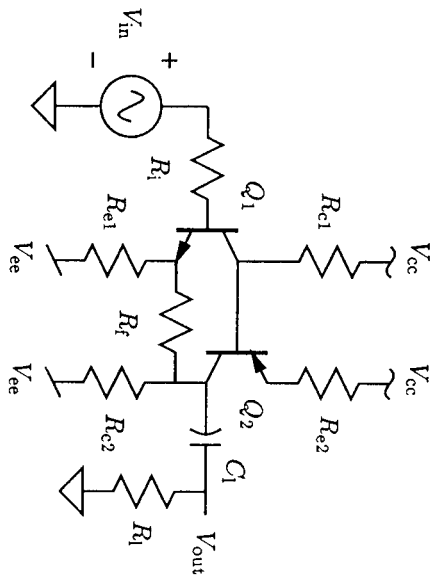


Figure 3.20: A simple feedback pair. Loading and bilateral coupling play an important role in this circuit. The simple schemes for measuring the feedback parameters that were presented earlier are not applicable on this circuit.

such that the imperfections are contained inside the new blocks. The feedback parameters are then derived for the new blocks. This approach acknowledges that the previous approaches were doomed to failure because they were constrained to take the designers partitioning even though the blocks as partitioned by the designer do not satisfy the required assumptions. In this approach, the circuit is taken as a whole and the blocks are recognized through their actual measurements. In other words, the blocks are recognized by their actual behavior, and not by some apriori partitioning. In fact, a single component can reside in more than one block. For example, the forward transmission through a resistor in the feedback network is naturally included as part of the amplifier, whereas the reverse transmission is included in the feedback block. This approach allows us to calculate the feedback parameters without unjustified assumptions [Gray84].

3.4. Applications of the AC Analyses

87

To handle bilateral coupling, simple redefine the amplifier to include all forward coupling, and redefine the feedback network to include all reverse coupling. In other words, if the feedback network exhibits forward coupling, then modify the open-loop gain of the amplifier to include it. If the amplifier exhibits some reverse coupling, simply modify the feedback factor to include it.

Loading is handled in a similar manner. Loading of the source by the amplifier and feedback circuit, and loading of the amplifier by the feedback circuit and the load, is handled by incorporating all of the loading effects at both the input and the output into the amplifier and adjusting the open-loop gain accordingly.

Shunt-Shunt Feedback This process can be illustrated if the non-ideal amplifier and feedback circuit are represented as two-ports as shown for the shunt-shunt configuration in Figure 3.21 on the following page. With the shunt-shunt configuration the two-ports are represented using Y-parameters to allow the loading and couplings to be easily merged. In this configuration, the amplifier and the feedback are connected in parallel at both the input and the output. Y-parameters, being admittances, are the most convenient representation because the admittance of a parallel combination of admittances is simply the sum of the admittances.

$$y_{11} = y_s + y_{11a} + y_{11f} \quad (3.65)$$

$$y_{21} = y_{21a} + y_{21f} \quad (3.66)$$

$$y_{12} = y_{12a} + y_{12f} \quad (3.67)$$

$$y_{22} = y_{1f} + y_{22a} + y_{22f} \quad (3.68)$$

Using these composite numbers, the effective open-loop gain and feedback factor is computed by first computing the closed-loop gain and matching terms with (3.29).

$$A = \frac{V_o}{I_s} = \frac{y_{21}}{y_{21}y_{12} - y_{11}y_{22}} \quad (3.69)$$

Divide through by $-y_{11}y_{22}$ to see the correspondence with (3.29).

$$A = \frac{-y_{21}/(y_{11}y_{22})}{1 - y_{21}y_{12}/(y_{11}y_{22})} \quad (3.70)$$

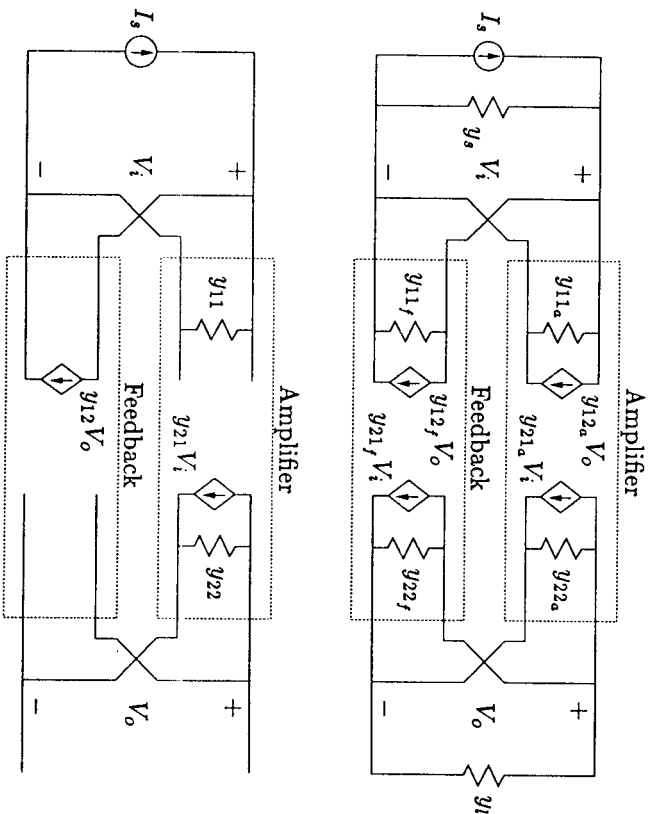


Figure 3.21: Above is the Y -parameter representation of the shunt-shunt feedback configuration. Below is the same circuit, except all forward coupling and loading effects have been incorporated into the amplifier, and all reverse coupling has been incorporated into the feedback.

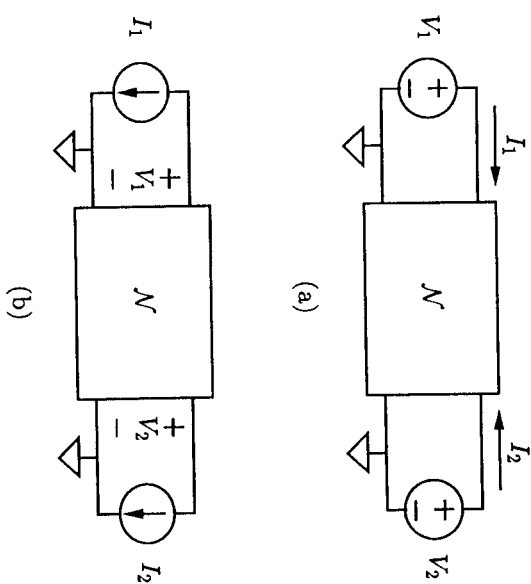


Figure 3.22: Measuring Y -parameters (a) and Z -parameters (b).

Thus, the effective open-loop gain and feedback factor are seen to be

$$a = -\frac{y_{21}}{y_{11}y_{22}} \quad (3.71)$$

$$f = y_{12} \quad (3.72)$$

A and T are computed from (3.29) and (3.30).

In most shunt-shunt feedback amplifiers, one cannot directly compute the Y -parameters because it requires connecting a voltage source directly to the input and the output as shown in Figure 3.22a. Doing so breaks the loop and disturbs the DC operating point. Just as it was possible to open the loop only during AC analysis by using “magic” resistors or switches, it is also possible to measure the Y -parameters by only connecting the voltage sources during the AC analysis. When doing so, it is better to use ideal switches than

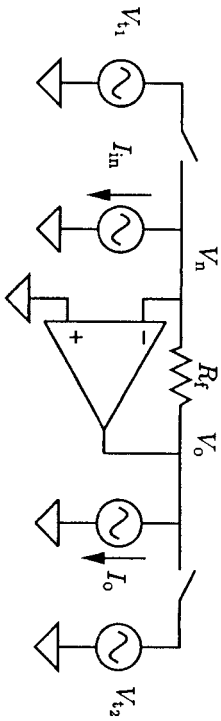


Figure 3.23: Test circuit that measures the Y - and Z -parameters of a shunt-shunt feedback circuit. The voltage sources needed to measure the Y -parameters are only connected to the circuit during AC analysis to prevent them from disturbing the operating point of the opamp. During normal operation the switches are open. To measure the Y -parameters, close the switches and perform two AC analyses, one with $V_1 = 1$ and $V_2 = 0$, the other with $V_1 = 0$ and $V_2 = 1$. To measure Z -parameters, open the switches and perform two AC analyses, the first with $I_{in} = 1$ and $I_o = 0$, and the second with $I_{in} = 0$ and $I_o = 1$.

resistors because of the very low input and output impedance of the feedback system. A circuit for making this measurement is shown in Figure 3.23 and Netlist 3.2.

Another approach is to rederive the feedback parameters in terms of Z -parameters, which are measured with current sources that do not disturb operating point. To measure Z -parameters, use the circuit shown in Figure 3.22b. First, set the magnitude of I_1 to one and the magnitude of I_2 to zero. Use an AC analysis to measure V_1 and V_2 .

$$z_{11} = V_1 \quad (3.73)$$

$$z_{21} = V_2 \quad (3.74)$$

Now set the magnitude of I_1 to zero and the magnitude of I_2 to one.

$$z_{12} = V_1 \quad (3.75)$$

```
// Test circuit for measuring AC Y-parameters
simulator lang=spectre

// Feedback circuit
OA1 (0 in out) ua741
Rf (out in ) resistor r=10k
Iin (in 0 ) isource mag=0

// Components used to measure Y-parameters
Vt1 (t1 0 ) vsource mag=1
Vt2 (t2 0 ) vsource mag=0
Sw1 (t1 in ) switch position=0 ac_position=1
Sw2 (t2 out ) switch position=0 ac_position=1

// Analyses
Vx1 ac start=1_Hz stop=1GHz
disableVt1 alter dev=Vt1 param=mag value=0
enableVt2 alter dev=Vt2 param=mag value=1
Vx2 ac start=1_Hz stop=1GHz
```

Netlist 3.2: Spectre netlist for the circuit shown in Figure 3.23 on the facing page. It is used to measure the Y -parameters of a shunt-shunt feedback circuit. The switches are special in that they are open during DC analysis and they close for AC analysis. A Spectre netlist is given because it is the only simulator that supports ideal switches that can be configured to change position only for AC analysis.

$$z_{22} = V_2 \quad (3.76)$$

The Z -parameters are converted to Y -parameters using [chna87]

$$\Delta = z_{11}z_{22} - z_{12}z_{21} \quad (3.77)$$

$$y_{11} = \frac{z_{22}}{\Delta} \quad (3.78)$$

$$y_{21} = -\frac{z_{21}}{\Delta} \quad (3.79)$$

$$y_{12} = -\frac{z_{12}}{\Delta} \quad (3.80)$$

$$y_{22} = \frac{z_{11}}{\Delta}. \quad (3.81)$$

Equations (3.71) and (3.72) are then rewritten as

$$a = \frac{z_{21}(z_{11}z_{22} - z_{21}z_{12})}{z_{11}z_{22}} \quad (3.82)$$

$$f = \frac{-z_{12}}{z_{11}z_{22} - z_{21}z_{12}} \quad (3.83)$$

$$T = \frac{z_{21}z_{12}}{z_{11}z_{22}} \quad (3.84)$$

$$A = -z_{21} \quad (3.85)$$

Unfortunately, while these formulas work well with real circuits, they fail on ideal systems. In particular, they fail if z_{11} or z_{22} are zero because the zero port impedance prevents measuring distinctly the gain of the forward and reverse paths from the terminals. Instead, (3.39–3.42) should be used, which involves inserting current probes into the loop.

Consider the inverting opamp configuration shown in Figure 3.24 on the next page. To measure z_{11} and z_{21} , set $I_i = 1$ and $I_o = 0$, then

$$z_{11} = V_n \quad (3.86)$$

$$z_{21} = V_o. \quad (3.87)$$

To measure z_{12} and z_{22} , set $I_i = 0$ and $I_o = 1$, then

$$z_{12} = V_n \quad (3.88)$$

$$z_{22} = V_o. \quad (3.89)$$

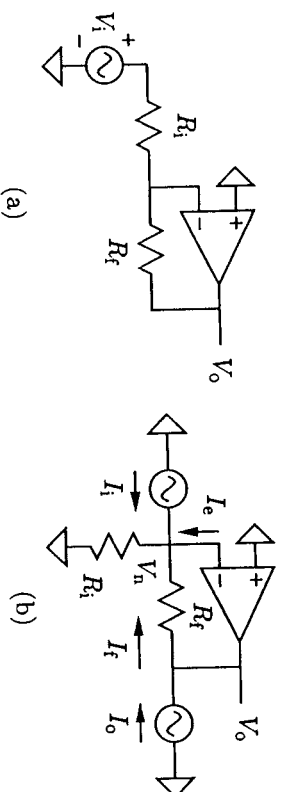


Figure 3.24: Standard inverting voltage amplifier configuration (a). This circuit employs shunt-shunt feedback and the feedback loop consists of the opamp and R_f . As such, the feedback amplifier takes the current as an input and produces voltage as an output. Thus the circuit is interpreted as the cascade of a voltage-to-current converter (R_i) that is followed by a current-to-voltage converter (the feedback amplifier). When measuring the Z -parameters to determine the feedback parameters of the feedback amplifier, R_i must be included, as in (b), because it provides considerable loading of the feedback circuit (R_f).

The Z -parameters are converted to Y -parameters using (3.77–3.81). And finally, the feedback parameters are computed using

$$a = -\frac{y_{21}}{y_{11}y_{22}} \quad (3.90)$$

$$f = y_{12} \quad (3.91)$$

$$T = af \quad (3.92)$$

$$A = z_{21}. \quad (3.93)$$

These results are shown in Figure 3.25 on the following page.

Series-Series Feedback A similar approach is used to measure the feedback parameters for the series-series configuration, except that in this case it is most convenient to use Z -parameters for the

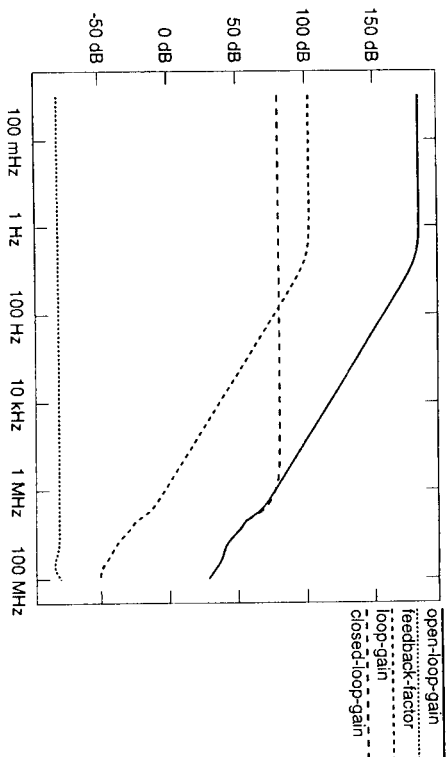
Feedback Parameters for ua741 In Inverting Configuration ($A_r = 10k$)

Figure 3.25: The feedback parameters for the shunt-shunt feedback circuit of Figure 3.24 on the page before. These results were computed using Netlist 3.2 on page 91 and (3.29), (3.30), (3.71), and (3.72).

derivations and Y -parameters for the measurements. In this configuration, the amplifier and the feedback are connected in series at both the input and the output. Z -parameters, being impedances, are the most convenient representation for the derivation because the impedance of a series combination of impedances is simply the sum of the impedances. Merging of the Z -parameters for the amplifier and feedback in a series-series feedback amplifier is shown in Figure 3.26 on the next page.

$$z_{11} = z_s + z_{11a} + z_{11f} \quad (3.94)$$

$$z_{21} = z_{21a} + z_{21f} \quad (3.95)$$

$$z_{12} = z_{12a} + z_{12f} \quad (3.96)$$

$$z_{22} = z_l + z_{22a} + z_{22f} \quad (3.97)$$

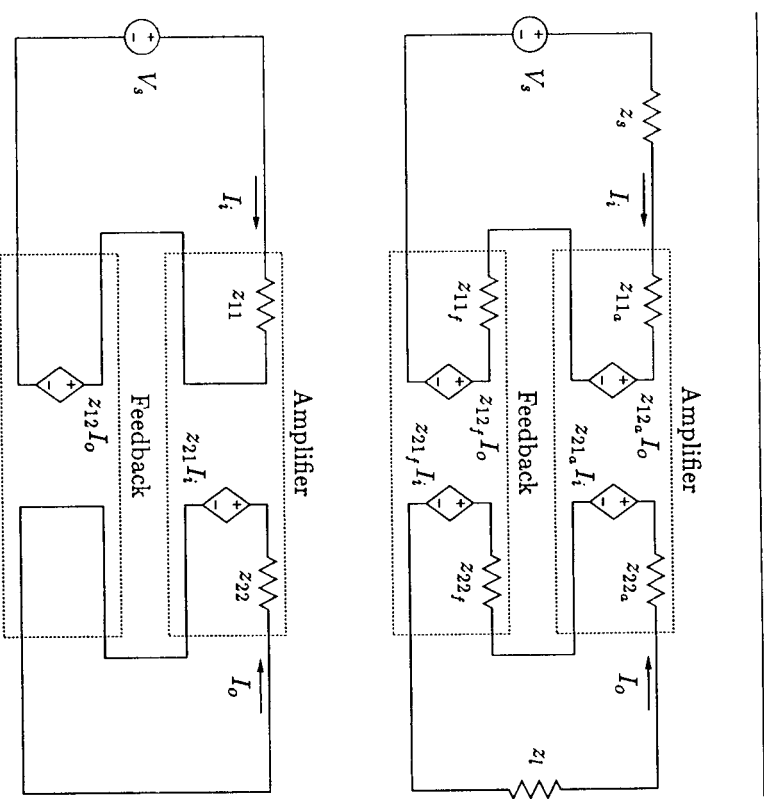


Figure 3.26: Above is the Z -parameter representation of the series-series feedback configuration. Below is the same circuit, except all forward coupling and loading effects have been incorporated into the amplifier, and all reverse coupling has been incorporated into the feedback.

The effective open-loop gain and feedback factor are computed by first computing the closed-loop gain and matching terms with (3.29).

$$A = \frac{I_o}{V_s} = \frac{z_{21}}{z_{21}z_{12} - z_{11}z_{22}} \quad (3.98)$$

Divide through by $-z_{11}z_{22}$ to see the correspondence with (3.29).

$$A = \frac{-z_{21}/(z_{11}z_{22})}{1 - z_{21}z_{12}/(z_{11}z_{22})} \quad (3.99)$$

Thus, the effective open-loop gain and feedback factor are seen to be

$$a = -\frac{z_{21}}{z_{11}z_{22}} \quad (3.100)$$

$$f = z_{12} \quad (3.101)$$

A and T are computed from (3.29) and (3.30).

In most series-series feedback amplifiers, one cannot directly compute the Z -parameters because it requires connecting a current source in series with the input and the output as shown in Figure 3.22 on page 89a. Doing so breaks the loop and disturbs the DC operating point. As before, there are two approaches to overcoming this problem. The “magic” switches that open only during AC analysis can be used to only open the loop in AC analysis to avoid disturbing the operating point. Or the feedback parameters can be rederived in terms of Y -parameters, which are measured with voltage sources that do not disturb operating point.

To measure Y -parameters, use the circuit shown in Figure 3.22a on page 89. First, set the magnitude of V_1 to one and the magnitude of V_2 to zero. Use an AC analysis to measure I_1 and I_2 .

$$y_{11} = I_1 \quad (3.102)$$

$$y_{21} = I_2 \quad (3.103)$$

Now set the magnitude of V_1 to zero and the magnitude of V_2 to one.

$$y_{12} = I_1 \quad (3.104)$$

$$y_{22} = I_2 \quad (3.105)$$

The Y -parameters are converted to Z -parameters using [chna87]

$$\Delta = y_{11}y_{22} - y_{12}y_{21} \quad (3.106)$$

$$z_{11} = \frac{y_{22}}{\Delta} \quad (3.107)$$

$$z_{21} = -\frac{y_{21}}{\Delta} \quad (3.108)$$

$$z_{12} = -\frac{y_{12}}{\Delta} \quad (3.109)$$

$$z_{22} = \frac{y_{11}}{\Delta} \quad (3.110)$$

Equations (3.100) and (3.101) are then rewritten as

$$a = \frac{y_{21}(y_{11}y_{22} - y_{21}y_{12})}{y_{11}y_{22}} \quad (3.111)$$

$$f = \frac{-y_{12}}{y_{11}y_{22} - y_{21}y_{12}} \quad (3.112)$$

$$T = \frac{y_{21}y_{12}}{y_{11}y_{22}} \quad (3.113)$$

$$A = -y_{21} \quad (3.114)$$

Unfortunately, while these formulas work well with real circuits, they fail on ideal systems. In particular, they fail if y_{11} or y_{22} are zero because the zero port admittance prevents measuring distinctly the gain of the forward and reverse paths from the terminals. Instead, (3.43–3.46) should be used.

Series-Shunt Feedback With the series-shunt configuration, it is best to use H -parameters for the derivations and G -parameters for the measurements. The non-ideal amplifier and feedback circuit are represented as two-ports as shown in Figure 3.27 on the following page. The same process used on the previous two configurations is also used on the series-shunt configuration. In this configuration, the amplifier and the feedback are connected in series at the input and in parallel at the output. H -parameters, being impedances at the input and admittances at the output, are the most convenient representation.

$$h_{11} = z_s + h_{11a} + h_{11f} \quad (3.115)$$

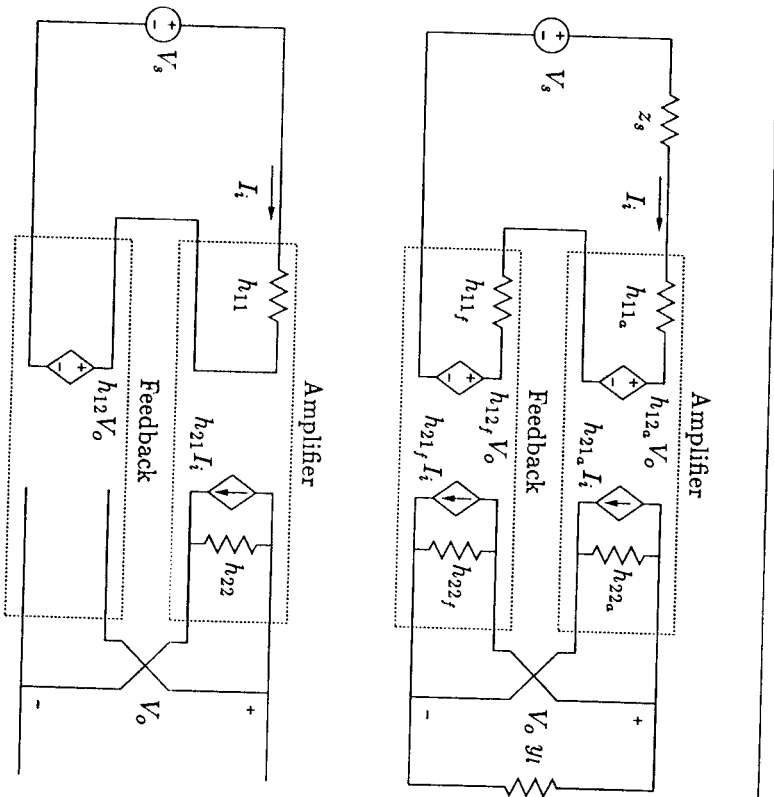


Figure 3.27: Above is the H -parameter representation of the series-shunt feedback configuration. Below is the same circuit, except all forward coupling and loading effects have been incorporated into the amplifier, and all reverse coupling has been incorporated into the feedback.

3.4. Applications of the AC Analyses

99

$$h_{21} = h_{21a} + h_{21f} \quad (3.116)$$

$$h_{12} = h_{12a} + h_{12f} \quad (3.117)$$

$$h_{22} = y_l + h_{22a} + h_{22f} \quad (3.118)$$

Using these composite numbers, the effective open-loop gain and feedback factor are computed by first computing the closed-loop gain and matching terms with (3.29).

$$A = \frac{V_o}{V_s} = \frac{h_{21}}{h_{21}h_{12} - h_{11}h_{22}} \quad (3.119)$$

Divide through by $-h_{11}h_{22}$ to see the correspondence with (3.29).

$$A = \frac{-h_{21}/(h_{11}h_{22})}{1 - h_{21}h_{12}/(h_{11}h_{22})} \quad (3.120)$$

Thus, the effective open-loop gain and feedback factor are seen to be

$$a = -\frac{h_{21}}{h_{11}h_{22}} \quad (3.121)$$

$$f = h_{12} \quad (3.122)$$

A and T are computed from (3.29) and (3.30).

As before, one generally cannot compute the H -parameters directly because it requires connecting a voltage source in parallel with the input and a current source in series with the output as shown in Figure 3.28a. Doing so breaks the loop and disturbs the DC operating point. As before, “magic” switches that change their position only during AC analysis are used to measure the H -parameters without affecting the operating point, as shown in Figure 3.29. Otherwise, it is necessary to rederive the feedback parameters in terms G -parameters, which can be measured without disturbing the operating point.

To measure G -parameters, use the circuit shown in Figure 3.28b. First, set the magnitude of V_1 to one and the magnitude of I_2 to zero. Use an AC analysis to measure I_1 and V_2 .

$$g_{11} = I_1 \quad (3.123)$$

$$\frac{V_2}{I_1} = V_2 \quad (3.124)$$

BIBLIOTHEQUE
UNIVERSITE

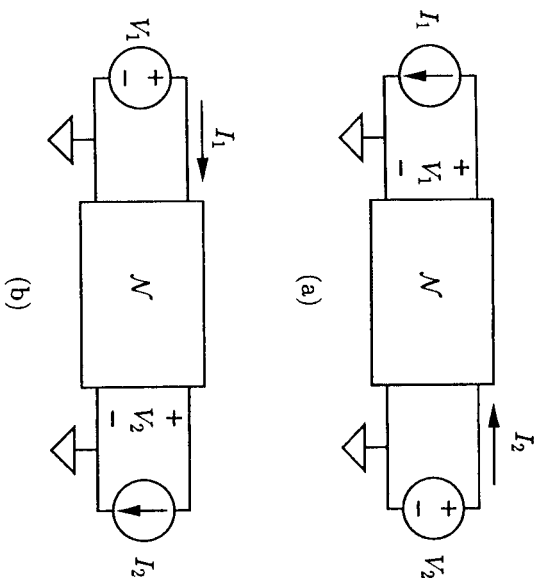


Figure 3.28: Measuring H -parameters (a) and G -parameters (b).

Now set the magnitude of V_1 to zero and the magnitude of I_2 to one.

$$g_{12} = I_1 \quad (3.125)$$

$$g_{22} = V_2 \quad (3.126)$$

The G -parameters are converted to H -parameters using [chu87]

$$\Delta = g_{11}g_{22} - g_{12}g_{21} \quad (3.127)$$

$$h_{11} = \frac{g_{22}}{\Delta} \quad (3.128)$$

$$h_{21} = -\frac{g_{21}}{\Delta} \quad (3.129)$$

$$h_{12} = -\frac{g_{12}}{\Delta} \quad (3.130)$$

$$h_{22} = \frac{g_{11}}{\Delta}. \quad (3.131)$$

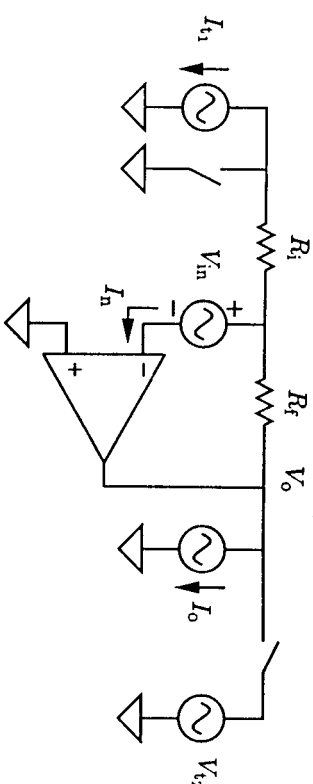


Figure 3.29: Test circuit that measures the H - and G -parameters of a series-shunt feedback circuit. The sources needed to measure the H -parameters are only connected to the circuit during AC analysis to prevent them from disturbing the operating point of the opamp. During normal operation, the switch on the input is closed and the switch on the output is open. To measure H -parameters, perform two AC analyses with the input switch open and the output switch closed. In the first AC analysis, $I_{i1} = 1$ and $V_{i2} = 0$. In the second, $I_{i1} = 0$ and $V_{i2} = 1$. To measure the G -parameters, set the switches for normal operation and perform two AC analyses, the first with $V_{in} = 1$ and $I_o = 0$, and the second with $V_{in} = 0$ and $I_o = 1$.

Equations (3.121) and (3.122) are then rewritten as

$$a = \frac{g_{21}(g_{11}g_{22} - g_{21}g_{12})}{g_{11}g_{22}} \quad (3.132)$$

$$f = \frac{-g_{12}}{g_{11}g_{22} - g_{21}g_{12}} \quad (3.133)$$

$$T = \frac{g_{21}g_{12}}{g_{11}g_{22}} \quad (3.134)$$

$$A = -g_{21} \quad (3.135)$$

Unfortunately, while these formulas work well with real circuits, they fail on ideal systems. In particular, they fail if g_{11} or g_{22} are zero

because the zero port immittance prevents measuring distinctly the gain of the forward and reverse paths from the terminals. Instead, (3.35–3.38) should be used.

Consider the non-inverting opamp configuration shown in Figure 3.29 on the page before. To measure g_{11} and g_{21} , close the input switch, open the output switch and set $V_{in} = 1$ and $I_o = 0$, then

$$g_{11} = I_n \quad (3.136)$$

$$g_{21} = V_o. \quad (3.137)$$

To measure g_{12} and g_{22} , set $V_{in} = 0$ and $I_o = 1$, then

$$g_{12} = I_n \quad (3.138)$$

$$g_{22} = V_o. \quad (3.139)$$

The G -parameters are converted to H -parameters with (3.127–3.131). And finally, the feedback parameters are computed using

$$a = -\frac{h_{21}}{h_{11}h_{22}} \quad (3.140)$$

$$f = h_{12} \quad (3.141)$$

$$T = af \quad (3.142)$$

$$A = g_{21}. \quad (3.143)$$

These results are shown in Figure 3.30 on the facing page.

Shunt-Series Feedback Finally for the shunt-series configuration, G -parameters are used for the derivation, and H -parameters are used for the measurements. The non-ideal amplifier and feedback circuit are represented as two-ports as shown in Figure 3.31. In this configuration, the amplifier and the feedback are connected in parallel at the input and in series at the output. G -parameters, being admittances at the input and impedances at the output, are the most convenient representation for the derivation.

$$g_{11} = g_s + g_{11a} + g_{11f} \quad (3.144)$$

$$g_{21} = g_{21a} + g_{21f} \quad (3.145)$$

$$g_{12} = g_{12a} + g_{12f} \quad (3.146)$$

$$g_{22} = g_l + g_{22a} + g_{22f} \quad (3.147)$$

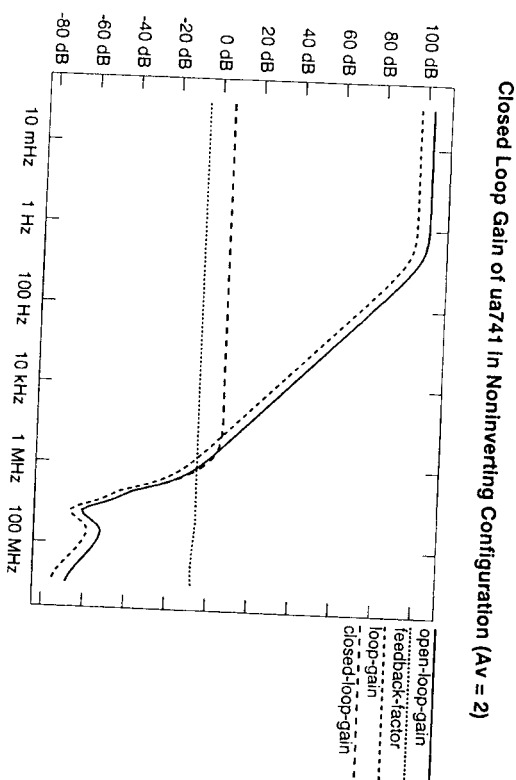


Figure 3.30: The feedback parameters for the series-shunt feedback circuit of Figure 3.11b on page 75. These results were computed using (3.140–3.143).

The effective open-loop gain and feedback factor are computed by first computing the closed-loop gain and matching terms with (3.29).

$$A = \frac{I_o}{I_s} = \frac{g_{21}}{g_{21}g_{12} - g_{11}g_{22}} \quad (3.148)$$

Divide through by $-g_{11}g_{22}$ to see the correspondence with (3.29).

$$A = \frac{-g_{21}/(g_{11}g_{22})}{1 - g_{21}g_{12}/(g_{11}g_{22})} \quad (3.149)$$

Thus, the effective open-loop gain and feedback factor are seen to be

$$a = -\frac{g_{21}}{g_{11}g_{22}} \quad (3.150)$$

$$f = g_{12} \quad (3.151)$$

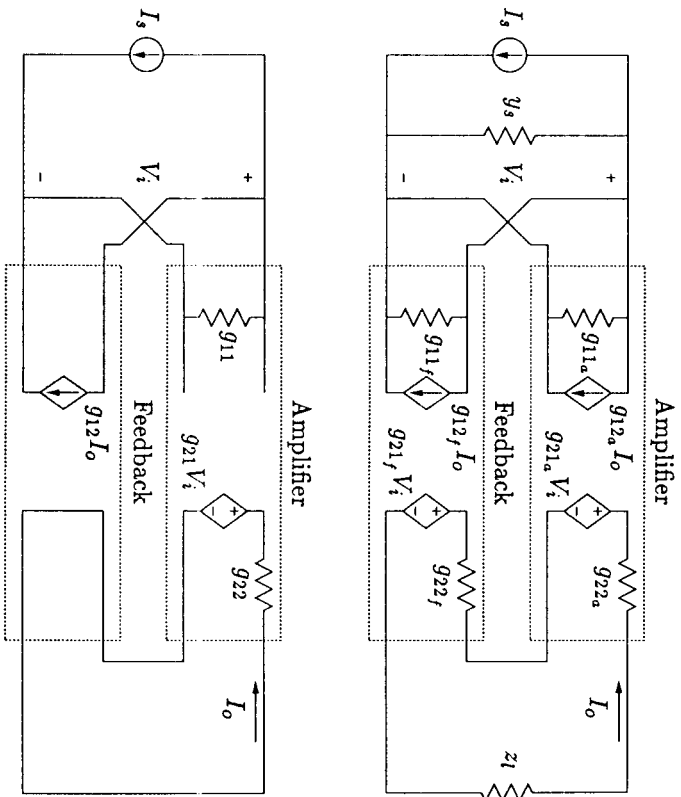


Figure 3.31: Above is the G -parameter representation of the shunt-series feedback configuration. Below is the same circuit, except all forward coupling and loading effects have been incorporated into the amplifier, and all reverse coupling has been incorporated into the feedback.

3.4. Applications of the AC Analyses

105

A and T are computed from (3.29) and (3.30).

In most shunt-series feedback amplifiers, one cannot directly compute the G -parameters because it requires connecting a voltage source in parallel with the input and a current source in series with the output as shown in Figure 3.28b. Doing so breaks the loop and disturbs the DC operating point. As before, “magic” switches that change their position only during AC analysis can be used to measure the H -parameters without affecting the operating point. Otherwise, the feedback parameters are rederived in terms of H -parameters, which can be measured without disturbing the operating point.

To measure H -parameters, use the circuit shown in Figure 3.28a. First, set the magnitude of I_1 to one and the magnitude of V_2 to zero. Use an AC analysis to measure V_1 and I_2 .

$$h_{11} = V_1 \quad (3.152)$$

$$h_{21} = I_2 \quad (3.153)$$

Now set the magnitude of I_1 to zero and the magnitude of V_2 to one.

$$h_{12} = V_1 \quad (3.154)$$

$$h_{22} = I_2 \quad (3.155)$$

The H -parameters are converted to G -parameters using [chna87]

$$\Delta = h_{11}h_{22} - h_{12}h_{21} \quad (3.156)$$

$$g_{11} = \frac{h_{22}}{\Delta} \quad (3.157)$$

$$g_{21} = -\frac{h_{21}}{\Delta} \quad (3.158)$$

$$g_{12} = -\frac{h_{12}}{\Delta} \quad (3.159)$$

$$g_{22} = \frac{h_{11}}{\Delta}. \quad (3.160)$$

Equations (3.150) and (3.151) are then rewritten as

$$a = \frac{h_{21}(h_{11}h_{22} - h_{21}h_{12})}{h_{11}h_{22}} \quad (3.161)$$

$$f = \frac{-h_{12}}{h_{11}h_{22} - h_{21}h_{12}} \quad (3.162)$$

$$T = \frac{h_{21}h_{12}}{h_{11}h_{22}} \quad (3.163)$$

$$A = -h_{21} \quad (3.164)$$

Unfortunately, while these formulas work well with real circuits, they fail on ideal systems. In particular, they fail if h_{11} or h_{22} are zero because the zero port immittance prevents measuring distinctly the gain of the forward and reverse paths from the terminals. Instead, (3.47–3.50) should be used.

A Final Comment These measurements are tricky and sensitive. Be skeptical when making these measurements and, as always, thoroughly explore unexpected behavior. A summary of the characteristics of each of the feedback configurations is shown in Table 3.1 on the next page.

3.4.2 Transfer Function Versus Bias

With the AC analyses in SPICE it is only possible to sweep frequency. Spectre can sweep frequency, temperature, or any component in-stance or model parameter. The ability to sweep component parameters provides a new capability not available with SPICE. One useful application is to compute the gain of an amplifier as a function of bias point as a way of exploring its linearity. Consider the simple differential pair shown in Figure 3.32 on page 108. Netlist 3.3 contains the differential pair as well as an analysis that sweeps the DC input voltage. When you sweep something other than frequency in an AC analysis, you must specify the analysis frequency. You are free to choose any value. In this case we are uninterested in any effects of frequency, so $f = 0$ was chosen. Figure 3.33 shows the transconductance as a function of differential input voltage. Notice that the transconductance immediately begins to drop as the input voltage moves away from zero, indicating that the differential pair is not very linear.

Type	Configurations			
	Shunt-Shunt	Series-Shunt	Shunt-Series	Series-Series
Trans-resistance				
S_i	I_i	V_i	I_i	V_i
S_o	V_o	V_o	I_o	I_o
S_e	I_e	V_e	I_e	V_e
S_f	I_f	V_f	I_f	V_f
Z_i	low	high	low	high
Z_o	low	low	high	high
Δ	$z_{11}z_{22} - z_{21}z_{12}$	$g_{11}g_{22} - g_{21}g_{12}$	$h_{11}h_{22} - h_{21}h_{12}$	$y_{11}y_{22} - y_{21}y_{12}$
$a = \frac{S_o}{S_e}$	$\frac{V_o}{I_e} = -\frac{y_{21}}{y_{11}y_{22}} \frac{z_{21}\Delta}{z_{11}z_{22}}$	$\frac{V_o}{V_e} = \frac{h_{21}}{h_{11}h_{22}} \frac{g_{21}\Delta}{g_{11}g_{22}}$	$\frac{I_o}{I_e} = -\frac{g_{21}}{g_{11}g_{22}} \frac{h_{21}\Delta}{h_{11}h_{22}}$	$\frac{I_o}{V_e} = -\frac{z_{21}}{z_{11}z_{22}} \frac{y_{21}\Delta}{y_{11}y_{22}}$
$f = \frac{S_f}{S_o}$	$\frac{I_f}{V_o} = \frac{y_{12}}{y_{12}\Delta} = -\frac{z_{12}}{\Delta}$	$\frac{V_f}{V_o} = h_{12} = \frac{-g_{12}}{\Delta}$	$\frac{I_f}{I_o} = g_{12} = \frac{-h_{12}}{\Delta}$	$\frac{V_f}{I_o} = z_{12} = \frac{-y_{12}}{\Delta}$
$T = \frac{S_f}{S_e}$	$\frac{I_f}{I_e} = \frac{-y_{21}y_{12}}{y_{11}y_{22}} \frac{z_{21}z_{12}}{z_{11}z_{22}}$	$\frac{V_f}{V_e} = \frac{h_{21}h_{12}}{h_{11}h_{22}} \frac{g_{21}g_{12}}{g_{11}g_{22}}$	$\frac{I_f}{I_e} = \frac{-g_{21}g_{12}}{g_{11}g_{22}} \frac{h_{21}h_{12}}{h_{11}h_{22}}$	$\frac{V_f}{V_e} = \frac{-z_{21}z_{12}}{z_{11}z_{22}} \frac{y_{21}y_{12}}{y_{11}y_{22}}$
$A = \frac{S_o}{S_i}$	$\frac{V_o}{I_i} = \frac{-y_{21}}{\Delta} = -z_{21}$	$\frac{V_o}{V_i} = \frac{-h_{21}}{\Delta} = -g_{21}$	$\frac{I_o}{I_i} = \frac{-g_{21}}{\Delta} = -h_{21}$	$\frac{I_o}{V_i} = \frac{-z_{21}}{\Delta} = -y_{21}$

Table 3.1: Properties of the four feedback configurations.

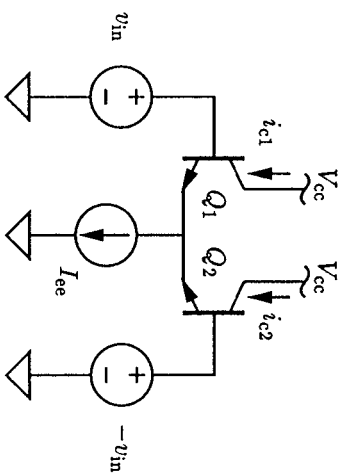


Figure 3.32: A simple differential pair.

There are many possible ways to make a more linear differential transconductance stage. One such way is to use emitter degeneration by placing a resistor in series with the emitter of each transistor. However, a more interesting approach is Barrie Gilbert's multi-tanh doublet [gilbert82] shown in Figure 3.34 on page 111. The normalized transconductance of this circuit is shown in Figure 3.35. The curve is much flatter about 0, which indicates greater linearity.

3.4.3 Capacitance Versus Bias

Computing the capacitance of nonlinear components as a function of bias point is another example of how it is useful to be able to perform a small-signal analysis while sweeping a parameter other than frequency. For example, consider a simple junction diode. To compute capacitance, apply a unit magnitude voltage source with a fixed frequency across the diode. The admittance of the diode is simply

$$Y = \frac{I}{V}, \quad (3.165)$$

```
// Test circuit for measuring differential
// transconductance of emitter-coupled pair
simulator lang=spectre

// Differential Pair
model npn bjt type=npn
Q1 (c1 b1 e) npn
Q2 (c2 b2 e) npn
Iee (e 0) isource dc=100ua
Vcc (cc 0) vsource dc=5
Ic1 (c1 cc) iprobe
Ic2 (c2 cc) iprobe

// Input network
Vin (b 0) vsource dc=0 mag=1
Av1 (b1 0 b 0) vcvs gain=1/2
Av2 (b2 0 b 0) vcvs gain=-1/2

// Output network
Ai1 (o 0) cccs probe=Ic1 gain=1/2
Ai2 (o 0) cccs probe=Ic2 gain=-1/2
Gm (o 0) iprobe

// Analyses
measureGm ac start=-0.25 stop=0.25 dev=Vin param=dc \
lin=200 freq=0
```

Netlist 3.3: Spectre netlist for the circuit shown in Figure 3.32 on the facing page. It is used to measure the transconductance of a emitter-coupled pair as a function of differential input voltage.

Transconductance of Differential Pair as Function of DC Input Voltage

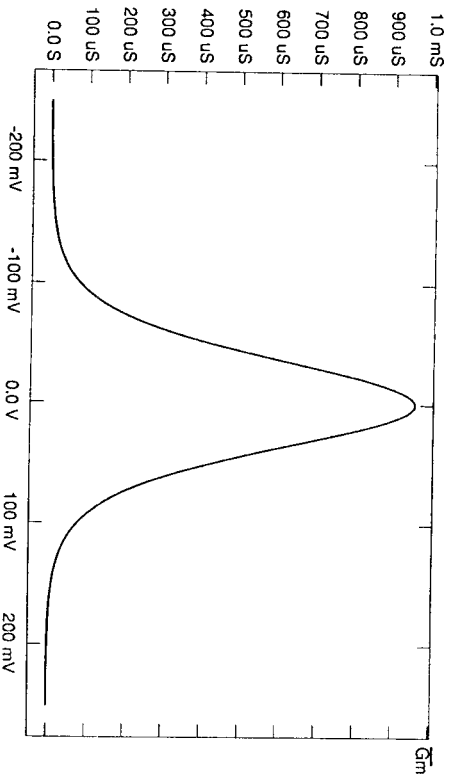


Figure 3.33: Transconductance of emitter-coupled pair as a function of input bias as measured with Netlist 3.3 on the preceding page.

where I is the small-signal diode current and $V = 1$ is the applied voltage. Capacitance and conductance are computed using

$$Y = G + j\omega C \quad (3.166)$$

where Y is complex, but G and C are both real. Thus,

$$G = \Re\{Y\} \quad (3.167)$$

$$C = \frac{1}{\omega} \Im\{Y\} \quad (3.168)$$

For best accuracy, choose ω such that the conductance ($\Re\{Y\}$) and the susceptance ($\Im\{Y\}$) are reasonably close in size over the whole range of biases. The actual frequency used is not critical though, and can be chosen to allow conversion from susceptance ($\Im\{Y\}$) to capacitance by inspection. For example, choosing $\omega = 10^{12}$ makes

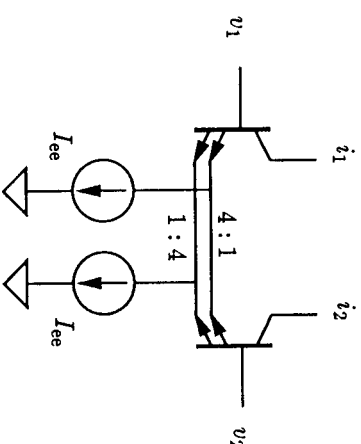


Figure 3.34: Barrie Gilbert's multi-tanh doublet.

the susceptance is equal to the capacitance in picofarads. Figure 3.36 shows the capacitance of a junction as a function of bias voltage.

A word of caution. This approach is suitable for measuring C and G of a component that is modeled by a parallel combination of a nonlinear resistor and capacitor. By using current as a test signal one can extract R and $1/C$ of a resistor and capacitor in series. It is questionable whether it makes sense to use this technique to measure the resistance and capacitance of more complicated structures or distributed components. Be aware that

1. Parasitic resistors in the semiconductor make the measurement of the nonlinear intrinsic resistance and capacitance inaccurate and so should be eliminated during this measurement.
2. The excess phase term in the bipolar transistor model makes the transistor behave as a distributed component. As such, it cannot be modeled by a multiterminal parallel RC . If the excess phase terms are present, either use a very small test frequency to make the effect of the excess phase term small, or change the model to eliminate it. The same is true for

Transconductance of Multi-Tanh Doublet

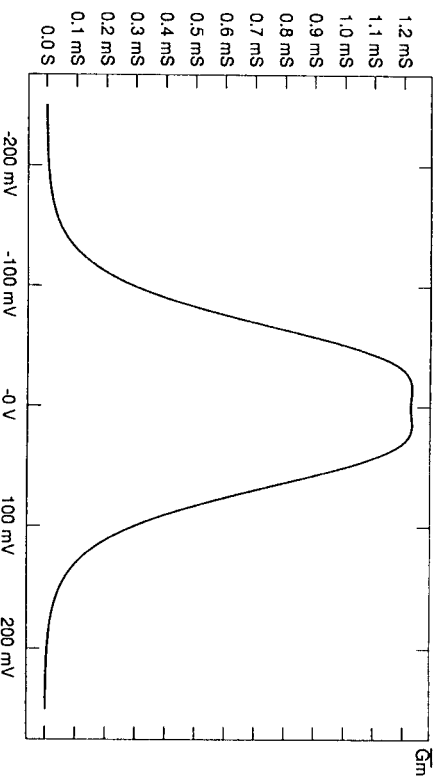
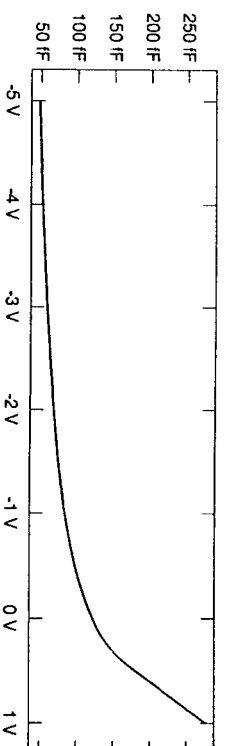


Figure 3.35: Transconductance of Gilbert's multi-tanh doublet.

Junction Capacitance

Figure 3.36: Capacitance of a pn -junction as a function of bias voltage.

any of the new non-quasistatic semiconductor models being developed.

3.4.4 Non-Quiescent Operating Points

While it is most common to perform a small-signal analysis at a DC or quiescent operating point, in some cases it is enlightening to perform a small-signal analysis about a non-quiescent operating point. The frequency response of a nonlinear circuit can change dramatically as a function of operating point. Consider the $\mu A741$ opamp in unity gain configuration. Its frequency response as computed about the DC operating point is shown in the bottom of Figure 3.37. Also shown is the frequency response computed while the opamp is undergoing slew-rate limiting. It is interesting to note that the direction that the output is slewing has a dramatic affect on the loop gain.

To make this measurement, the opamp was driven with a unit step, the response to which is shown in the top of Figure 3.37. A transient analysis is performed to 10 μs , which is right in the middle of the slew-rate limiting region. The transient analysis is immediately followed by an AC analysis. To get the AC analysis in Spectre (SPICE does not provide this capability) to use the final point of the transient analysis as the operating point, the prevpoint parameter of the AC analysis is set to yes. This prevents the AC analysis from recomputing the DC operating point and causes it to linearize the circuit about the last point computed by the transient analysis.

3.4.5 Differential Amplifiers

When characterizing differential amplifiers, it is desirable to characterize both the differential- and common-mode behavior of the amplifier, as well as the coupling between differential- and common-mode signals. Using the balun shown in Figure 3.38, Netlist 3.4 and Netlist 3.5, it is easy to drive or measure both differential- and common-mode signals, making these measurements convenient. The balun implements the following equations:

$$v_d = v_p - v_n \quad \text{--- differential-mode voltage} \quad (3.169)$$

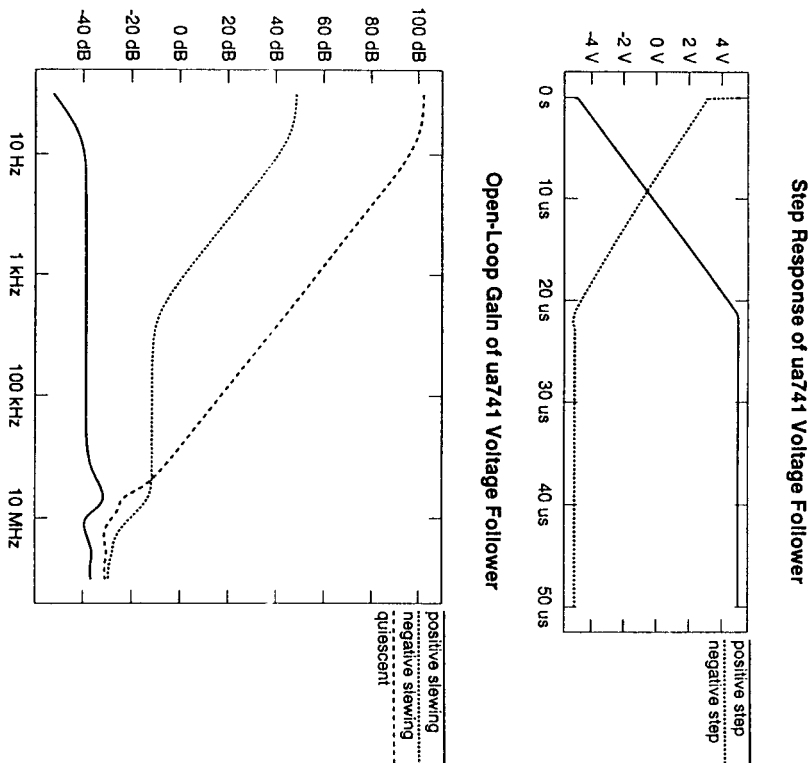


Figure 3.37: Step response of μ A741 voltage follower (top). Transient analysis is stopped at 10 μ s in order to perform a small-signal analysis to compute the frequency response of the amplifier during slew-rate limiting. Frequency response when quiescent, and at 10 μ s when undergoing slew-rate limiting (bottom).

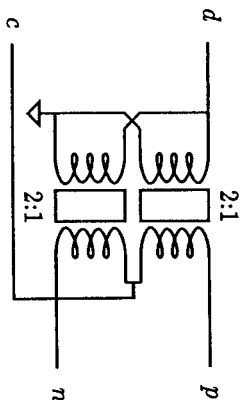


Figure 3.38: A balun for converting ground-referred differential-mode (d) and common-mode (c) signals to balanced positive (p) and negative (n) signals. Ideals transformers are used, and so the balun works and is accurate at all frequencies, including DC.

```
// BALUN
//
// A bidirectional balanced-unbalanced converter.
// Maps between the unbalanced signals 'd' and 'c'
// and the balanced signals 'p' and 'n'.
simulator lang=spectre

subckt balun (d c p n)
  T1 (d 0 p c) transformer n1=2
  T2 (d 0 c n) transformer n1=2
ends balun
```

Netlist 3.4: Balun for Spectre implemented using ideal transformers.

```

* BALUN
*
* A bidirectional balanced-unbalanced converter.
* Maps between the unbalanced signals 'd=1' and 'c=2'
* and the balanced signals 'p=3' and 'n=4'.
.subckt balun (1 2 3 4)
d c p n
*
* Ideal transformer for the positive input
E1 5 2 1 0 2.0
V1 3 5
F1 1 0 V1 0.5
R1 1 0 1T
*
* Ideal transformer for the negative input
E2 6 4 1 0 2.0
V2 2 7
F2 1 0 V2 0.5
R2 7 6 1u
.ends balun

```

Netlist 3.5: Balun for SPICE implemented with controlled sources. This balun implements the ideal transformers using controlled sources. The resistors are present only to prevent SPICE from complaining about topology errors.

$v_c = (v_p + v_n)/2$	—	common-mode voltage	(3.170)
$v_p = v_c + v_d/2$	—	positive voltage	(3.171)
$v_n = v_c - v_d/2$	—	negative voltage	(3.172)
$i_d = i_p - i_n$	—	differential-mode current	(3.173)
$i_c = (i_p + i_n)/2$	—	common-mode current	(3.174)
$i_p = i_c + i_d/2$	—	positive current	(3.175)
$i_n = i_c - i_d/2$	—	negative current	(3.176)

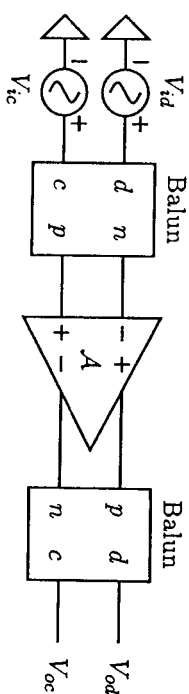


Figure 3.39: Test circuit for a differential amplifier that uses baluns at the input and output.

Notice that the balun is bidirectional. Either the unbalanced signals (d for differential mode and c for common mode) or the balanced signals (p for positive and n for negative) can act as the inputs or the outputs. This means that the same circuit is used at the input of a differential amplifier to convert the stimuli to the balanced form needed to drive the amplifier, and at the output to separate the balanced output into distinct differential-mode and common-mode signals for easy measurement. An example test circuit is shown in Figure 3.39.

Another advantage of using a the balun of Figure 3.38 is that it accurately separates the differential- and common-mode currents. Thus, to measure the differential output impedance, drive the differential output (d terminal on the output balun) with a unit-magnitude AC current source. The differential output impedance then equals the voltage on the differential output (d terminal on the output balun). The differential and common-mode input and output impedance are measured in a similar manner.

Using the balun, it is also possible to use the 2-port parameter methods described in Section 3.4.1.4 on page 84 to measure the feedback parameters such a loop-gain.

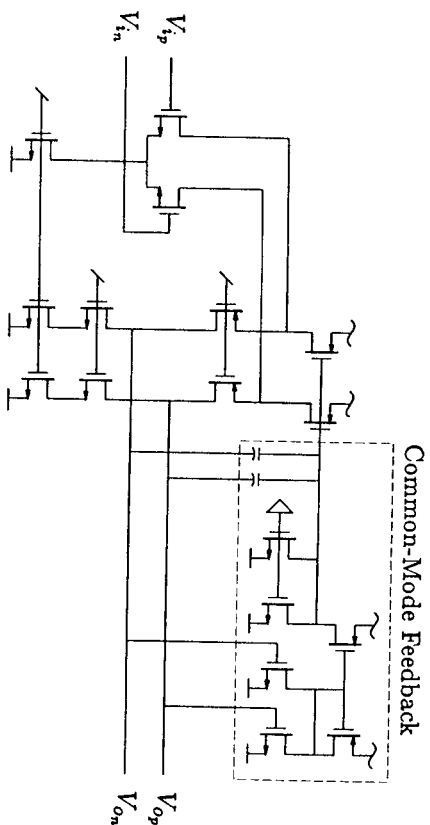


Figure 3.40: Fully-differential CMOS opamp used to illustrate many of the measurements described in this section.

Example In this section we use the fully-differential CMOS opamp shown in Figure 3.40 to illustrate many of the measurements common to differential amplifiers. The opamp is configured as a fully-differential amplifier with a capacitive feedback network as shown in Figure 3.41 on the next page. Capacitive feedback is commonly used in switched-capacitor filters. The balun of Section 3.4.5 is used to simplify the measurement of differential mode and common mode quantities. Netlists 3.6–3.9 contain the test circuit and the analyses used to characterize the circuit. XF analysis directly computes differential- and common-mode gain, differential/common-mode conversion, output impedance, and power supply rejection (Figure 3.42). AC analysis allows direct measurement of differential- and common-mode gain, differential/common-mode conversion and input impedance (there is some overlap between the quantities computable with AC and XF analyses) (Figure 3.43). The step response to differential- and common-mode signals is computed using transient analysis (Figure 3.44). Finally, noise analysis measures differential-

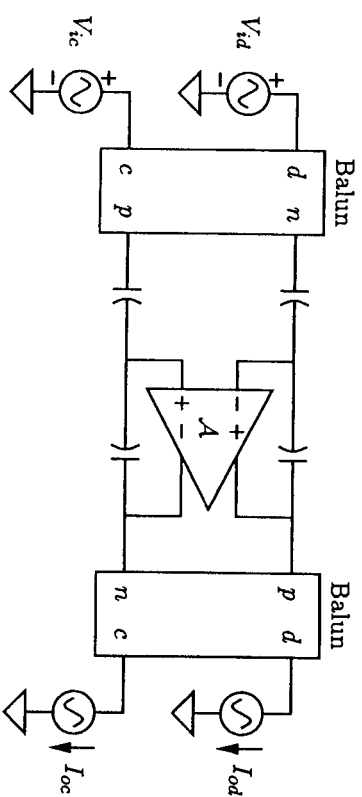


Figure 3.41: Fully-differential CMOS opamp with capacitive feedback.

and common-mode output and input referred noise (Figure 3.45).

3.5 Summary

This is a list of the key points presented in this chapter along with the section and page numbers where they were presented.

- The AC analyses compute the small-signal sinusoidal steady-state response of a circuit. Section 3.1 on page 51.
 1. The AC analyses do not compute the transient behavior of a circuit, and so it is difficult in general to determine stability of a circuit by inspecting the results computed by AC analysis.
 2. AC analysis computes the response of linearized time-invariant circuits, and so cannot be applied to circuits where frequency conversion or translation are significant, such as mixers.

```
// Fully-Differential Operational Amplifier
global 0 vdd vss
simulator lang=spectre

// Select models
#define PROCESS_CORNER TYPICAL
#define VDD 5.0_V
#include "cmos.mod"

// Include circuit
#include "opamp.sub"
#include "balun.sub"
#include "test.sub"

// Save only signals at the top level
spectre options save=lv1pub nestlv1=1

// Calculate operating point
opPoint dc opPoint=logfile save=allpub

// Differential-mode characteristics
#include "dm-measure.anal"

// Common-mode characteristics
#include "cm-measure.anal"
```

Netlist 3.6: Top-level netlist that characterizes the differential and common-mode characteristics of a fully-differential CMOS opamp. The opamp is shown in Figure 3.40, the balun is given in Netlist 3.4, the test circuit is given Netlist 3.7, the differential-mode measurements are made in Netlist 3.8 and the common-mode measurements are made in Netlist 3.9.

```
// Fully-Differential Operational Amplifier Test Circuit
simulator lang=spectre

// Power supplies
Vdd (vdd 0 ) vsource dc=VDD
Vss (vss 0 ) vsource dc=-VDD

// Inputs (disabled initially)
Vid (din 0 ) vsource type=dc mag=0 val0=0 val1=10 \
    width=1u delay=10ns
Vic (cin 0 ) vsource type=dc mag=0 val0=0 val1=10 \
    width=1u delay=10ns
Iod (dout 0 ) isource // to measure Rout-DM
Ioc (cout 0 ) isource // to measure Rout-CM

// Convert separated DM and CM inputs to balanced inputs
Tin (din cin pin nin ) balun

// Convert balanced output into separated DM and CM outputs
Tout (dout cout pout nout) balun

// Feedback amplifier
A1 (pout nout pvg nvg ) opamp
Cf1 (pout nvg ) capacitor c=8p
Cf2 (nout pvg ) capacitor c=8p
C11 (pout 0 ) capacitor c=8p
C12 (nout 0 ) capacitor c=8p
C11 (pin pvg ) capacitor c=2p
C12 (nin nvg ) capacitor c=2p
```

Netlist 3.7: Test circuit, shown in Figure 3.41, use to characterize the fully-differential amplifier. Netlist continues from Netlist 3.6 and continues in Netlist 3.8.

```
// Measure Differential-Mode Characteristics

// Gain, CM to DM coupling, PSRR, Rout
dmXferFunctions (dout 0) xf start=1 stop=1G dec=10 \
    title="DM Transfer Functions to V(dout)"

// Gain, DM to CM coupling, Rin
dmEnableDiffIn alter dev=Vid param=mag value=1
dmAC ac start=1 stop=1G dec=10 \
    title="DM Transfer Functions from Vid"
dmDisableDiffIn alter dev=Vid param=mag value=1

// Step response
dmEnablePulse alter dev=Vid param=type value=pulse
dmStepResponse tran stop=2us errpreset=conservative \
    title="DM Step Response"
dmDisablePulse alter dev=Vid param=type value=dc

// Noise
dmNoise (dout 0) noise start=1 stop=1G dec=10 \
    iprobe=Vid title="DM Noise"
```

Netlist 3.8: Differential-mode measurements used to characterize the fully-differential amplifier shown in Figure 3.41. Netlist continues from Netlist 3.7 and continues in Netlist 3.9.

- The circuit is first linearized before applying AC analysis. The response computed by AC analysis is a simple linear function of the stimulus. If you double the stimulus, the response doubles. Nonlinear effects such as distortion and compression are not modeled in AC analysis. As a result, you are free to set the stimulus to any amplitude and phase you like. Typically it is set to have a magnitude of 1 and a phase of 0 so as to directly compute the transfer function of the circuit. Section 3.3.1 on page 54.
- XF analysis is like AC analysis, except that while AC analysis simultaneously computes the response of every node to a single

```
// Measure Common-Mode Characteristics

// Gain, DM to CM coupling, PSRR, Rout
cmXferFunctions (cout 0) xf start=1 stop=1G dec=10 \
    title="CM Transfer Functions to V(cout)"

// Gain, CM to DM coupling, Rin
dmEnableCommIn alter dev=Vic param=mag value=1
cmAC ac start=1 stop=1G dec=10 \
    title="CM Transfer Functions from Vic"
dmDisableCommIn alter dev=Vic param=mag value=1

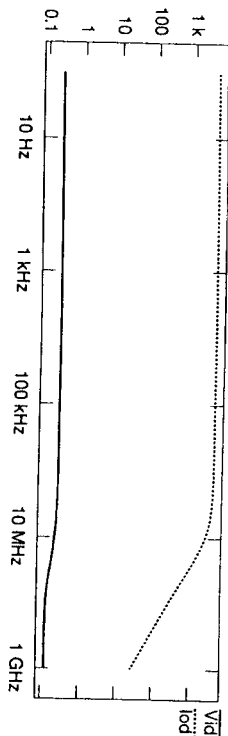
// Step response
cmEnablePulse alter dev=Vic param=type value=pulse
cmStepResponse tran stop=2us errpreset=conservative \
    title="CM Step Response"
cmDisablePulse alter dev=Vic param=type value=dc

// Noise
cmNoise (cout 0) noise start=1 stop=1G dec=10 \
    iprobe=Vic title="CM Noise"
```

Netlist 3.9: Common-mode measurements used to characterize the fully-differential amplifier shown in Figure 3.41. Netlist continues from Netlist 3.8.

- stimulus, XF analysis simultaneously computes the response at a single output to every stimulus. Section 3.3.2 on page 55.
- Noise analysis in SPICE is not accurate when circuits exhibit a significant amount of frequency conversion. Noise analysis should not be applied to mixers, oscillators, samplers, sample-and-holds, ADCs, DACs, switched-filters, etc. Section 3.3.4.6 on page 66.
- Simple-minded approaches to measuring feedback parameters, such as loop gain, can generate grossly inaccurate results when applied without great care. Approaches that open the loop

Differential-Mode Transfer Functions to V(dout)



Common-Mode Transfer Functions to V(cout)

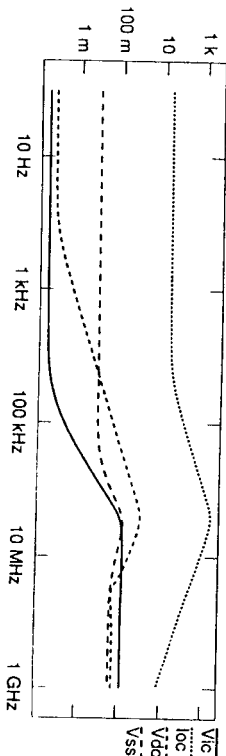
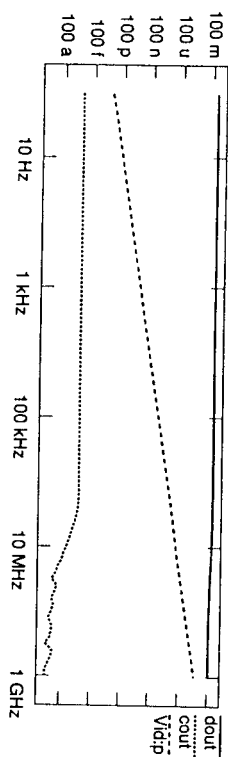


Figure 3.42: Results from the XF analyses of Netlists 3.8 and 3.9. The top figure shows the transfer functions to the differential-mode output. The differential-mode gain is V_{id} and the differential-mode output impedance is I_{od} . Conversion from common-mode signals (V_{ic} and I_{oc}) as well as coupling from power supplies (V_{dd} and V_{ss}) was negligible. The bottom figure shows the transfer functions to the common-mode output. The common-mode gain is V_{ic} and the common-mode output impedance is I_{oc} . Conversion from differential-mode signals (V_{id} and I_{od}) was negligible. Coupling from power supplies is V_{dd} and V_{ss} .

Differential-Mode Transfer Functions from Vid



Common-Mode Transfer Functions from Vic

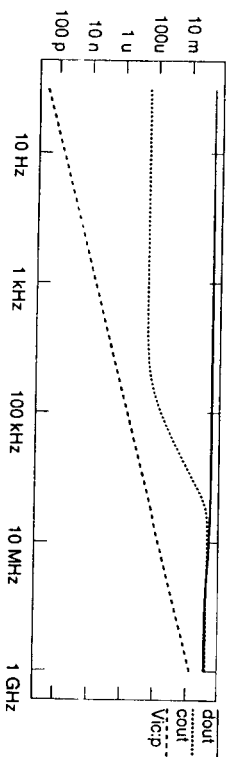


Figure 3.43: Results from the AC analyses of Netlists 3.8 and 3.9. The top figure shows the transfer functions from the differential-mode input. The differential-mode gain is d_{out} and the differential-mode input admittance is $V_{id,p}$. Conversion to the common-mode output is c_{out} . The bottom figure shows the transfer functions from the common-mode input. The common-mode gain is c_{out} and the common-mode input admittance is $V_{ic,p}$. Conversion to the differential-mode output is d_{out} .

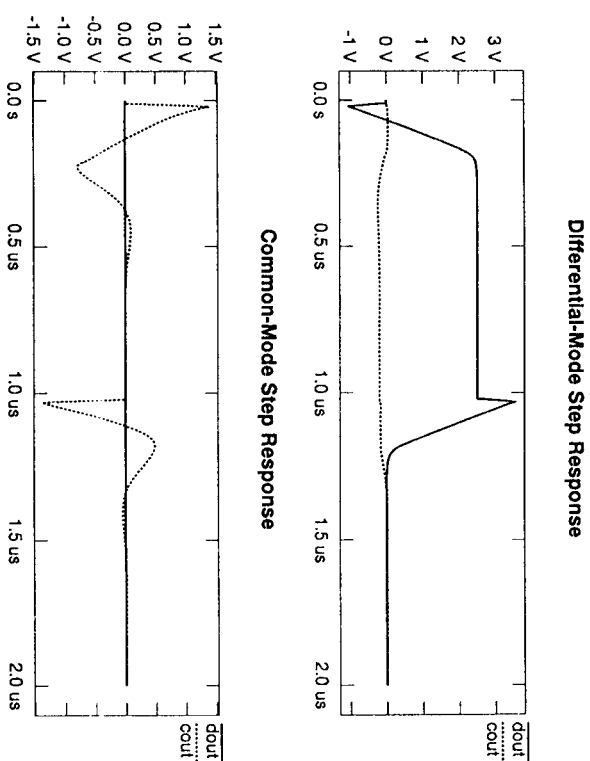


Figure 3.44: Results from the transient analyses of Netlists 3.8 and 3.9. The top figure shows the step response when a differential-mode step input is applied. The differential-mode response is *dout* and the coupling to the common-mode output is *cout*. The bottom figure shows step response when a common-mode step input is applied. The differential-mode response is *dout* and the coupling to common-mode output is *cout*.

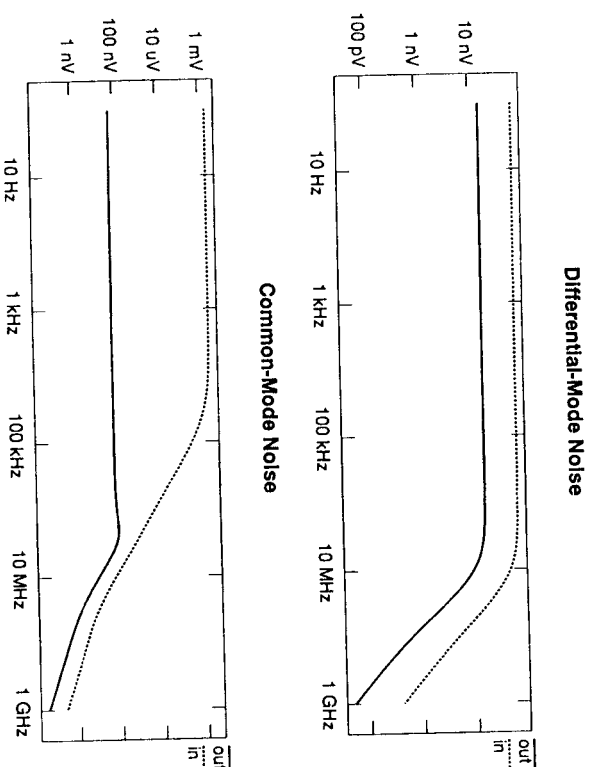


Figure 3.45: Results from the noise analyses of Netlists 3.8 and 3.9. The top figure shows the differential output and the differential input referred noise. The bottom figure shows the common-mode output and the common-mode input referred noise.

- When measuring the feedback parameters of the standard non-inverting opamp configuration, it is best to reposition the stimulus source to avoid generating common-mode signals, which contaminate the results. Section 3.4.1.3 on page 73.

- If only the characteristics of the opamp are needed, such as open loop gain, then the accuracy of the measurement is improved by replacing the feedback circuit with one that is ideal using controlled sources. Section 3.4.1.3 on page 79.
- Closed-loop approaches based on the 2-port parameters are applicable to almost all feedback amplifiers, even those that are not constructed from near ideal components. Section 3.4.1.4 on page 84.
- The characteristics of differential amplifiers are more easily measured if the differential- and common-mode signals are separated at the input and output using baluns. Section 3.4.5 on page 113.

Chapter 4

Transient Analysis

4.1 Introduction

This chapter is partitioned into three major sections that present the theory, the practice, and the application of transient analysis. The first section introduces some of the theoretical background and then proceeds to discuss many of the important artifacts that result from the approximations inherent in transient analysis. The middle of the chapter concentrates on the practical aspects of transient analysis, including how errors accrue and why some circuits are more sensitive to errors than others. Two approaches for choosing the time step used by the simulator to control errors are contrasted. The manner in which initial conditions are implemented is discussed. And, reasons are given for why simulators have convergence difficulties in transient analysis along with some suggestions on how to avoid convergence problems. The chapter concludes by presenting particular types of circuits that present issues or difficulties for transient analysis. In particular, suggestions are given that allow you to anticipate and avoid common problems with oscillators, sinusoidal circuits, charge-storage circuits, and power distribution circuits.

Certain parts of this chapter use mathematics that you are likely to have seen before, but which may be rusty. Feel free to simply skim forward until you reach material that is more comfortable. No parts of this chapter are indispensable, and you can always come back for a second, more determined, reading if you feel the material

is important.

4.2 Transient Analysis Theory

Transient analysis computes the response of a circuit as function of time. One cannot numerically solve a nonlinear differential equation such as

$$i(v(t)) + \frac{d}{dt}q(v(t)) + u(t) = 0 \quad (4.1)$$

$$v(0) = a \quad (4.2)$$

directly if for no other reason that the solution is a waveform (in other words, it is an infinite-dimensional continuum of points). The best one can hope for is to solve for a finite-dimensional approximation to the actual solution, such as finite sequence of points. In transient analysis, time is discretized and the solution is computed piecewise. In other words, the simulation interval is broken into small individual time steps, some simplifying approximation is made in order to evaluate the time-domain derivative, and the differential equation is solved over the span of one time step at a time. Typically, the simplifying assumption that is made is that the signal trajectory follows a low-order polynomial over a time step. Assuming that the signal trajectory is a polynomial allows the simulator to replace the time-derivative operator in the differential equation with a discrete-time approximation, and therefore allows us to evaluate the differential equation.

Consider the simple RC circuit shown in Figure 4.1 on page 134 and described by

$$\frac{v(t)}{R} + C \frac{d}{dt}v(t) = 0. \quad (4.3)$$

Assume that the solution trajectory is linear over one time step. The time derivative can be replaced by the simple forward Euler finite difference

$$\frac{dv(t_k)}{dt} \approx \frac{v(t_{k+1}) - v(t_k)}{t_{k+1} - t_k}. \quad (4.4)$$

Substituting into (4.3) results in

$$\frac{v(t_k)}{R} + C \frac{v(t_{k+1}) - v(t_k)}{t_{k+1} - t_k} = 0, \quad (4.5)$$

4.2. Transient Analysis Theory

131

which is rearranged to compute $v(t_{k+1})$ from $v(t_k)$.

$$v(t_{k+1}) = v(t_k) - \frac{t_{k+1} - t_k}{RC} v(t_k) \quad (4.6)$$

$$v(t_{k+1}) = v(t_k) \left(1 - \frac{h_k}{RC} \right) \quad (4.7)$$

where $h_k = t_{k+1} - t_k$. If the solution trajectory of (4.3) were linear, then (4.7) would be exact. However, the solution to (4.3) is a decaying exponential (assuming $RC > 0$) and so (4.7) is only an approximation to the true solution. The accuracy of the approximation depends on the accuracy of the assumption that the solution trajectory is linear over each time step. The smaller the time step, the smaller the difference between the true solution trajectory and a straight line over a time step, and therefore, the more accurate the results computed using (4.7).

In general, reducing the time step results in a low order polynomial better matching the true solution trajectory, and therefore in a more accurate approximation to the time derivative and more accurate results from the simulator.

4.2.1 Integration Methods

Replacing the time derivative operator with a discrete-time approximation and solving the resulting finite-difference equations one time point at a time starting from some initial condition is called numerically integrating the differential equation. The particular discrete-time approximation is referred to as the integration method. There are four integration methods commonly used in circuit simulation, forward Euler, backward Euler, trapezoidal rule, and the backward difference formulas (also known as Gear's methods). Forward and backward Euler are first order methods, meaning that the discrete-time approximation to the time-derivative operation is derived by assuming the solution trajectory is a first order (linear) polynomial over one time step. Trapezoidal rule is a second-order method, meaning that its approximation was derived by assuming that the solution trajectory is quadratic over each time step. The backward difference formulas or Gear's methods are a family of methods that are of any

order. However, only the first six orders are available in SPICE, and in general, only the first two methods are commonly used. Gear's first order method is identical to backward Euler.

The defining equations for the most common integration methods used in circuit simulation are now given assuming fixed time steps with the time step $h = t_{k+1} - t_k$.

Forward Euler

$$\frac{d}{dt}v(t_k) \approx \frac{1}{h}[v(t_{k+1}) - v(t_k)] \quad (4.8)$$

Backward Euler

$$\frac{d}{dt}v(t_{k+1}) \approx \frac{1}{h}[v(t_{k+1}) - v(t_k)] \quad (4.9)$$

Trapezoidal Rule

$$\frac{d}{dt}v(t_{k+1}) \approx \frac{2}{h}[v(t_{k+1}) - v(t_k)] - \frac{d}{dt}v(t_k) \quad (4.10)$$

Second-Order Backward Difference Formula (Gear2)

$$\frac{d}{dt}v(t_{k+1}) \approx \frac{3}{2h}v(t_{k+1}) - \frac{2}{h}v(t_k) + \frac{1}{2h}v(t_{k-1}) \quad (4.11)$$

Applying backward Euler discretization to (4.1) gives

$$i(v_{k+1}) + \frac{1}{h}(q(v_{k+1}) - q(v_k)) + u_{k+1} = 0. \quad (4.12)$$

If h is chosen small enough for the discrete approximation to be accurate, and if (4.12) is solved repetitively from the initial condition $v_0 = a$, then $v_k \approx v(t_k)$. Carefully choosing the time step is important to achieving an accurate result. Virtually all circuit simulators automatically vary the size of the time step in order to achieve an accurate result with the fewest possible time steps. All simulators also accept some input from the user on how to choose time steps, such as the desired level of accuracy or the maximum step size.

4.2.2 Characteristics of the Integration Methods

Trapezoidal rule and the second-order backward-difference formula (Gear2) are the most heavily used integration methods in circuit simulation. Higher order backward-difference formula are also available, but are rarely used. Backward Euler is also used by SPICE in certain circumstances, such as on the first time step and on break points (break points are generated at each corner in an input waveform). Forward Euler is used in some timing simulators, but not in circuit simulators such as SPICE.

For a fixed time step, the most accurate of these methods in a local sense¹ is the trapezoidal rule, followed by Gear2. However, circuit simulators automatically control the time step to maintain an acceptable level of error, so it is not possible to say in advance which method is more accurate for a particular circuit, but the trapezoidal rule would typically allow larger time steps and so require a shorter run time. Unfortunately, trapezoidal rule is overly sensitive to errors made on previous time steps, thus it is not usually the best choice when *reltol* is loose.

Backward Euler and the trapezoidal rule are one-step methods, meaning they compute the value of the current time point using the value of the one immediate predecessor. N^{th} order backward-difference formulas (GearN) are N -step methods, meaning they need N predecessors. Thus, Backward Euler and the trapezoidal rule adapt faster to abrupt signal changes, with Backward Euler adapting even faster than the trapezoidal rule (which is one reason why backward Euler is used after every break-point). When computing signals with many abrupt transitions, the N -step methods become more and more inefficient as N increases.

The higher-order backward-difference formulas are efficient when tolerances are tight or when computing very smooth waveforms. For example, circuits that generate sinusoidal or near sinusoidal signals

¹An integration method is accurate in a local sense if it is accurate on infinitely small time steps. This is a useful measure because it does not depend on the circuit being simulated (as long as the circuit satisfies certain mild assumptions). How integration methods do on large time steps is very difficult to say because it depends strongly on the behavior of the circuit and the solution.

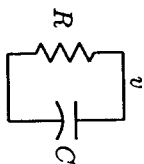


Figure 4.1: Linear RC circuit used to explore characteristics of various integration methods.

are good candidates for the higher-order backward-difference formulas because the higher-order polynomials allow a larger time step without sacrificing accuracy. However, the higher-order Gear methods can exhibit stability problems on lightly damped circuits (see Section 4.2.2.3 on page 150). In addition, the higher-order Gear methods have never been heavily used and so there is some risk of tripping over a bug when using them.

4.2.2.1 Stiff Circuits

The characteristics of the various integration methods can be further explored by applying them to a simple test case. Consider the RC circuit shown in Figure 4.1. This circuit linear has a single real pole at $\lambda = -1/RC$. Writing KCL gives

$$\frac{1}{R}v(t) + C\dot{v}(t) = 0 \quad (4.13)$$

$$\dot{v}(t) = -\frac{1}{RC}v(t) \quad (4.14)$$

$$\dot{v}(t) = \lambda v(t) \quad (4.15)$$

Applying backward Euler to discretize time and assuming uniform time step $h = t_{k+1} - t_k$ gives

$$\frac{1}{h}[v(t_{k+1}) - v(t_k)] = \lambda v(t_{k+1}) \quad (4.16)$$

Rewriting to calculate $v(t_{k+1})$ as a function of $v(t_k)$,

$$v(t_{k+1}) = \frac{v(t_k)}{1 - \lambda h} \quad (4.17)$$

Let $\sigma = \lambda h$ be the pole frequency normalized by the time step.

$$v(t_{k+1}) = \frac{v(t_k)}{1 - \sigma} \quad (4.18)$$

Figure 4.2 on the next page shows response computed with backward Euler of the RC circuit to a non-zero initial condition for various values of σ . Similarly, Figures 4.3, Figures 4.4, and 4.5 show the responses computed by the trapezoidal rule, Gear2, and forward Euler. Large values of σ represent time-constants that are very small compared to the time step. Such circuits are considered *stiff* and can cause problems for the integration methods.

Notice that for stiff circuits, backward Euler and Gear2 perform well whereas the trapezoidal rule exhibits strong point-to-point ringing. Also notice that Gear2 exhibits a slight amount of overshoot for time constants that are slightly smaller than the time step. Lastly, notice that Figure 4.6 on page 140 shows that forward Euler is unstable on stiff circuits. Backward Euler, the trapezoidal rule, and Gear2 are stiffly stable, meaning that they are stable on stiff circuits.

Physical systems in general, and electrical circuits in particular, have many very high frequency poles that do not play an important role in the response of the system, usually because they are never excited or any transients due to those poles decay very quickly. Thus, the time steps taken by the simulator would normally be much larger than the time-constants associated with these poles, and so the circuits would be stiff. In order to apply forward Euler to such a circuit, it would be necessary to shrink the time step to the point where it was smaller than the smallest time constant present. For this reason, applying forward Euler on a stiff circuit would be much less efficient than applying a stiffly stable method. In other words, the time step taken by a stiffly stable method is bounded to assure accuracy. The time step for forward Euler, and other methods that are not stiffly stable, is bounded to assure both accuracy and stability. Circuits tend to be very stiff, and so the stability constraint often requires

Backward Euler Zero Input Response

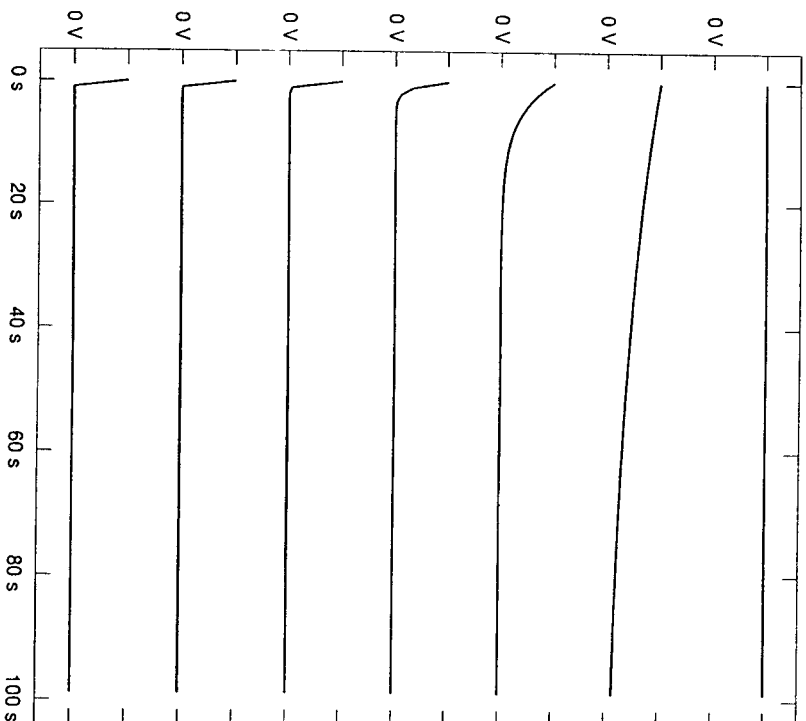


Figure 4.2: Response computed with backward Euler of the RC circuit to a non-zero initial condition for $\sigma = 0, -0.02, -0.2, -2, -20, -200$, and $-\infty$ (top to bottom). σ is the pole frequency λ normalized by the time-step h .

Trapezoidal Zero Input Response

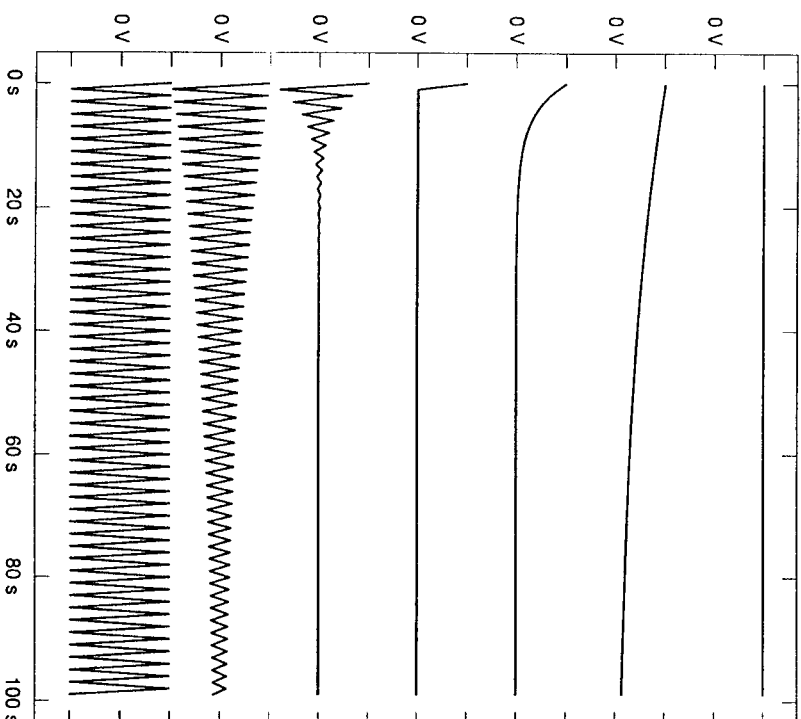


Figure 4.3: Response computed with the trapezoidal rule of the RC circuit to a non-zero initial condition for $\sigma = 0, -0.02, -0.2, -2, -20, -200$, and $-\infty$ (top to bottom). σ is the pole frequency λ normalized by the time-step h .

2nd Order Gear Zero Input Response

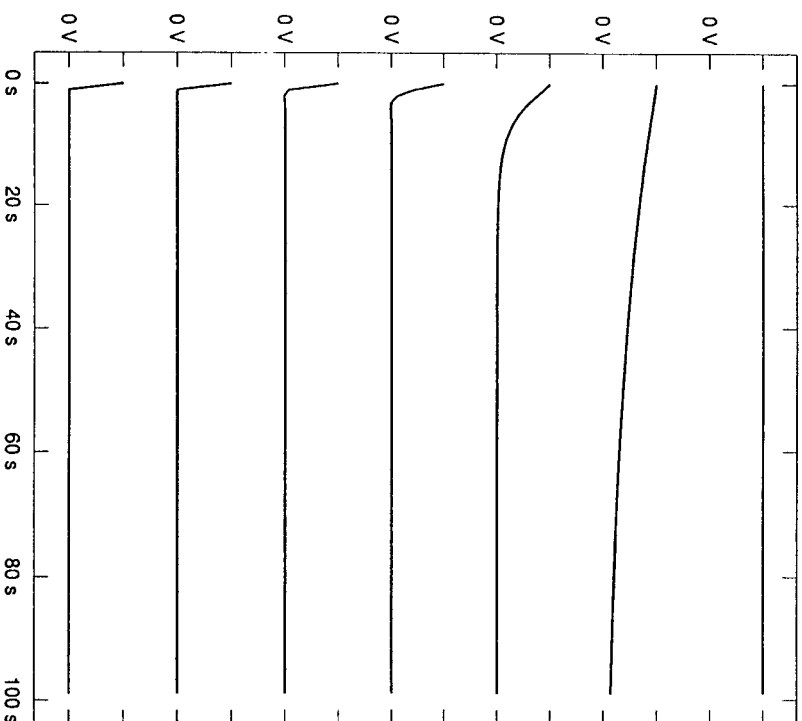


Figure 4.4: Response computed with Gear's 2nd-order backward-difference formula (Gear2) of the RC circuit to a non-zero initial condition for $\sigma = 0, -0.02, -0.2, -2, -20, -200$, and $-\infty$ (top to bottom). σ is the pole frequency λ normalized by the time-step h .

Forward Euler Zero Input Response

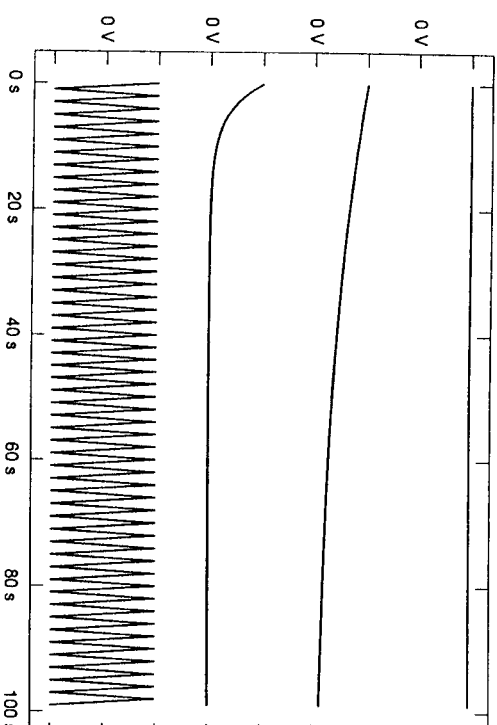


Figure 4.5: Response computed with forward Euler of the RC circuit to a non-zero initial condition for $\sigma = 0, -0.02, -0.2$, and -2 (top to bottom). σ is the pole frequency λ normalized by the time-step h . Forward Euler is unstable for $\sigma < -2$.

a much smaller time step than the accuracy constraint. For these reasons, integration methods that are not stiffly stable are rarely used in circuit simulators. Some timing simulators do use forward Euler, and so are not suitable for use on stiff circuits.

4.2.2.2 Mapping the s-Plane to the z-Plane

Consider the linear single real pole test case described above in (4.15). More information could be taken from that example if the pole was allowed to be complex. The integration method is used to map a differential equation into a difference equation. If the time step is

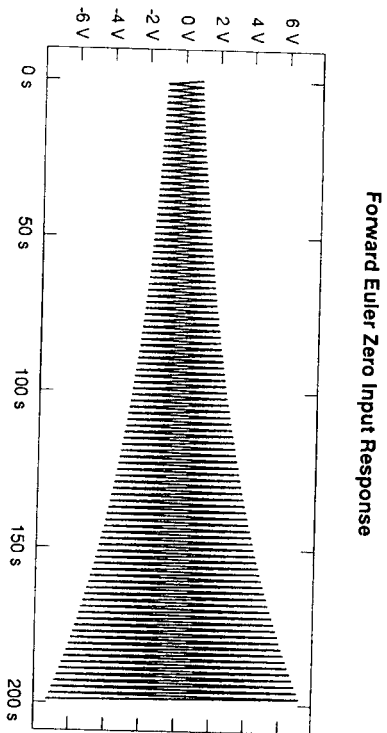


Figure 4.6: Response computed with forward Euler of the RC circuit to a non-zero initial condition for $\sigma = -2.01$. Forward Euler is unstable for $\sigma < -2$.

constrained to be uniform, it is useful to determine how the poles of the continuous time differential equation are mapped to the discrete-time difference equation.

Backward Euler Once (4.15) was discretized using backward Euler with a uniform time step, it could be rewritten as (4.18). Applying the z -transform converts this equation into the frequency domain by replacing the unit delays with multiplications by z^{-1} .

$$V = \frac{Vz^{-1}}{1 - \sigma} \quad (4.19)$$

$$z = \frac{1}{1 - \sigma} \quad (4.20)$$

This equation maps poles in the s -plane ($\sigma = h\lambda$) into poles in the z -plane. It is shown in Figure 4.7 on page 142 mapping the entire left-half of the s -plane into the z -plane. Recall that differential equation is stable if its poles are contained within the left-half of the s -plane

and that a difference equation is stable if its poles are contained within the unit disk of the z -plane. Notice that the entire left-half of the s -plane maps inside the unit circuit of the z -plane. Thus, any stable differential equation that is mapped into a difference equation by backward Euler is also stable. However, notice that also a portion of the right-half of the s -plane is also mapped into the unit circuit of the z -plane. This means that some unstable differential equations are also mapped into stable difference equations by backward Euler. In addition, the imaginary axis of the s -plane is mapped to the interior of the unit disk, which implies that backward Euler adds loss to lossless resonators. In summary, while the process of converting a continuous time differential equation into a discrete-time difference equation necessarily results in changes in the pole frequencies, backward Euler performs the mapping by making the system more stable. The smaller the frequency σ , the less warping occurs. This is equivalent to saying that the undesirable artifacts that occur when discretizing with backward Euler become less noticeable as the time step is decreased.

Trapezoidal Rule Discretizing (4.15) with the trapezoidal rule and applying the z -transform results in a one-to-one map from s - to z -planes.

$$v(t_{k+1}) = \frac{2 + \sigma}{2 - \sigma} v(t_k) \quad (4.21)$$

$$z = \frac{2 + \sigma}{2 - \sigma} \quad (4.22)$$

Figure 4.8 on page 143 shows the left-half plane of the s -plane mapped into the z -plane by the trapezoidal rule. Note that the imaginary axis of the s -plane is mapped to the unit circle, that the left-half of the s -plane maps into the interior of the unit disk, and that the right-half of the s -plane maps into the exterior of the unit disk. Thus stability is preserved by the trapezoidal rule. Stable differential equations are mapped to stable difference equations and unstable differential equations are mapped to unstable difference equations. Frequency warping occurs, but it is less severe than with backward Euler.

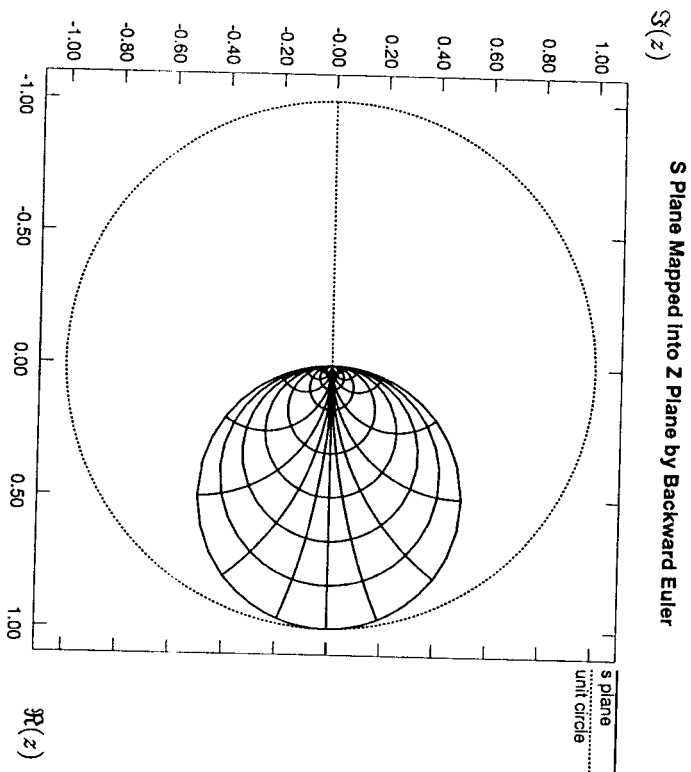


Figure 4.7: Left half of s -plane mapped into z -plane by backward Euler. The real and imaginary axes of the s -plane are marked at $\sigma = 0, \pm 0.2, \pm 0.5, \pm 1.0, \pm 2.0, \pm 5.0, \pm 10.0, \pm 20.0$, and $\pm \infty$. $0 + 0j$ in the s -plane maps to $1 + 0j$ in the z -plane and $-\infty + 0j$ in the s -plane maps to $0 + 0j$ in the z -plane.

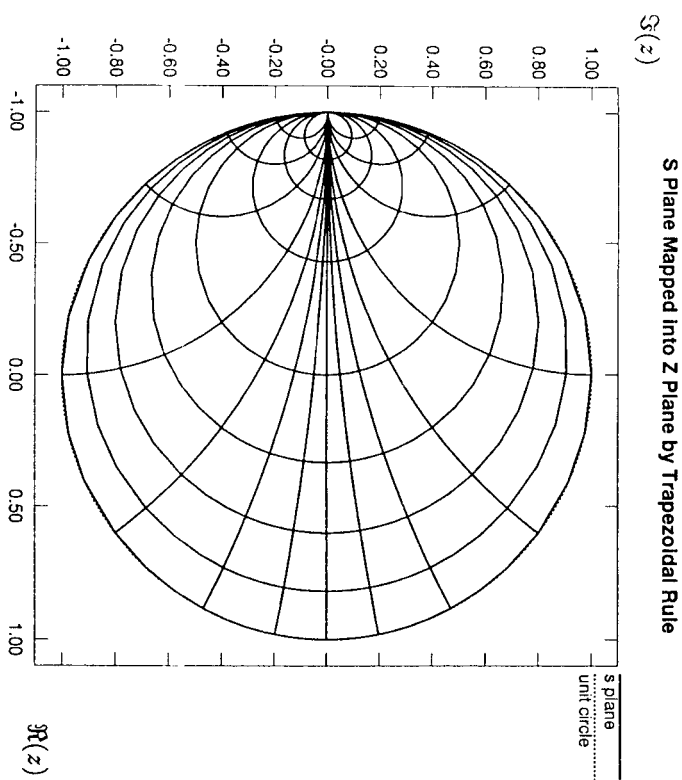


Figure 4.8: Left half of s -plane mapped into z -plane by the trapezoidal rule. The real and imaginary axes of the s -plane are marked at $\sigma = 0, \pm 0.2, \pm 0.5, \pm 1.0, \pm 2.0, \pm 5.0, \pm 10.0, \pm 20.0$, and $\pm \infty$. $0 + 0j$ in the s -plane maps to $1 + 0j$ in the z -plane and $-\infty + 0j$ in the s -plane maps to $-1 + 0j$ in the z -plane.

Gear's Second Order Backward Difference Formula Applying Gear2 to discretize (4.15) and applying the z -transform results in a one-to-two map from s - to z -planes.

$$v(t_{k+1}) = \frac{4v(t_k) - v(t_{k-1}))}{3 - 2\sigma} \quad (4.23)$$

$$V(z) = \frac{4zV(z) - z^2V(z)}{3 - 2\sigma} \quad (4.24)$$

$$z = \frac{-2 \pm \sqrt{1 + 2\sigma}}{2\sigma - 3} \quad (4.25)$$

Each pole in the s -plane maps into two poles in the z -plane because Gear2 is a two step method. Figure 4.9 and Figure 4.10 shows the left-half plane of the s -plane mapped to the position of each of the two poles in the z -plane by Gear2. Notice that the entire left-half of the s -plane maps inside the unit circuit of the z -plane for both poles. Thus, any stable differential equation that is mapped into a difference equation by Gear2 is also stable. However, notice that also a portion of the right-half of the s -plane is also mapped into the unit circuit of the z -plane. This means that some unstable differential equations are also mapped into stable difference equations by Gear2. In addition, the imaginary axis of the s -plane is mapped to the interior of the unit disk, which implies that Gear2 adds loss to lossless resonators.

The real axis of the s -plane behaves in an interesting way. A single pole at $s = 0 + 0j$ maps to two poles, one at $z = 1 + 0j$ and the other at $z = 1/3 + 0j$. As the pole moves along the real axis in the s -plane, the pair of poles move together until they become coincident at $s = 1/2 + 0j$. As the pole continues to move left along the real axis in the s -plane, the pair of poles leave the real axis in opposite directions and travel along a circle with a radius of $1/4$ centered at $z = 1/4 + 0j$. This shows that some real poles in the s -plane map to complex conjugate pairs of poles in the z -plane. Thus, a real pole can cause over-shoot when Gear2 is used.

Forward Euler Applying forward Euler to discretize (4.15) and applying the z -transform gives

$$z = 1 + \sigma \quad (4.26)$$

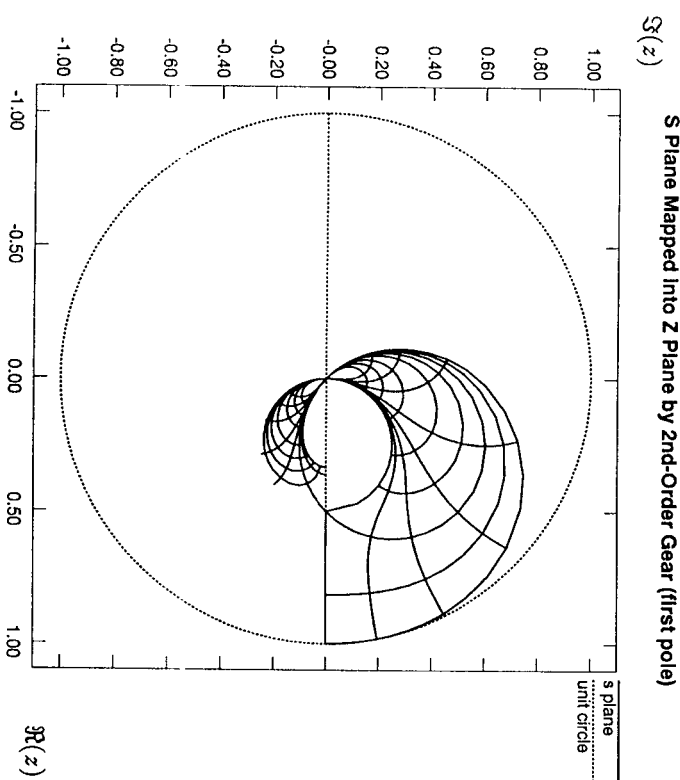


Figure 4.9: Left half of s -plane mapped into the position of the first pole in the z -plane by the Gear2. The real and imaginary axes of the s -plane are marked at $\sigma = 0, \pm 0.2, \pm 0.5, \pm 1.0, \pm 2.0, \pm 5.0, \pm 10.0, \pm 20.0$, and $\pm \infty$. $0 + 0j$ in the s -plane maps to $1 + 0j$ in the z -plane and $-\infty + 0j$ in the s -plane maps to $0 + 0j$ in the z -plane.

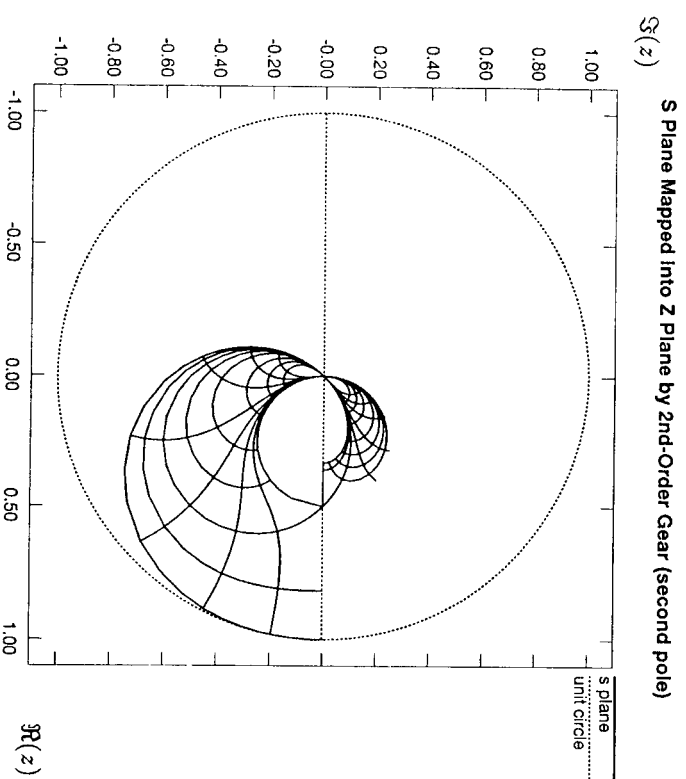


Figure 4.10: Left half of s -plane mapped into the position of the second pole in the z -plane by the Gear2. The real and imaginary axes of the s -plane are marked at $\sigma = 0, \pm 0.2, \pm 0.5, \pm 1.0, \pm 2.0, \pm 5.0, \pm 10.0, \pm 20.0$, and $\pm \infty$. $0 + 0j$ in the s -plane maps to $1/3 + 0j$ in the z -plane and $-\infty + 0j$ in the s -plane maps to $0 + 0j$ in the z -plane.

Figure 4.11 on the next page shows the left-half plane of the s -plane mapped into the z -plane by forward Euler. Note that the imaginary axis of the s -plane is mapped to the exterior of the unit disk. Thus, unstable differential equations are mapped to unstable difference equations, but some stable differential equations are mapped to unstable difference equations and imaginary poles in the s -plane that result from lossless resonators map to unstable difference equations.

4.2.2.3 Artificial Numerical Damping

As shown in the previous section, some integration methods exhibit an artificial numerical damping that manifests itself by artificially increasing the damping in the circuit. When simulating circuits containing low-loss (high- Q) resonators such as oscillators and filters, these integration methods make it appear as if the Q of the circuit is less than it actually is. Under certain conditions, the damping may be sufficient to make a strongly unstable circuit appear to be stable (see Section 4.4.3 on page 219).

The trapezoidal rule does not exhibit numerical damping. However, Gear's second-order backward difference formula does exhibit some damping and backward Euler exhibits heavy damping. Thus, backward Euler, and to a lesser degree, Gear2, appears to increase the loss, or decrease the Q , of lowloss resonators. This is shown in Figure 4.12 on page 149, which shows the response of the lossless LC resonator of Figure 4.20 on page 161 to a nonzero initial condition computed with a fixed time step using the trapezoidal rule, Gear2, and backward Euler. This effect decreases with smaller time steps. So if artificial numerical damping is a problem, try using the trapezoidal rule. If that is not desirable, reduce the step size by tightening `reltol`.

Artificial numerical damping is a problem when looking for ringing on inductive power supply buses and clock lines because the ringing is small relative to the DC operating values. The large signals distract the simulator from the small ringing, which cause the simulator to take large time steps, increasing the damping to unacceptably large levels. This situation is described more fully in Section 4.4.7 on

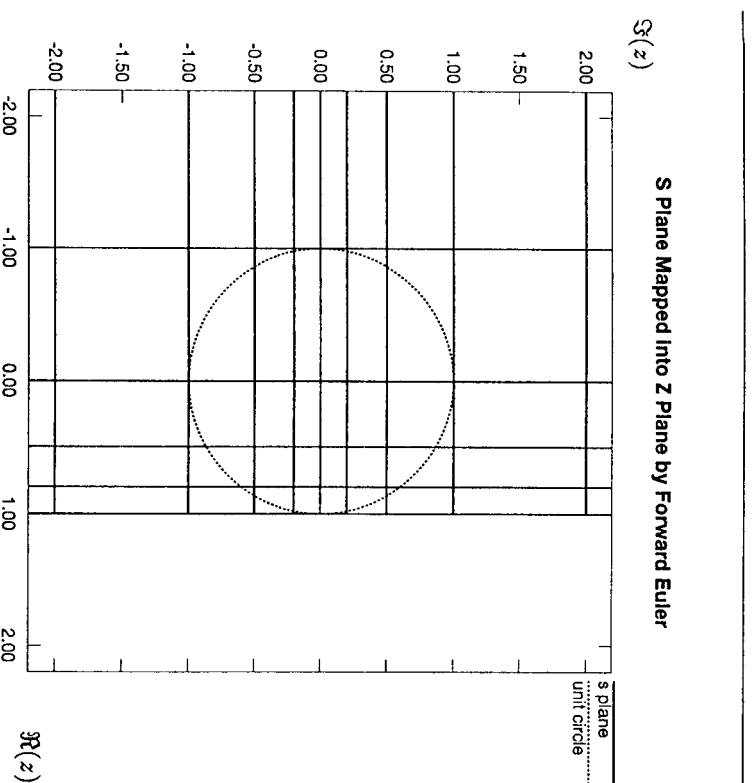


Figure 4.11: Left half of s -plane mapped into z -plane by forward Euler. The real and imaginary axes of the s -plane are marked at $\sigma = 0, \pm 0.2, \pm 0.5, \pm 1.0$, and ± 2.0 . $0 + 0j$ in the s -plane maps to $1 + 0j$ in the z -plane and $-2 + 0j$ in the s -plane maps to $-1 + 0j$ in the z -plane.

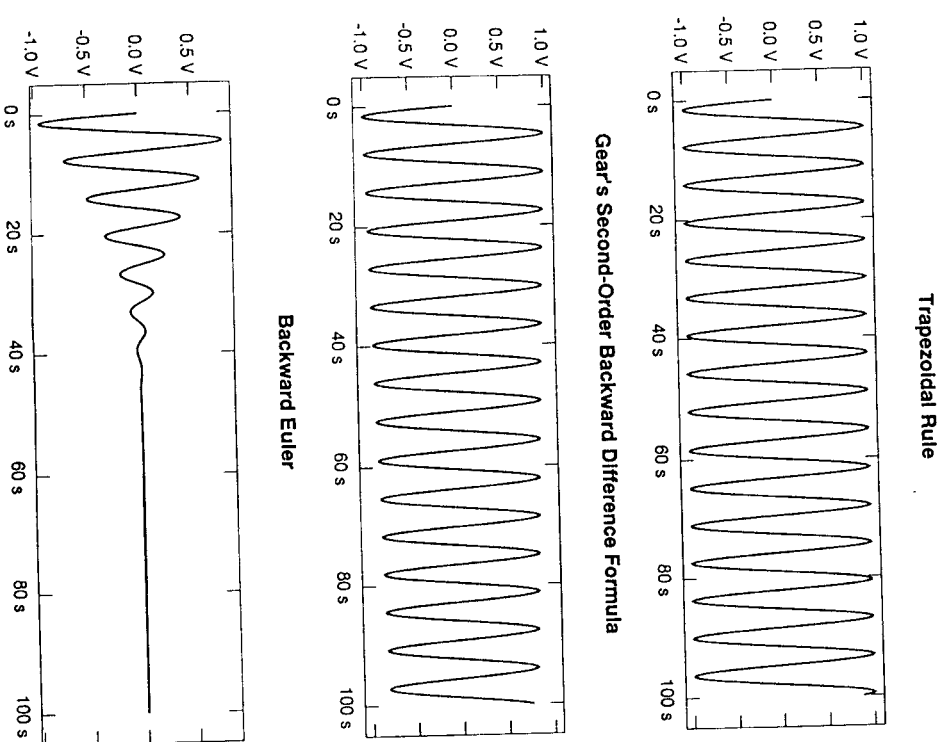


Figure 4.12: Response of LC resonator to nonzero initial condition computed with fixed time step using trapezoidal rule (top), Gear's second order backward-difference formula (middle), and backward Euler (bottom). Trapezoidal rule exhibits no artificial numerical damping, Gear2 exhibits some damping, and backward Euler exhibits heavy damping.

page 234.

Unlike first- and second-order Gear methods, which always add a certain amount of artificial damping, the higher-order Gear methods can actually reduce the amount of damping on some circuits. This occurs on circuits with complex poles near or on the $j\omega$ axis of the s -plane. In fact, those poles can even be mapped to outside the unit disk of the z -plane. Thus, stable complex poles in the s -plane can be mapped to unstable poles in the z -plane by the higher-order Gear methods. This problem becomes more severe as the order of the method increases, as shown in Figure 4.13 on the next page.

4.2.2.4 Ringing

On stiff circuits, the trapezoidal rule generates solutions where the sign of the error alternates on every time step. The solution is generally within specified limits, but exhibits a ringing, often referred to as *trapezoidal ringing*. A typical case of ringing in SPICE is shown in Figure 4.14 on page 152. Other examples are shown in Figures 4.3 on page 137, 4.16 on page 154, 4.19 on page 159, and 4.39 on page 185. Thus, the trapezoidal rule can be a poor choice if the solution were to be Fourier transformed because of the built-in noise floor if ringing is present. It is also a poor choice if currents through capacitors or voltages across inductors are of interest. The problem is that the error on these waveforms is controlled only loosely because capacitor currents and inductor voltages are not state variables. And because these quantities are time derivatives of the voltage across capacitors and the current through inductors, any error in these state variables are amplified. Thus, trapezoidal ringing can be quite large on capacitor currents and inductor voltages, particularly if small time steps are being used. The trapezoidal rule also has problems if all of the capacitors on a node disappear (the total capacitance becomes zero for some but not all biases), which can sometimes happen when capacitors are nonlinear. This can be a problem with gate capacitances on MOSFETs unless the side-wall and overlap capacitances are given. Since capacitances never disappear in reality, this can be considered more of a modeling problem than a simulator problem.

Occasionally, trapezoidal rule ringing is so strong and so fast that

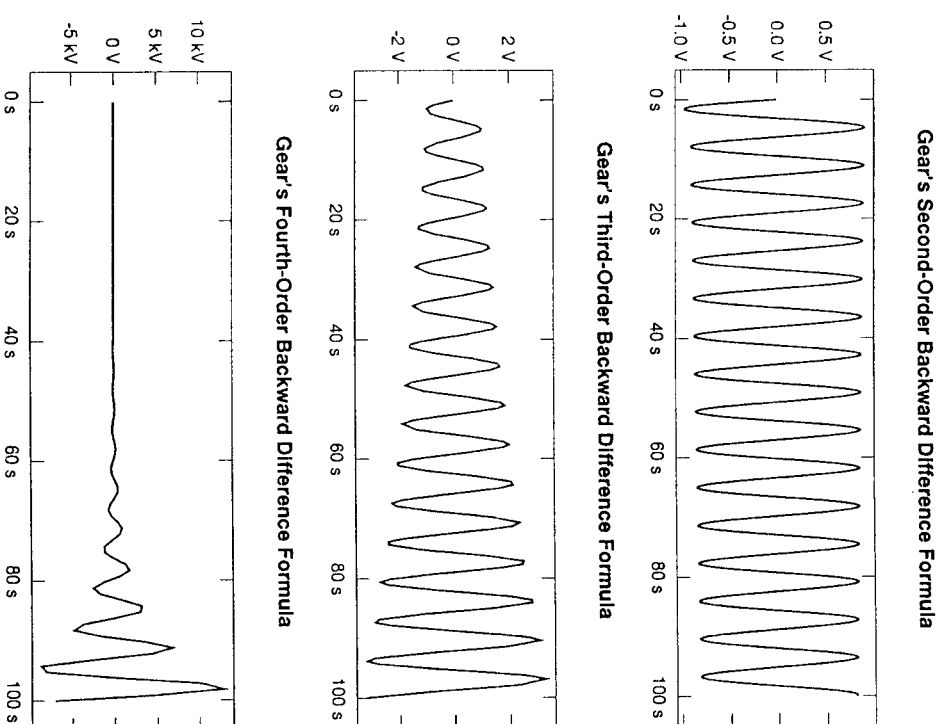


Figure 4.13: Response of lossless LC resonator to nonzero initial condition computed with fixed time step using 2nd-order (top), 3rd-order (middle), and 4th-order Gear (bottom). Gear's higher-order methods are unstable on high- Q circuits unless the time step is sufficiently small.

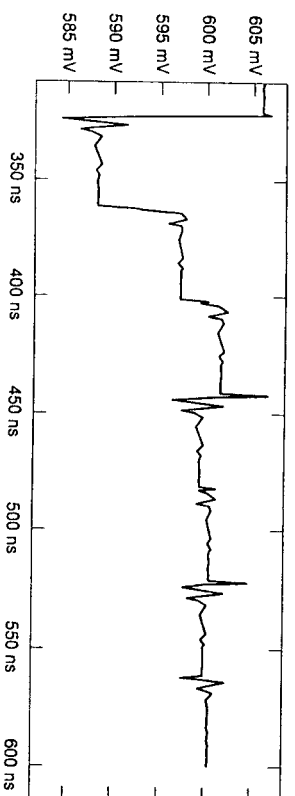


Figure 4.14: A waveform computed by SPICE that exhibits trapezoidal ringing.

it forces the simulator to take infinitesimal time steps. If transient analysis is taking much longer than anticipated, try killing the simulation and using rerunning with Gear2. To determine if this is the case on large circuits, where it is very difficult to save and inspect all node voltage waveforms, plot the supply current waveforms. If any node voltage in the circuit is exhibiting pathological trapezoidal rule ringing, then it most likely shows up in the supply current as well. If extremely fast trapezoidal rule ringing is found on the supplies, switch to Gear2.

In general, if trapezoidal ringing is a problem, use Gear2. However, realize that while switching to Gear2 makes waves prettier, they may be no more accurate. The poor time-step control that allowed noticeable trapezoidal rule ringing is still there taking the same large time steps with Gear. While Gear2 gets rid of the artifact, it may not make the solution any more accurate.

4.2.2.5 Capacitor Currents and Inductor Voltages

In general, circuit simulators produce less accurate waveforms for capacitor currents and inductor voltages than for capacitor voltages and inductor currents. This section discusses capacitor currents, though everything said applies equally well to inductor voltages.

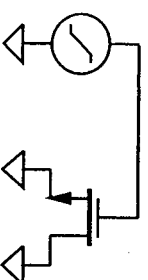
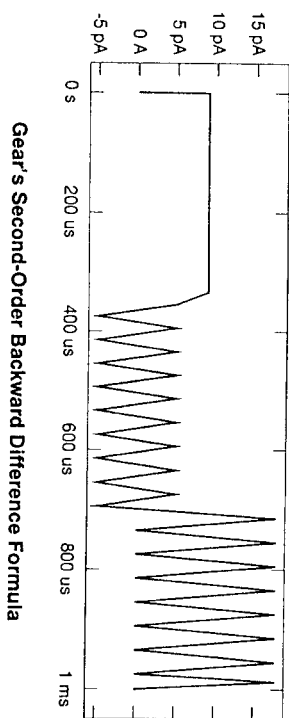


Figure 4.15: Circuit that measures the current into a MOS capacitor.

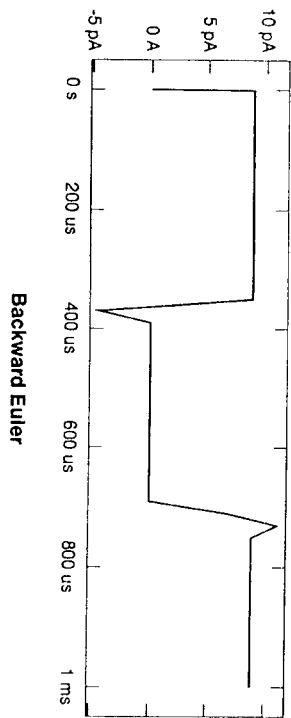
An unavoidable reason that capacitor currents are computed less accurately than capacitor voltages is that capacitor currents are the derivative of capacitor voltages. Any short duration disturbance on a capacitor voltage is amplified by the derivative operator, resulting in a much larger disturbance on the capacitor current. The simulator worsens the situation by ignoring capacitor currents when choosing the time step. Unlike the voltage across a capacitor, errors in the current through a capacitor do not accumulate, and therefore the currents through capacitors are ignored by the time-step control algorithm. Finally, the integration methods are being applied in a slightly different manner when computing the current through capacitors. Rather than being involved with solving implicit differential equations, they instead are computing simple explicit derivatives. These methods do a much better job of solving differential equations than they do computing simple derivatives.

A circuit that exposes some of the flaws exhibited by the integration methods when computing simple derivatives is shown in Figure 4.15. The circuit consists of a voltage source driving the gate of a MOSFET with a ramp. The source, drain, and bulk are grounded. The signal of interest is the current through the gate. The current is computed using the trapezoidal rule, Gear2, and backward Euler and is shown in Figure 4.16 on the following page. The gate voltage starts out low enough so that the channel is present in the MOSFET. As the voltage ramps upward, a point is reached where the channel depletes and the gate capacitance drops to near zero very quickly, which causes the

Trapezoidal Rule



Gear's Second-Order Backward Difference Formula



Backward Euler

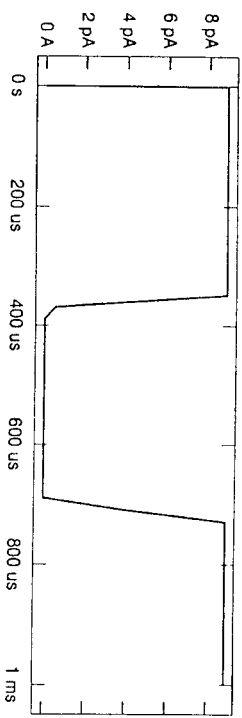


Figure 4.16: Current through MOS capacitor of circuit in Figure 4.15 measured using trapezoidal rule, Gear's second-order backward-difference formula, and backward Euler. These results are also representative of what occurs when computing the voltage across an inductor.

4.2. Transient Analysis Theory

155

gate current to drop to near zero. Further along in the ramp, the channel forms again and the gate capacitance and the gate current return to their original values.

The first problem exhibited in the waveforms of Figure 4.16 is that the abrupt changes in the current occur asynchronously. Unlike when a circuit responds to an abrupt change in a stimulus, the simulator has no way of knowing precisely when the abrupt change in the current occurs. Since the simulator does not put a time point precisely on the corner of the transition, as it would if the transition resulted from a pulsed source, the corner appears to be knocked off. This is most visible in the waveform computed by backward Euler and trapezoidal rule, but it can occur with any integration method. There is no direct solution to this problem, however tightening `reltol` does reduce the effect.

The current through the gate is

$$i(t) = \dot{q}(v(t)). \quad (4.27)$$

Applying the trapezoidal rule gives

$$i(t_{k+1}) = \frac{2}{h} [q(v(t_{k+1})) - q(v(t_k))] - i(t_k). \quad (4.28)$$

Assume that the voltage across the capacitor is unchanging, which implies the current through the capacitor should be zero. The equation simplifies down to $i(t_{k+1}) = -i(t_k)$. If there is any error in the current on the previous time point, then the same error returns on the new time point with its sign reversed. Even if the current is changing, the result is the same. Any error in current at a time point is retained with alternating sign in all subsequent time points. All that is needed to start the point-to-point ringing is a single error in the current. In Figure 4.16, as in most cases, the error results when there is a discontinuous jump in the current. SPICE and Spectre both switch to backward Euler for one step after a break point, which is when most discontinuous jumps in current waveforms occur, thereby eliminating most occurrences of ringing in capacitor currents. However, in the circuit of Figure 4.15, the discontinuity is created by the characteristics of the MOSFET rather than by an independent source. The discontinuity does not occur at a break point and so the ringing is not avoided. To avoid trapezoidal ringing, use Gear2.

Applying Gear2 to (4.27) gives

$$i(t_{k+1}) = \frac{3}{2h}q(v(t_{k+1})) - \frac{2}{h}q(v(t_k)) + \frac{1}{2h}q(v(t_{k-1})). \quad (4.29)$$

Whenever there is an abrupt change in the slope in the charge waveform, the current is overstated by 50% because of the factor of 3/2 on the leading charge term. This error is of a different nature than the one that plagues trapezoidal rule. Trapezoidal rule is accurate until an error occurs, and once an error has occurred, trapezoidal rule retains the error forever. Gear2 simply accentuates the high frequencies. Thus, the leading edge of a step, which has considerable energy at high frequencies, is accentuated and the results computed by the simulator overshoot.

No integration method is without faults, and when computing capacitor currents, the fault of backward Euler is that it adds a small amount of delay to the results. In Figure 4.16 the delay is manifested as a longer transition time. On circuits that generate sinusoidal currents, the delay manifests itself as a slight phase shift. The amount of delay is roughly equal to half a time step.

Generally, simulators do not use backward Euler on every time step. Instead, they take a step of backward Euler occasionally, such as at break points. Thus, the delay exhibited by backward Euler is only present on isolated time points. The significance of this delay is shown with the help of the circuit shown in Figure 4.17. V_1 drives C_1 with a sinusoidal voltage, which should result in a sinusoidal current. The current computed by the simulator using the Gear2 is shown in Figure 4.18 on page 158. The bumps on the current waveforms result from the simulator switching the integration method from Gear2 to backward Euler for one time point after a break point. The break points are generated by V_2 at each corner of its pulse train. While V_2 is in a circuit that is completely disconnected from the circuit containing C_1 , its presence causes the simulator to use backward Euler on isolated time points for the entire circuit.

A similar problem occurs when the simulator switched from the trapezoidal rule to backward Euler. The circuit of Figure 4.17 is used again, except V_2 now generates a delayed sine wave rather than a pulse train. V_2 generates one break point at the point where the

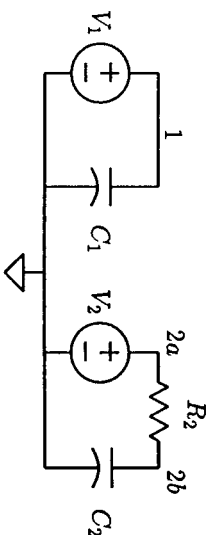


Figure 4.17: Simple test circuit that shows the effect of switching between integration methods.

sine wave begins. Thus, the simulator switches to backward Euler only once. The one step of backward Euler creates an error big enough to cause undamped trapezoidal rule ringing in the current of V_1 , as shown in Figure 4.19.

4.2.3 Transient Analysis Accuracy

While transient analysis suffers from the same error mechanisms as DC analysis, and even adds new mechanisms, it is the way in which errors accumulate that make transient analysis fundamentally different from other analyses. Unlike other analyses, the solution computed at a particular point in a transient analysis strongly depends on the solution computed on the previous step. Thus, errors propagate from one step to the next, and how they propagate becomes critically important. It is desirable that the effect of an error made on one step fades on succeeding steps, but this is not guaranteed. In some cases, the effect of the error actually gets larger on later steps. Whether the effect of the error shrinks or grows is determined only by the circuit, not the simulator. Furthermore, the simulator cannot know if errors dissipate or accumulate in a particular circuit, and so cannot compensate for this phenomenon. Simulating a circuit where errors dissipate with loose tolerances generally gives accurate results. In fact, tolerances can generally be loosened without de-

Voltages Waveforms of Capacitor Current Test Circuit

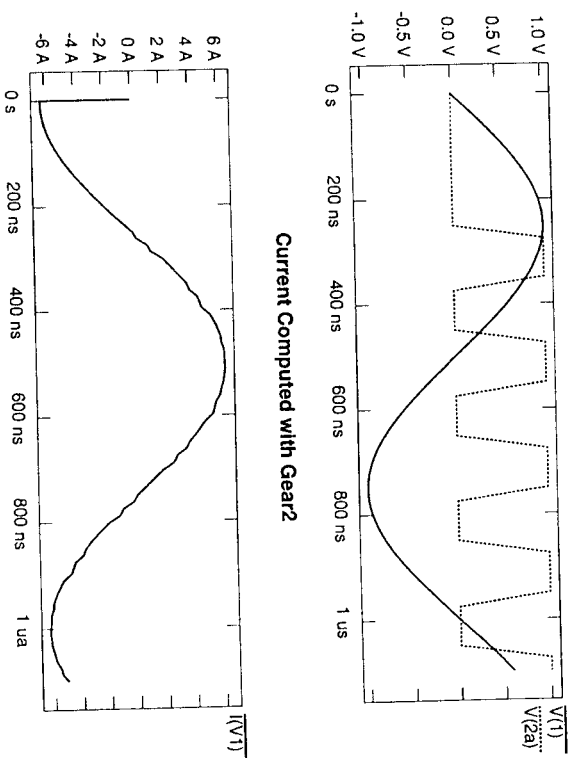


Figure 4.18: Signals computed for Figure 4.17 on the page before using Gear2. The small bumps that occur on current I_{V1} results whenever V_2 generates a break point. The break point forces a switch from Gear2 to backward Euler for one time step. The delay exhibited by backward Euler causes the bump.

grading accuracy significantly. However, simulating a circuit where errors accumulate can give poor results even if tight tolerances are used. This is one reason why approximate timing simulators can give accurate answers on logic circuits, which are designed to be insensitive to errors, while analog circuits require considerably more accurate simulators.

Voltages Waveforms of Capacitor Current Test Circuit

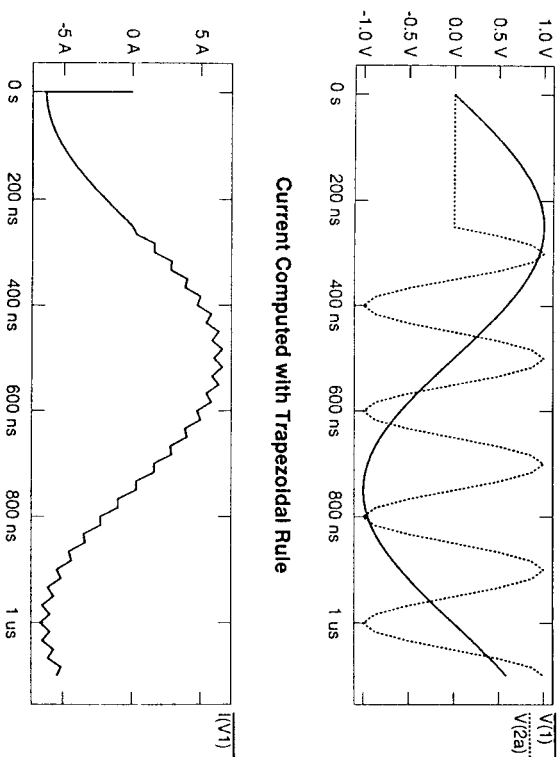


Figure 4.19: Signals computed for Figure 4.17 on page 157 using trapezoidal rule. The current I_{V1} exhibits trapezoidal rule ringing that starts at the break point generated by V_2 . The break point forces a switch from trapezoidal rule to backward Euler for one time step. The delay exhibited by backward Euler when computing currents is enough error to cause trapezoidal rule to start ringing.

4.2.4 Truncation Error

Truncation error is the error made by replacing the time derivatives with a discrete-time approximation. It is useful to consider separately the error made on each time step by using a finite-difference derivative, and the accumulated effect of the error made on each step. The *local truncation error* (LTE) is the truncation error made on a single step assuming all previous steps are accurate. The *global truncation error* (GTE) is the maximum accumulated truncation error. The GTE is related to the LTE made on each step and the tendency of a circuit to accumulate or dissipate errors.

The most important factor contributing to GTE is the circuit's propensity to accumulate error. Some circuits, such as logic and bias circuits, are very insensitive to error. In fact, logic circuits are designed to be insensitive to error. Consider the output of a logic gate sitting at zero that is perturbed slightly. It quickly returns to zero. After a few time constants, the effect of the perturbation is gone. It is the fast time constants that act to quickly restore the signals to their proper values that make logic and bias circuits so insensitive to truncation errors and simulator errors in general.

4.2.4.1 Sensitivity of Circuit to Errors

Circuits vary as to how they accumulate error. Circuits that tend to be very sensitive to simulator errors include those that have very long time constants. Examples of sensitive circuits include charge storage circuits such as switched-capacitor circuits and dynamic memories, chaotic² circuits, such as oversampled analog/digital converters, and autonomous circuits such as oscillators. To determine whether a circuit is sensitive to simulator errors, ask yourself, if an error is made, does it dissipate or persist.

For example, consider Netlist 4.1 as shown in Figure 4.20 on the facing page. This circuit contains no damping elements, so the error made on every time step simply accumulates. It also happens to be very sensitive to any error because it depends on precise cancellation.

²Chaotic circuits are nonlinear circuits whose response is an extremely sensitive function of its state.

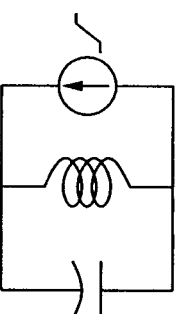


Figure 4.20: *LC resonator.* The computed response of this circuit is very sensitive to simulator errors.

If you run the circuit with very tight tolerances, you get the solution shown in Figure 4.21.

The start of the ramp causes the resonator to start oscillating. The end of the ramp is an equal and opposite stimulus, and comes after exactly 10 cycles on the true solution. Thus it should kill the oscillation. However, if the simulator has not precisely reached the end of the tenth cycle exactly when the end of the ramp occurs, the oscillation is not killed. To show how errors generated by the simulator affect the computed response, the solution computed by SPICE2 is given in Figure 4.22 on page 163 along with the true solution. Notice that the frequency of oscillation computed by SPICE2 is off, and that prevents the oscillation from being killed properly.

Analog circuits, particularly oscillators and integrators are very sensitive to truncation error. These circuits exhibit very long time constants that retain the effect of any perturbations for a very long time. Consider the *RC* circuit shown in Figure 4.23. Use superposition to consider the effect of truncation error separately from any other signals present. Let current from the source represent error injected into the circuit. Figure 4.24 shows the response of the *RC* circuit to a single error that occurs at time t_n . A fast time constant results in the effect of the error disappearing quickly. Figure 4.24 also shows the response of the *RC* circuit to a continuous stream of error starting at t_n .

LC Oscillator Accuracy Test

* The computed response for this circuit is very
 * sensitive to errors made in the numerical
 * integration. The response should consist of 10
 * periods of a pure sine wave with the minimum peaks
 * at 0 and the maximum peaks at $1/(10\pi) = 0.0318$.
 * After the tenth period, the response should become
 * constant at exactly 0 volts.

```
.option acct lvltim=2 tnom=27
.option reitot=0.001 trtol=7
.option abstol=1e-12 vntol=1e-06 chgtol=1e-14
.option method=trapezoidal
```

```
C1 1 0 1
L1 1 0 1
I1 1 0 pulse 1 0 0 62.831853071795862 62.831853071795862
```

```
.tran 10 80
.plot tran v(1)
.end
```

Netlist 4.1: SPICE netlist of *LC* resonator. The computed response of this circuit is very sensitive to simulator errors.

Now consider the circuit shown in Figure 4.25, which is very sensitive to error because it has an infinitely long time constant (or in other words, it has a pole at 0). This results in the effect of any error being retained forever. Figure 4.26 shows the response of the circuit to a single error occurring at t_n . Note that the effect of the error remains forever. Figure 4.26 also shows the response of the circuit to a continuous stream of error starting at t_n . Notice that the effect of the error grows forever.

Response of LC Resonator

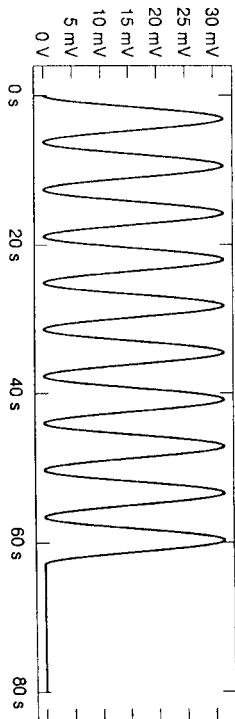


Figure 4.21: Correct response of *LC* resonator.

Response of LC Resonator

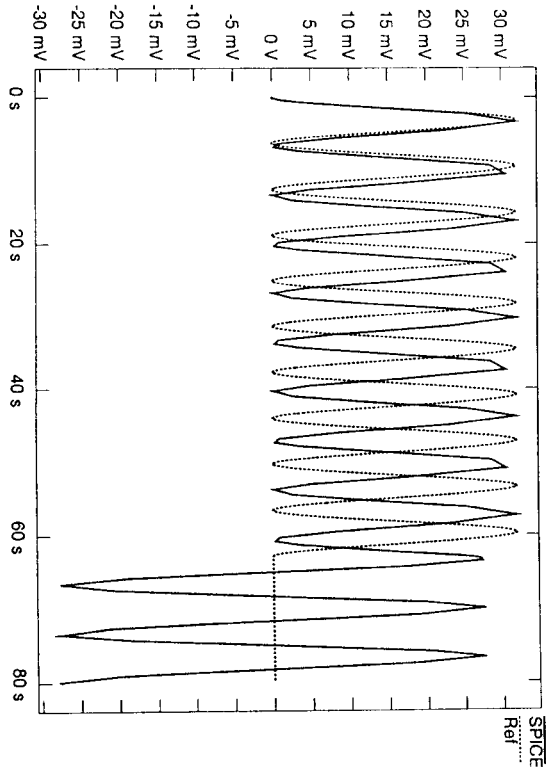


Figure 4.22: Response of *LC* resonator computed by SPICE2 along with the correct response. The phase error in the SPICE results is responsible for the oscillation not being killed.

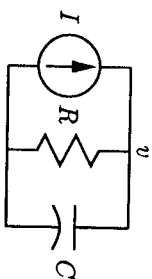


Figure 4.23: RC circuit, whose finite time constant serves to dissipate simulator errors.

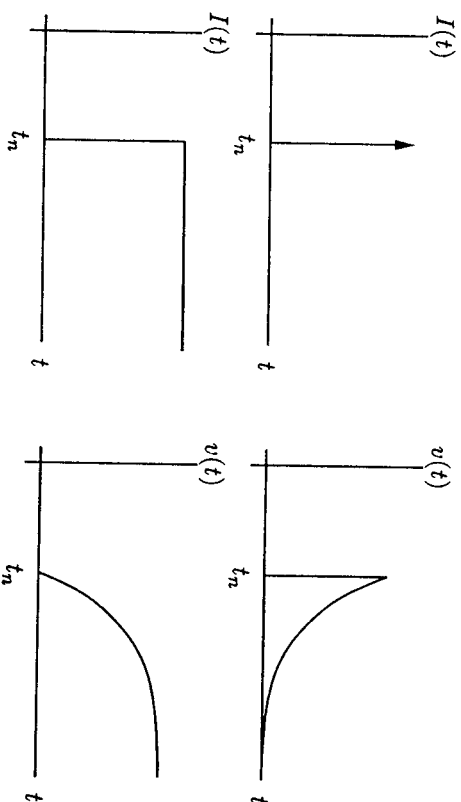


Figure 4.24: Response of RC circuit to a single error at time t_n (top) and to a continuous stream of error starting at t_n (bottom). The error stimulus is on the left and the response is on the right.

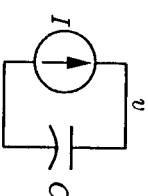


Figure 4.25: LC circuit, whose infinite time constant serves to store and accumulate simulator errors.

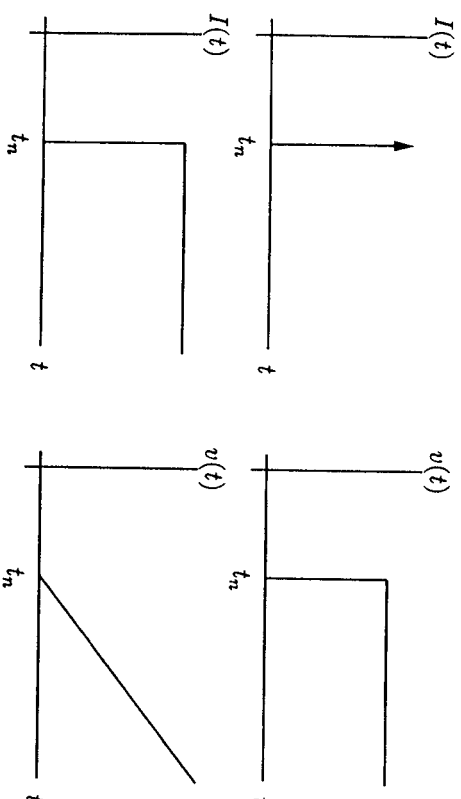


Figure 4.26: Response of LC circuit to a single error at time t_n (top) and to continuous stream of error starting at t_n (bottom). The error stimulus is on the left and the response is on the right.

4.2.4.2 Local Truncation Error

Discretization or truncation error results from analyzing the circuit at a finite number of time steps. This error stems from the approximations made in (4.9–4.11). Generally, more time steps means less error, however they must be chosen carefully. It is very difficult to estimate the accuracy of transient analysis. It depends on the type of circuit being simulated, as well as on the number of time steps and their placement. Truncation error is controlled by `reltol` × `trtol` (SPICE's `trtol` is similar to Spectre's `lteratio`). The simulator chooses the time step to control the truncation error made at each step. The amount of error is related to the step size. For first-order methods such as backward Euler the error is proportional to the square of the step size (for small steps). For second-order methods such as the trapezoidal rule and the second-order backward difference formula, the error is proportional to the cube of the stepsize (again for small steps). How this error accumulates is determined by the circuit.

The integration methods used in circuit simulation are taken from a general class of methods called multistep methods. These methods are all formulated to be exact for some low-order polynomial. For example, backward Euler is exact if the solution is a line, as is seen from (4.9). Trapezoidal rule and the second-order backward difference formula are exact for lines and quadratics. It is rare when solution waveforms follow linear or parabolic trajectories exactly, however over short time spans they can be approximated as such, and when using a quadratic the span can be longer than when using a line. An important practical point is that there is no truncation error when solution waveforms are constant valued. Thus, truncation errors affect time constants, such as settling times and periods of oscillation, but do not affect DC values or settled values unless they depend on time constants (for example, if an oscillator were to drive a frequency-to-voltage converter). The end result is that if you are very interested in finding equilibrium points accurately and the accuracy of time constants is not of paramount importance, you can get accurate results by tightening `reltol` and speed up the simulation by loosening `trtol` (lteratio in Spectre).

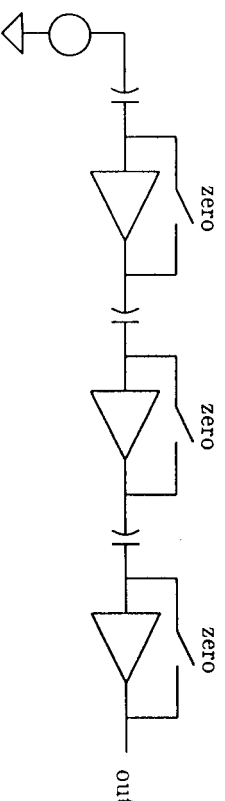


Figure 4.27: AC-coupled comparator, a circuit that is very sensitive to charge errors.

4.2.5 Charge Conservation

There are several mechanisms that affect charge conservation. The charge conserving nature of semiconductor models is the most important factor that affects charge conservation because of the large amount of charge that is created or annihilated on every time step by capacitance-based models. The Meyer capacitance model [meyer71] used in MOS1, MOS2, and MOS3 models is capacitance-based and so is not charge conserving. The Ward-Dutton model [ward78], available in MOS2, the Yang-Chatterjee model [yang82], and the BSIM model, both available on all MOSFETs installed in Spectre, are all charge-based and so conserve charge.

Even when exclusively using charge-conserving models, there can still be some problems with charge conservation. The error due to the convergence criteria was discussed in the section on DC analysis. This results in charge not being completely conserved, which is an important point. Even simulators with charge-conserving models do not conserve charge completely, though they do a much better job of it than simulators that use non-charge-conserving models.

While charge conservation errors are small, they occur on each time step and in sensitive circuits, such as charge-storage circuits, the small charge errors can accumulate until they become a big problem. Consider the AC-coupled comparator shown in Figure 4.27. This is an example of a circuit that is extremely sensitive to error

Stimulus for AC-Coupled Comparator

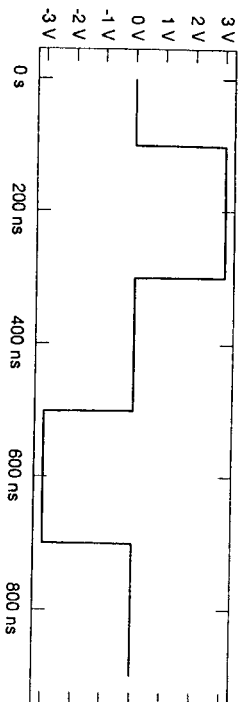


Figure 4.28: Stimulus waveform for the AC-coupled comparator.

(see Section 4.2.4 on page 160 to recall how certain types of circuits cause errors to accumulate). The switches in this circuit are initially set such that the offset in the amplifiers is stored on the interstage capacitors. Ideally, when switched back to the normal mode of operation, the offset stored on the capacitor cancels the offset on the amplifiers. Unfortunately, if the capacitor model used in the FETs acting as switches do not conserve charge, the charge error is stored on the interstage capacitors and amplified. This is shown in Figure 4.29, which is the output of the comparator when driven with the input shown in Figure 4.28. Since the stimulus returns to zero at the end of the simulation interval, the output should at least return to being close to zero. The charge error that results from using non-charge-conserving models prevents this from occurring. When the circuit is resimulated using charge conserving models, the computed solution (as shown in Figure 4.30 on the next page) is considerably more accurate, though it does exhibit some charge error. If `reltol` is tightened and the circuit resimulated, as shown in Figure 4.31 on page 170, all hint of charge error is gone.

Why Capacitance-based Models do not Conserve Charge In SPICE, the Meyer capacitance model is used to model the dynamic nature of the MOS devices. The Meyer model consists of equations that give capacitance as a function of terminal voltages. The capac-

Response of AC-Coupled Comparator as Computed by SPICE

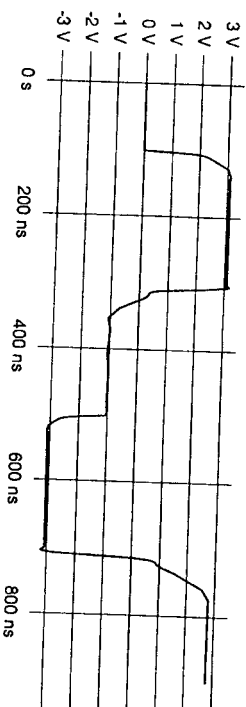


Figure 4.29: Response of the AC-coupled comparator computed by SPICE with non-charge conserving models. Notice that solution does not return to zero as it should.

Response of AC-Coupled Comparator as Computed by Spectre

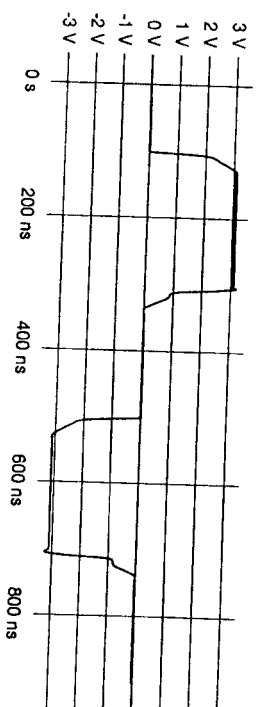


Figure 4.30: Response of the AC-coupled comparator computed by Spectre with charge conserving models. Notice that solution returns almost to zero.

Response of AC-Coupled Comparator when Computed with Tight Tolerances

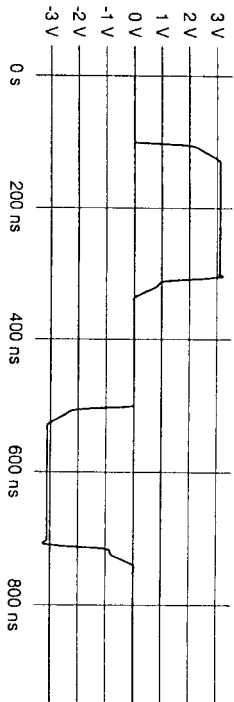


Figure 4.31: Response of the AC-coupled comparator computed by Spectre with `reltol` tightened. Notice that solution returns to zero as it should.

itors are nonlinear. Capacitance is an incremental or small-signal quantity that is defined as $C = dq/dv$. Defining a capacitor model by giving capacitance as a function of voltage results in the model not conserving charge. The reason capacitance-based models do not conserve charge is that capacitance is an incremental quantity that only accurately predicts the change in charge versus voltage for infinitesimally small changes in voltage. The other method of defining a capacitor model, that of giving charge as a function of voltage, does conserve charge.

Consider the following experiment, connect a linear $1\mu\text{F}$ capacitor across a voltage source. Assume that the voltage across the source changes from 0 to 1mV in one time step, and then back to zero on the next as shown in Figure 4.32. Let both time steps be 1ns . Then current is computed on the first step as $i = Cdv/dt = 1\text{A}$, and the charge is $Q = Cv = 1\text{nCoul}$. On the second step, the charge flow is equal and opposite, with a net zero charge being supplied by the source after the second step. Thus, capacitance based models do not result in charge conservation problems when the capacitor is linear and the capacitance is exact.

If the capacitor is nonlinear and described with an equation that

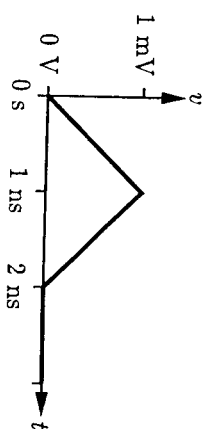


Figure 4.32: Stimulus waveform to the charge conservation example.

gives capacitance as a function of voltage, neither the current nor the charge can be computed exactly. They can be approximated using backward Euler (4.9) as follows,

$$i(t_1) = C(v(t_0))[v(t_1) - v(t_0)]/[t_1 - t_0] \quad (4.30)$$

$$\Delta q(t_1) = C(v(t_0))[v(t_1) - v(t_0)] \quad (4.31)$$

Thus if $C(0) = 1\mu\text{F}$ and $C(0.001) = 1.1\mu\text{F}$, then

$$i(t_1) = 1\text{A} \quad (4.32)$$

$$\Delta q(t_1) = 1\text{nCoul} \quad (4.33)$$

$$i(t_2) = -1.1\text{A} \quad (4.34)$$

$$\Delta q(t_2) = -1.1\text{nCoul} \quad (4.35)$$

Thus, the net charge is -0.1 nCoul . This is illustrated in Figure 4.33 on the next page.

If a charge-based model is used, the problem goes away. Assume the capacitor is modeled with the equation $q(v) = Cv + Dv^2$, where $C = 1\mu\text{F}$ and $D = 50\mu\text{F}/\text{V}$. This equation was chosen because it satisfies $C(0) = 1\mu\text{F}$ and $C(0.001) = 1.1\mu\text{F}$. Now the current still has to be approximated, but the approximation is such that charge is always conserved.

$$i(t_1) = [q(v(t_1)) - q(v(t_0))]/[t_1 - t_0] \quad (4.36)$$

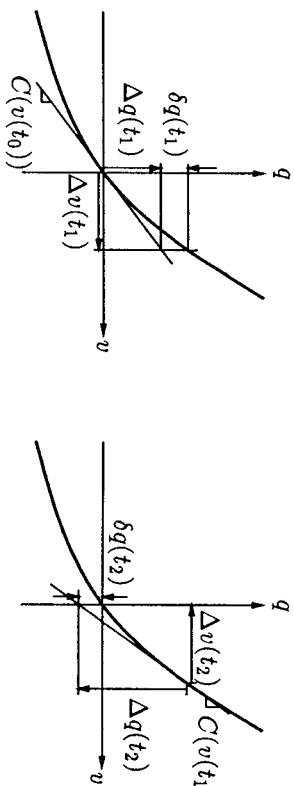


Figure 4.33: Example that shows that charge is not conserved (in other words, $\Delta q(t_1) \neq -\Delta q(t_2)$) even though $\Delta v(t_1) = -\Delta v(t_2)$) when using capacitance-based models because the change in charge on a time step is computed from a linear approximation of the charge function rather than the using the charge function itself. $\delta q(t_n)$ represents the charge error made on step n .

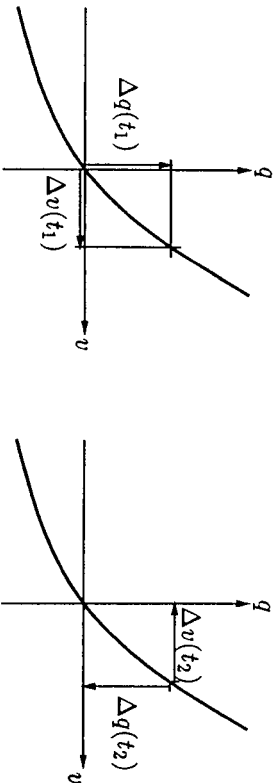


Figure 4.34: Example that shows that charge is conserved when using charge-based models (because $\Delta q(t_1) = -\Delta q(t_2)$ when $\Delta v(t_1) = -\Delta v(t_2)$).

$$i(t_1) = 1.05\text{A} \quad (4.37)$$

$$\Delta q(t_1) = 1.05\text{nCoul} \quad (4.38)$$

$$i(t_2) = -1.05\text{A} \quad (4.39)$$

$$\Delta q(t_2) = -1.05\text{nCoul} \quad (4.40)$$

The net charge is 0. This is illustrated in Figure 4.34 on the facing page.

The crucial point here is that since charge is computed explicitly, if a charge-based capacitor model is evaluated for the charge at a particular voltage, and then the voltage on the capacitor goes through some large excursion but returns to its original value, then the charge must also return to its original value because it is computed with an algebraic expression that has not changed using the same input. With a capacitance-based model, the small errors made on each step of the voltage trajectory accumulate and the final charge does not equal the starting charge.

This example also partially illustrates why the MOSFET capacitor models in SPICE have not been fixed. It is first necessary to derive a charge equation that fits the model. In other words, it is necessary to derive a set of charge relations for each terminal such that the derivative of these charges match the Meyer capacitances. Unfortunately, that is not possible. The Meyer capacitances are incomplete and inconsistent and so there exists no charge functions that when differentiated give the Meyer capacitances.

If you are forced to use the Meyer model, the charge-conservation problem can be reduced by tightening `reltol`, but it cannot be eliminated. Tightening `reltol` reduces charge error because it results in the simulator taking smaller time steps, which implies the change in voltage across the capacitor over a time step is smaller and so the linearized capacitance model is more accurate. However, Meyer's incomplete set of capacitances does result in charge being created or destroyed regardless of how small a time step is used.

The Statz GaAsFET model, unlike the Meyer model, does conserve total device charge because it has an explicit relationship between total charge and the terminal voltages. However, the model does not have explicit relationships for the drain and source charge, so charge

does mysteriously appear or disappear on the drain and source. Total charge is conserved because the explicit total-charge relationship forces the exact amount of charge that disappears from the drain to appear on the source and visa versa.

Requirements for Capacitor Models to Conserve Charge³

For a capacitor model to be charge conserving, it must be such that if the voltage is changed and then returned to its original value, the final charge must equal the initial charge, regardless of the path taken or the starting point. This is true for models described with single-valued charge functions because $q(v_i) = q(v_f)$ if $v_i = v_f$. As shown previously, capacitance-based models do not conserve charge if the capacitor is nonlinear and the path is discretized with a finite number of steps because the capacitance is a linear approximation to the charge function and if the steps are not infinitesimal, a finite error accumulates on each step.

An interesting question remains. If only a capacitance-based model is available, can the charge error be reduced to negligible levels by taking small time steps. For a capacitance-based model to be charge conserving when the time step asymptotically approaches zero the capacitances must include all of the derivatives of a charge function, and must be exact. This is shown by integrating all the charge that flows into a capacitor while traversing a closed path of voltages. Consider the charge that flows from gate-to-source on a MOSFET. This charge is a function of V_{gs} , V_{gd} , and V_{gb} and so is denoted $Q_{gs}(V_{gs}, V_{gd}, V_{gb})$. In the notation of vector calculus, the total charge accumulated around the path is written

$$\oint_C \frac{\partial Q_{gs}}{\partial V} dV = \oint_C \nabla Q_{gs}(V) \cdot dV = Q_{err} \quad (4.41)$$

where $V = [V_{gs} \ V_{gd} \ V_{gb}]^T$ is the voltage vector, $\nabla Q_{gs}(V)$ is the capacitance written as a gradient, C is any closed path, and Q_{err} is the total charge accumulated along the path. Expanding the gradient

³This section assumes the reader has a background in vector calculus. You may skip this section without loss of continuity.

4.2. Transient Analysis Theory

175

gives

$$\oint_C \left(\frac{\partial Q_{gs}}{\partial V_{gs}} dV_{gs} + \frac{\partial Q_{gs}}{\partial V_{gd}} dV_{gd} + \frac{\partial Q_{gs}}{\partial V_{gb}} dV_{gb} \right) = Q_{err}. \quad (4.42)$$

Stoke's theorem states that

$$\oint_C \nabla Q_{gs}(V) \cdot dV = \int_S \nabla \times \nabla Q_{gs} \cdot dS \quad (4.43)$$

where S is the surface enclosed by C . Using Stoke's theorem on (4.41) converts the path integral in terms of the first derivatives of charge to a surface integral in terms of the second derivatives of charge.

$$\int_S \nabla \times \nabla Q_{gs} \cdot dS = Q_{err} \quad (4.44)$$

Expanding the cross and dot products gives

$$\begin{aligned} \int_S \left(\frac{\partial^2 Q_{gs}}{\partial V_{gd} \partial V_{gb}} (V_{gs}, V_{gd}, V_{gb}) - \frac{\partial^2 Q_{gs}}{\partial V_{gb} \partial V_{gd}} (V_{gs}, V_{gd}, V_{gb}) \right) dS_{gs} + \\ \left(\frac{\partial^2 Q_{gs}}{\partial V_{gb} \partial V_{gs}} (V_{gs}, V_{gd}, V_{gb}) - \frac{\partial^2 Q_{gs}}{\partial V_{gs} \partial V_{gb}} (V_{gs}, V_{gd}, V_{gb}) \right) dS_{gd} + \\ \left(\frac{\partial^2 Q_{gs}}{\partial V_{gs} \partial V_{gd}} (V_{gs}, V_{gd}, V_{gb}) - \frac{\partial^2 Q_{gs}}{\partial V_{gd} \partial V_{gs}} (V_{gs}, V_{gd}, V_{gb}) \right) dS_{gb} = Q_{err} \end{aligned} \quad (4.45)$$

If Q_{gs} is continuously differentiable, it can be shown that the second derivative must be symmetric⁴. This condition requires that

$$\frac{\partial^2 Q_{gs}}{\partial V_{gd} \partial V_{gb}} (V_{gs}, V_{gd}, V_{gb}) = \frac{\partial^2 Q_{gs}}{\partial V_{gb} \partial V_{gd}} (V_{gs}, V_{gd}, V_{gb}) \quad (4.46)$$

$$\frac{\partial^2 Q_{gs}}{\partial V_{gs} \partial V_{gd}} (V_{gs}, V_{gd}, V_{gb}) = \frac{\partial^2 Q_{gs}}{\partial V_{gd} \partial V_{gs}} (V_{gs}, V_{gd}, V_{gb}) \quad (4.47)$$

$$\frac{\partial^2 Q_{gs}}{\partial V_{gb} \partial V_{gs}} (V_{gs}, V_{gd}, V_{gb}) = \frac{\partial^2 Q_{gs}}{\partial V_{gs} \partial V_{gb}} (V_{gs}, V_{gd}, V_{gb}). \quad (4.48)$$

⁴It is a standard theorem in vector calculus that if a function is continuously differentiable at a point, then its second derivative or Hessian must be symmetric at that point.

Substituting (4.46–4.48) into (4.45) gives

$$0 = Q_{\text{err}}. \quad (4.49)$$

Thus, the charge accumulated along a closed-path of voltages for a capacitance-based model is zero if the steps taken to traverse the path are infinitesimally small, if the capacitances actually stem from some charge function, if they include all of the derivatives of the charge function, and if they are exact.

A complete set of derivatives for $Q_{gs}(V_{gs}, V_{gd}, V_{gb})$ include

$$\frac{\partial Q_{gs}}{\partial V_{gs}}(V_{gs}, V_{gd}, V_{gb}) = C_{gs}(V_{gs}, V_{gd}, V_{gb}) \quad (4.50)$$

$$\frac{\partial Q_{gs}}{\partial V_{gd}}(V_{gs}, V_{gd}, V_{gb}) \quad (4.51)$$

$$\frac{\partial Q_{gs}}{\partial V_{gb}}(V_{gs}, V_{gd}, V_{gb}). \quad (4.52)$$

Of this set, the Meyer capacitance model only provides C_{gs} , the other derivatives are neglected. This is insufficient to conserve charge because (4.47–4.48) are not satisfied. In fact, since $\frac{\partial Q_{gs}}{\partial V_{gd}}$ is zero in the Meyer model, then $\frac{\partial^2 Q_{gs}}{\partial V_{gs} \partial V_{gd}}$ is zero for all V_{gs} , V_{gd} , and V_{gb} . C_{gs} is a function of V_{gd} , so $\frac{\partial^2 Q_{gs}}{\partial V_{gs} \partial V_{gd}} C_{gs} = \frac{\partial^2 Q_{gs}}{\partial V_{gd} \partial V_{gs}}$ is nonzero. Thus,

$$\frac{\partial^2 Q_{gs}}{\partial V_{gs} \partial V_{gd}}(V_{gs}, V_{gd}, V_{gb}) \neq \frac{\partial^2 Q_{gs}}{\partial V_{gd} \partial V_{gs}}(V_{gs}, V_{gd}, V_{gb}). \quad (4.53)$$

Similarly,

$$\frac{\partial^2 Q_{gs}}{\partial V_{gs} \partial V_{gb}}(V_{gs}, V_{gd}, V_{gb}) \neq \frac{\partial^2 Q_{gs}}{\partial V_{gb} \partial V_{gs}}(V_{gs}, V_{gd}, V_{gb}). \quad (4.54)$$

Therefore, there is no charge function that has a derivative that equals the Meyer capacitance. Also, since important terms in (4.45) are neglected, $Q_{\text{err}} \neq 0$ and the model does not conserve charge, regardless of how little the voltage changes on each time step.

4.3 Transient Analysis Practice

This section starts by discussing some of the important heuristics of how a circuit simulator chooses its time step. Details of both SPICE's and Spectre's time step control algorithms are presented. Initial conditions and convergence are discussed at the end of the section.

4.3.1 SPICE's Time-Step Control

SPICE provides 2 time-step control algorithms. The first uses estimates of the local truncation error to choose the time step. The second uses the number of iterations required for convergence at a step to decide how big to make the next time step. This method does not reliably compute accurate waveforms and should not be used.

4.3.1.1 LTE-Based Time Step Control

When using local truncation error (LTE)-based time-step control, (lvtim=2), SPICE estimates the LTE made on every capacitor and inductor and chooses the time step small enough to assure that the largest LTE remains within tolerances. To do so, SPICE needs a measure of the LTE. Recall that N^{th} -order multistep methods accurately compute solutions if the trajectory follows an order N or less polynomial. Thus, N^{th} -order methods are accurate if the derivatives of the solution trajectory order greater than N are zero. SPICE computes the $N + 1^{\text{th}}$ derivative and uses it as an estimate of the LTE. For a second-order method such as trapezoidal rule, SPICE shrinks the time step if the 3rd derivative of the charge on any capacitor (or flux through any inductor) is large.

The LTE threshold criterion for SPICE is too complicated to give here. There are five parameters that affect the criterion.

`reltol` is the universal accuracy control. It affects both the Newton convergence criteria and the LTE criterion. It sets the threshold for the error relative to the charges and currents on the

capacitor and relative to the fluxes and voltages on the inductors. In general, if you need more accuracy, tighten `reltol`.

`trtol` is only used in the LTE criterion where it multiplies `reltol`.

Its primary function is to allow control of the LTE criterion independent of the Newton convergence criteria. It is not a good idea to tighten the LTE criterion without also tightening the Newton convergence criteria because it can cause the simulator to take very small time steps trying to follow the numerical noise generated by errors allowed by the Newton criteria. Thus, one should never reduce `trtol` much below its default value of 7. There is benefit to loosening the LTE criterion relative to the Newton criteria. For example, if you were less concerned about accurately depicting time constants than equilibrium points, you might want to loosen the LTE criterion relative to the Newton criteria by increasing `trtol`. Similarly, if charge conservation is important, you might want to tighten the Newton criteria without affecting the LTE criterion. In this case, tighten `reltol` and loosen `trtol` by the same amount.

$T_{\max}$ is the optional fourth argument on the transient analysis statement. It places an upper bound on the time step. If $T_{\max}$ is not specified, SPICE sets it to

$$T_{\max} = \frac{T_{\text{stop}} - T_{\text{start}}}{50}. \quad (4.55)$$

In addition, SPICE reduces $T_{\max}$ to T_{step} (the second argument on the transient analysis statement) if there are no energy-storage elements such as capacitor or inductor present in the circuit. Finally, SPICE bounds $T_{\max}$ to be no larger than half the transmission delay of the shortest transmission line. Very short transmission lines can significantly slow a SPICE transient analyses.

`abstol` is the absolute tolerance for the derivative of the state variable in the LTE criterion. Thus, `abstol` is the absolute tolerance for current through capacitors and voltage across inductors. `abstol` limits how closely LTE is controlled when the time step is large and when the current through the capacitor or voltage across the inductor is small.

`chgtol` is the absolute tolerance for the state variable in the LTE criterion. Thus, `chgtol` is the absolute tolerance for charge through capacitors and flux across inductors. `chgtol` limits how closely LTE is controlled when the time step is small and when the charge through the capacitor or flux across the inductor is small. Unlike `abstol`, `chgtol` is multiplied by `reltol` in the LTE criterion. Thus, capacitor charges must be less than `reltol` $\times$ `chgtol` before they are dominated by `chgtol`.

In general, SPICE tends to have rather sloppy time-step control. It is not an issue in many circuits because SPICE takes a minimum of 50 time steps, which in many circuits is sufficient to hide the errors. For example, when simulating only one or two periods of the response, 50 points is generally adequate to accurately represent the response. However, if your circuit runs for 10 or more cycles, you should consider tightening `reltol`, `chgtol`, and $T_{\max}$.

4.3.1.2 Iteration-Count Time Step Control

SPICE provides a method of time-step control called *iteration-count time-step control*, which chooses a time step based on the number of iterations required for the previous time step. However, there is no direct relationship between iterations and truncation error. Thus, using this method essentially disables all control of the truncation error. For example, linear circuits always require only one iteration per time step, but they still need their time steps chosen to control truncation error. This is shown in Figure 4.35 on the next page, which shows the response of the circuit in Figure 4.20 computed by SPICE with `lv1tim=1`. Because the circuit is linear, SPICE takes its minimum number of time steps, which in this case is 60, regardless of the error in the results. This results in a 20% error in the computed oscillation frequency. The results would have been much worse if the simulation interval had been longer. *Iteration-count time-step control* (`lv1tim=1`) *should never be used* because the response computed by SPICE could be greatly in error.

With `lv1tim=0`, SPICE also uses iteration-count time-step control, but tries to maintain a time step equal to the smaller of $T_{\max}$ or

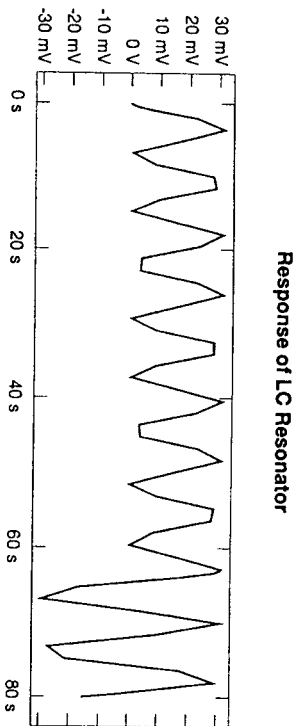


Figure 4.35: Response of LC resonator computed by SPICE2 using $\text{lvltim}=1$. Notice that there are 8 cycles between 0 and 60s rather than the 10 expected.

T_{step} . This is even more dangerous than $\text{lvltim}=1$.

When using iteration-count time-step control, SPICE limits the time step to be no greater than T_{max} or T_{step} . If T_{max} is not specified, it is taken to be

$$T_{\text{max}} = \frac{T_{\text{stop}} - T_{\text{start}}}{50}. \quad (4.56)$$

4.3.2 Spectre's Time-Step Control

To control truncation error, one needs a measure of the local truncation error (LTE), which is the truncation error made on each step. Since multistep methods are derived assuming waveforms are polynomials, the standard measure of LTE is the difference between the computed solution and the polynomial extrapolation from the previous few steps. For a second-order method, such as the trapezoidal rule or Gear2, the LTE is the deviation from a parabola. This is consistent with earlier statements that second-order methods have no error if the solution trajectory follows a second-order polynomial. This is shown in Figure 4.36 on the facing page. In Spectre, the step is accepted if

$$v_n(t) - u_{n,\text{pred}}(t) < \alpha [\text{reltol}v_{n,\text{max}} + \text{vabstol}] \quad (4.57)$$

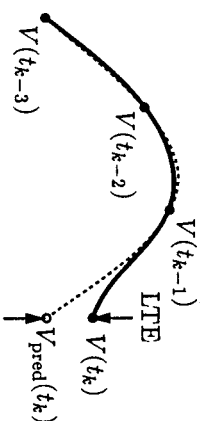


Figure 4.36: The difference between the computed solution and the solution predicted using a polynomial extrapolation through the previous few steps is a measure of the local truncation error (LTE).

on every node n , where α is a constant less than one that depends on the time point spacing and the integration method. Spectre's iteratio serves the same purpose as SPICE's trtol . It is half the size of trtol and so iteratio should not be set less than 1. $v_{n,\text{max}}$ is the reference for relative comparisons. More is said about it in the next section.

Using reltol , both the Newton-Raphson convergence criteria and the truncation error criterion is loosened or tightened simultaneously. The additional parameter iteratio is added to allow you to loosen the truncation error criteria relative to the Newton-Raphson convergence criteria. In general, iteratio is rarely tightened because it might cause the simulator to reduce the time step in order to follow the numerical noise generated by the incomplete convergence of the Newton-Raphson iteration. It is common, however, for iteratio to be loosened. As with SPICE's trtol , iteratio should be loosened when you need a fast simulation and the accuracy of the time constants computed by the simulator is less important than the accuracy of the equilibrium points (flat portions of the waveforms). When charge conservation is important, it is necessary to tighten reltol to avoid charge error in the convergence criteria. But tightening reltol also acts to shrink the time steps, which does not affect charge conservation if charge conserving models are used. Tightening reltol

alone results in an unnecessarily accurate and slow simulation. Loosening `lteratio` by the same amount that `reltol` is tightened returns the time steps to normal size without adversely affecting the charge error.

In SPICE, *charge trajectory* is assumed to be a low-order polynomial over the length of a time step, whereas in Spectre, the *voltage trajectory* is assumed to be a low-order polynomial. Spectre integrates charge, but controls the error in the voltage. For linear capacitors, the two approaches are similar except that in SPICE, `chgtrl` determines the minimum interesting signal level, whereas in Spectre it is determined by `vabstol`. Charge and voltage are related by the capacitor constitutive equation. So, Spectre directly controls the error in voltage and indirectly in charge, whereas SPICE directly controls the error in charge and indirectly in voltage. The benefit of controlling the error in voltage as opposed to charge is that voltage is output by the simulator and is very interesting to the user whereas charge is not interesting and so is simply discarded. From the perspective of charge conservation, it makes no difference whether the time step is set to control error in voltage or charge, because charge creation and annihilation in circuits that use charge conserving models is independent of time step.

In most situations, Spectre's time-step control is more careful than SPICE's. Figures 4.37 on the next page and 4.38 show waveforms computed by SPICE and Spectre. The waveform is the output of the DAC in a bipolar successive-approximation ADC. The circuit contains many small capacitors. SPICE exhibits a large amount of trapezoidal-rule ringing because of its rather sloppy LTE control.

Another example shows that the errors that result from SPICE's LTE criterion manifest in ways other than just trapezoidal-rule ringing. Figure 4.39 on page 185 shows waveforms from a MOS relaxation oscillator computed by SPICE and Spectre. The circuit causes problems for SPICE's charge-based LTE time-step control algorithm because it contains several very small capacitors that determine the duration of the transition between the two states. SPICE, with `chgtrl` set to its default value of 10^{-14} Coul, ignores the large truncation error in these small capacitors. Notice the trapezoidal ringing and the cor-

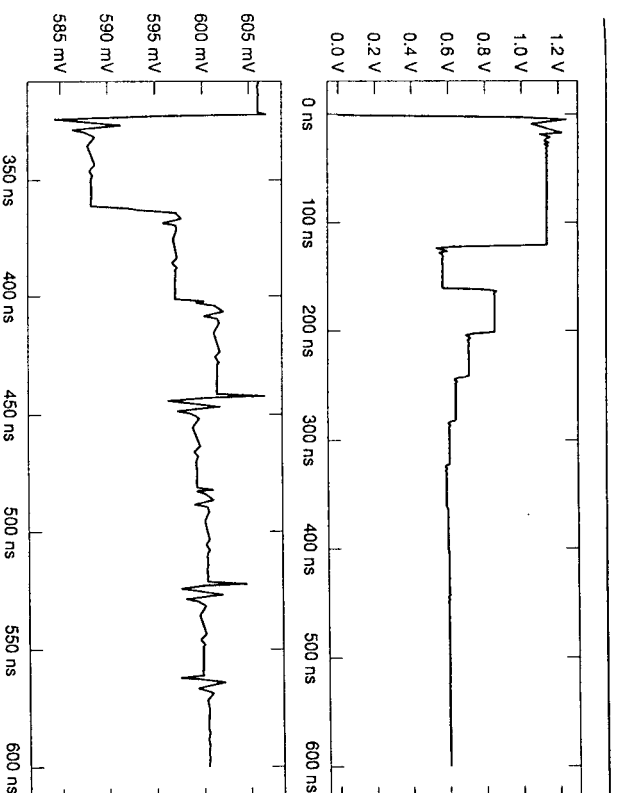


Figure 4.37: Waveform computed by SPICE from the output of the DAC in a bipolar successive-approximation ADC. The bottom waveform is the latter portion of the top waveform magnified.

ners missing from the waveforms near the falling edges in the SPICE waveform. These artifacts are eliminated by Spectre, which chooses the time step to control the LTE in voltage directly.

4.3.2.1 Standard of Reference for Relative Convergence Criteria

The number v_{nmax} acts as the standard of reference for the `reltol` portion of the convergence criteria (2.9) and the local truncation error criterion. Errors that are much smaller than v_{nmax} are considered to be negligible and are ignored by the simulator. There are several different values that can be used for v_{nmax} . For example, two of the

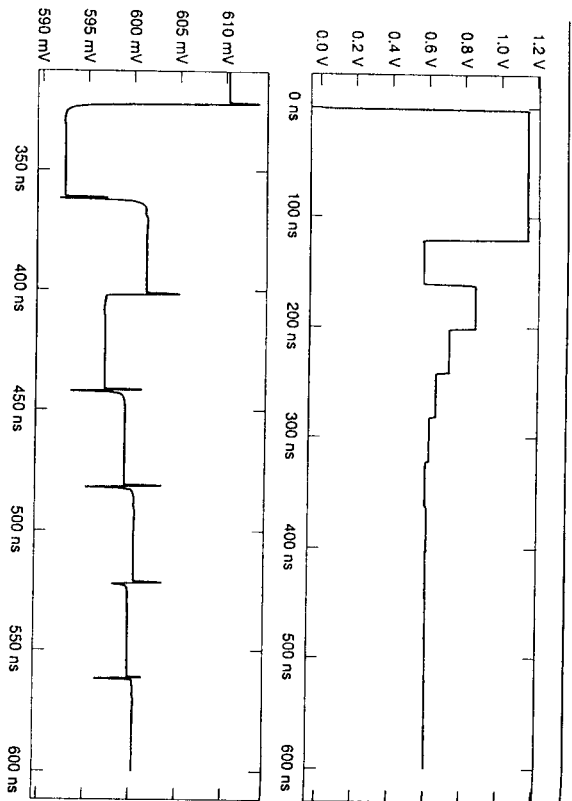


Figure 4.38: Waveform computed by Spectre from the output of the DAC in a bipolar successive-approximation ADC. The bottom waveform is the latter portion of the top waveform magnified.

most common values are:

global:	$v_{n\max} = \max_t(\max_n(v_n(t)))$	(4.58)
local:	$v_{n\max} = \max_t(v_n(t))$	(4.59)
point-local:	$v_{n\max} = v_n(t) $	(4.60)

In the global criterion, $v_{n\max}$ is taken to be the absolute value of the largest voltage present on any node in the circuit over all past time, in the local criterion $v_{n\max}$ is taken to be the absolute value of the largest voltage on the node over all past time, and with the point-local criterion, $v_{n\max}$ is taken to be the absolute value of the current voltage on the node. The global criterion is the loosest of the three, and point-local is the tightest. All three of these definitions for $v_{n\max}$ are useful. The global definition provides the best performance and is preferred if the overall size of the signals is roughly the same on

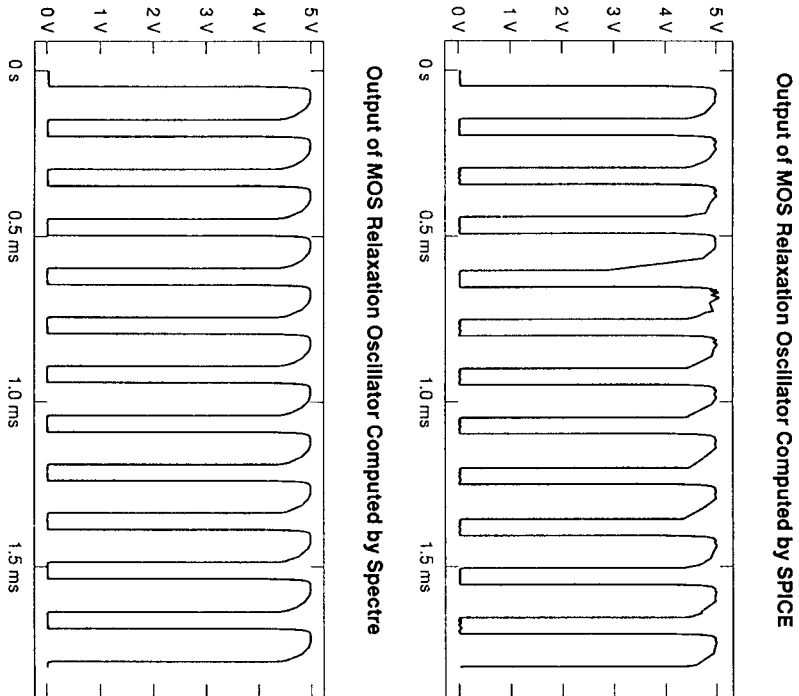


Figure 4.39: Waveform from MOS relaxation oscillator computed by SPICE with its charge-based LTE control (top) and Spectre with its voltage-based LTE control (bottom). Notice the trapezoidal-rule ringing and the corner missing from the fourth pulse in the waveform computed by SPICE.

all nodes. The local definition is more conservative and should be used if the scale of signals varies widely from node to node. Finally, point-local is the most conservative and least used. It should be used when the error criteria should be tightened when the signal level approaches zero.

Consider a circuit that contains low voltage circuitry controlling high voltage circuitry. In particular, node x is inside an opamp with ± 5 V supplies and node y is the power supply node for a 10,000 V interface circuit. If the global criterion were used for this circuit, then v_{nmax} would be at least 10,000 V. Using this v_{nmax} in (2.9) and assuming $reltol = 0.001$ implies that node x could change as much as 10 V from one iteration to the next and still be considered converged. Thus, the convergence criterion has been effectively disabled on node x , which could lead to gross errors. Similarly, the large value of v_{nmax} also disables the LTE criterion (4.57). In this situation, the local criterion for v_{nmax} would place convergence constraints of roughly 10 V on node y and 5 mV on node x , which would generally give accurate results.

For an example of when the global criterion is best, consider a 5 V logic circuit with roughly half its nodes at logical 0 ($= 0$ V) and half at logical 1 ($= 5$ V). If the local criterion were used, then v_{nmax} would be zero on the nodes that are logical 0 and the convergence constraints on these nodes would be $vntol$, which is typically 1 μ V. This constraint is 5000 times tighter than the 5 mV constraint on the logical 1 nodes. Since a logical 0 need not be computed with any more absolute accuracy than a logical 1, the tighter constraints on the logical 0 nodes is unnecessary and wasteful. If the global criterion were used, the v_{nmax} would be 5 V for all nodes and all nodes would use the 5 mV convergence constraint.

Local and global criteria also exist for f_{nmax} in the residue convergence criterion. Circuit simulators are less tolerant of excessive slop in the residue criteria than in the update criteria, therefore one should be more cautious using the global version of f_{nmax} .

Most simulators do not allow you to select whether you use a local or global criterion for either of the Newton convergence criteria. Indeed most simulators do not even tell you what they use for their criteria.

SPICE uses local criteria, but this does not imply that all descendants of SPICE also use local criteria. Spectre uses local criteria in DC analyses and allows you to choose the criteria in transient analysis. Use the transient analysis parameter `relref`. Setting `relref=allglobal` implies that the global criterion should be used for (2.9), (4.57), and (2.10). Setting `relref=sigglobal` implies that the global criterion should be used for (2.9) and (4.57), whereas the local criterion is used for (2.10). Setting `relref=alllocal` implies that the local criterion should be used for (2.9), (4.57), and (2.10). Finally, setting `relref=pointlocal` implies that the point-local criterion should be used for (2.9), (4.57), and (2.10).

4.3.2.2 Error Preset

Generally, users of circuit simulators are more interested in understanding their circuits than in setting all of the various simulator parameters to achieve optimum performance. For this reason, Spectre provides another transient analysis parameter called `errpreset`. Using `errpreset`, you can trade off speed for accuracy. `errpreset` takes one of three values, `conservative`, `moderate`, and `liberal`. The default value of `moderate` represents a reasonable compromise between speed and accuracy, and between recklessness and caution. With `conservative`, the tolerances are tightened slightly, but more importantly, are interpreted in a more cautious fashion. Use this mode for unfamiliar types of circuits or circuits that are known to be sensitive to simulator errors. Circuits generally run more slowly, but there is little chance the simulation results exhibit any significant error. Finally, using `liberal` tells the simulator to be a little reckless in order to run more quickly. Tolerances are loosened somewhat, but again the most important thing is that the accuracy criteria are interpreted more loosely.

The `errpreset` parameter presets the default values of several transient analysis parameters as shown in Table 4.1. It affects `reltol`, `relref`, `method`, `maxstep`, and `iteratio`. If the user specifies a value for one of these parameters, it overrides the values specified in the table. In addition, to presetting these parameters, `errpreset` also determines how Spectre controls the time step to follow signals other

	errpreset		
	liberal	moderate	conservative
reltol	$\times 10$	—	$\div 10$
lteratio	3.5	3.5	10
relref	allglobal	sigglobal	alllocal
method	gear2	traponly	gear2only
maxstep	$\frac{T_{\text{stop}} - T_{\text{start}}}{10}$	$\frac{T_{\text{stop}} - T_{\text{start}}}{50}$	$\frac{T_{\text{stop}} - T_{\text{start}}}{100}$

Table 4.1: The effect of `errpreset` on various simulator parameters during a transient analysis in Spectre.

than capacitor voltages and inductor currents. Unlike errors in capacitor voltages and inductor currents, errors in other signals do not depend on the time step and do not accumulate and so are ignored in other simulators. However, large time steps on these signals results in them being difficult to interpret or aesthetically undesirable. When `errpreset=liberal`, the time step is not controlled to follow the changes in signals other than the capacitor voltages and inductor currents. When `errpreset=moderate`, the time step is reduced to follow large changes in these signals. And when `errpreset=conservative`, the time step is reduced to accurately follow even small changes in these signals. However, in all cases, the time step is never set smaller than the suggested time step. The suggested is using the step transient analysis parameter, or as the first argument on the `SPICE .tran` statement.

When working with unfamiliar types of circuits, it is a good idea to start with `errpreset=conservative`. As your familiarity with the circuit grows, along with your ability to recognize invalid behavior that results from artifacts of the simulation), you can try `moderate` and `liberal`.

Spectre not only provides the three standard integration methods, it also allow you to combine them in various ways. The method can be set to the following values:

4.3. Transient Analysis Practice

189

`euler` Spectre uses backward Euler only.

`trap` Spectre used backward Euler only where advantageous. Otherwise it uses trapezoidal rule.

`traponly` Spectre tries to always use trapezoidal rule. However, it still uses backward Euler at break points (points where stimulus waveforms abruptly change slope).

`gear2` Spectre used backward Euler only where advantageous. Otherwise it uses second-order Gear.

`gear2only` Spectre tries to always use second-order Gear. However, it still uses backward Euler at break points.

`trapgear` Spectre tries to alternate between trapezoidal rule and second-order Gear. This often times provides most of the accuracy of trapezoidal rule, without trapezoidal ringing. Backward Euler is used at break points.

4.3.3 Break Points

Break points are used by all simulators to assure that sharp corners in waveforms are faithfully rendered. They also assure that short duration events, such as that shown in the top of Figure 4.40, are not missed. Break points are generated by independent sources whenever an abrupt change in slope occurs. Simulators never step beyond a break without first stepping on a break point. In other words, simulators always place time points on break points, and so simulators always place time points on the corners of stimulus waveforms. This prevents the simulator from knocking off corners, as in Figures 4.16 on page 154 and 4.39. The sharp corners in these waveforms result from causes other than sharp corners in stimulus, and so there are no break points to square up the corners. This commonly occurs with relaxation oscillators, which is autonomous⁵ and generates waveforms with sharp corners.

Independent sources in SPICE generate break points at:

⁵ An *autonomous* circuit generates an output without having an input. A more formal definition of an autonomous circuit is one that contains no time-varying components, particularly, time varying sources.

1. Every corner of a pulse waveform.
2. The start of a sine wave (at the end of the delay).
3. On every point specified in a piece-wise linear waveform.
4. At the beginning of each time constant in an exponential waveform.

Occasionally, problems result because SPICE generates break points for every point specified in a piece-wise linear waveform. Break points are generated regardless of whether the waveform has a sharp corner at that point or not. To specify a smooth curved waveform with a piece-wise linear source requires that many points be used to specify the waveforms. Each point specified generates a break point, and the simulator is obliged to place a time point to place a time point on every one. The more points specified in the PWL waveform, the longer the simulator runs.

Spectre attempts to eliminate this problem by inspecting the waveform and only generating break points at specified points that produce a sharp corner. Thus, you may use smoothly curved waveforms with points specified very closely without slowing down the simulator. This feature is only applied to piece-wise linear waveforms with many points. If the waveform contains less than 20 points, or if you specifically request it using `allbrkpts`, break points are generated for each point specified in the piece-wise linear waveform.

4.3.3.1 Multiple Sources

If more than one piece-wise linear source is used in a circuit, care should be taken to avoid having a large number of points specified at times that are close to, but not equal to, each other. Since the simulator must place a time point at each break point, and since each point specified in a piece-wise linear waveform generates a break point, two points specified near each other in time requires that the simulator take a very small time step, even if the points are specified for two different sources. Once a simulation takes a small step, it must continue to take small steps for a while. For stability reasons,

simulators typically limit a time step to be no more than twice as large as its predecessor. For example, say that V_1 has points specified at 0, 1 μs , and 2 μs , and V_2 has points specified at 0, 1.001 μs , and 2 μs . Further assume that the simulator maintains accuracy with 1 μs steps. The simulator is forced to take a time step no larger than 1 ns at 1 μs . It then must take about 10 time points to build back to a 1 μs step. Thus, with the points specified as given, it takes approximately 12 time points to go from 0 to 2 μs . Only 3 steps are required if the second point for V_2 is respecified as 1 μs .

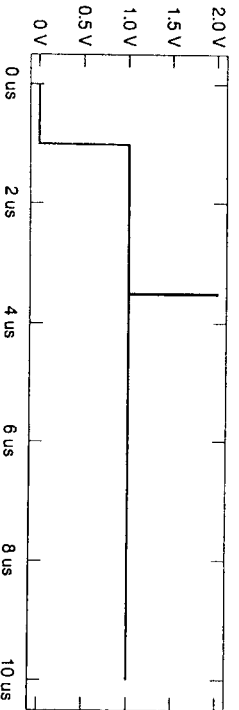
4.3.3.2 Transmission Lines

Another serious problem that is an artifact of break points occurs when the circuit contains several transmission lines. SPICE provides ideal transmission lines. When ideal transmission lines are terminated in their characteristic impedance, they act like ideal delay lines. In other words, a waveform that enters one end of the line exits the other end delayed, but otherwise unchanged. Thus, a sharp corner or a short event that enters a transmission line may exit it T seconds later, where T is the electrical length of the line. As shown in Figure 4.40, if the time step is not controlled using break points, the corner may be beveled, or the short event may be missing entirely. To avoid this, the transmission line generates its own delayed break points.

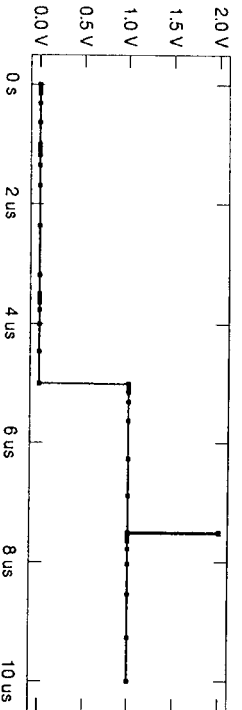
When a transmission line is not terminated with its characteristic impedance, then a sharp transition entering a transmission line generates multiple sharp transitions in the response because of reflections. This is shown in Figure 4.41 with $R = 5Z_0$. If the circuit contains a transmission line of length T , then every break point, even those generated by the transmission line, must be followed by a break point T seconds later. A break point is always placed at the beginning of the transient analysis interval. So, at the very least, there is a break point every T seconds and SPICE cannot take a time step larger than the electrical length of the shortest transmission line in the circuit.

If there are several transmission lines of varying lengths, the situation is even worse. Consider a circuit that contains two transmission lines

Stimulus to a Transmission Line



Transmission Line Response With Break-Points



Transmission Line Response Without Break-Points

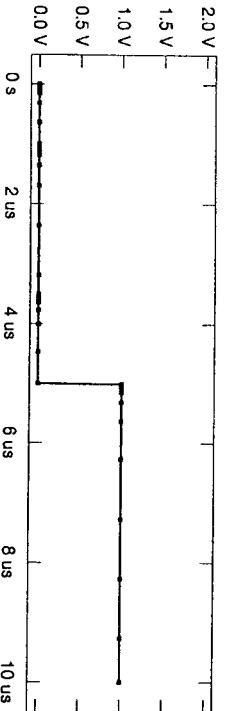


Figure 4.40: Response of a 4 μ s long transmission line terminated in its characteristic impedance ($Z_0 = R$) to a short pulse. If the transmission line does not generate delayed break points, the short pulse is completely missed at the output of the line. Time points are marked to show how a simulator without break points steps right over the thin pulse.

Response of a Mismatched Transmission Line

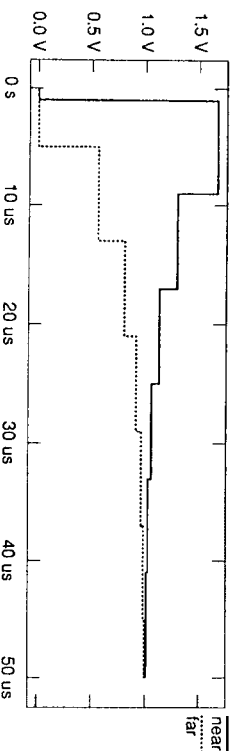


Figure 4.41: Response of a transmission line that is not terminated in its characteristic impedance ($Z_0 \neq R$) to a step. The multiple transitions shows that a single break point can result in a large number if transmission lines are contained in the circuit.

with lengths of 1 ns and 1.01 ns and assume the transient analysis interval is 100 ns. The break point at the beginning of the transient analysis interval causes the first transmission line to generate 100 more at 1 ns, 2 ns, etc. It causes the second transmission line to generate 99 more at 1.01 ns, 2.02 ns, etc. The first break point generated by the first transmission line generates 99 more at 2 ns, 3 ns, etc, but they are all coincident with already existing break points and so are ignored. However, this same break point causes the second transmission line to generate 98 more, at 2.01 ns, 3.02 ns, etc. These are not coincident with previously generated break points and so cannot be ignored. The first break point generated by the second line causes the first line to generate new break points at 3.01 ns, 4.01 ns, etc. This process continues to generate new break points at an explosive rate until at the end of the interval, there are break points every 10 ps (the greatest common multiple of 1 ns and 1.01 ns). If a circuit contains several short transmission lines of various lengths, and if the greatest common multiple for the lengths is much smaller than the transient analysis interval, then the number of break points generated can be extremely large, making the circuit impractical to simulate. SPICE generates all break points in advance, and then combines those that are close. When the circuit generates an extremely

large number of break points, SPICE may run out of memory before the actual simulation begins just trying to remember all of the break points.

SPICE3 and Spectre avoid most these problems by generating break points on the fly, rather than in advance. Break points that do not accompany sharp corners in the waveform are filtered out. The algorithm used in early versions of SPICE3 to filter the break points generated by the transmission lines is unreliable. It would sometimes delete significant break points and not delete insignificant break points, resulting in slow simulations. I am not aware of whether this was fixed in later releases.

4.3.4 Bypass

Signals in digital circuits tend to toggle between two levels, and during a transitions in one signal, it often happens that many other signals in the circuit are unchanging. SPICE implements a feature named 'device bypass' that attempts to exploit that fact that all of the voltages on many of the devices present in a digital circuit do not change significantly during a large number of the time steps. The idea is to record the voltages and currents of each nonlinear component, and if the voltages on that component do not change significantly from the previous time point, simply reuse the previously computed current rather than going through the costly process of recomputing it. While this idea seems very seductive, it usually turns out to be more trouble than its worth.

The first problem with bypass is that it requires that all the voltages, currents, conductances, and capacitances associated with a component must be saved. A large amount of memory is required to hold all of these numbers, often doubling the memory required for each component. This reduces to half the size of circuits that can be simulated when memory is a constraint. In addition, the test to determine if a component should bypass is itself quite expensive, and unlike the current and charge calculations, cannot be bypassed. Thus, the test is an overhead that is always incurred regardless of whether the component is bypassed or not. Finally, problems occur when a device goes out of bypass. In this situation, the current jumps discontinu-

ously, which generates noise in the circuit and sometimes results in convergence problems.

In the best cases, bypass has been found to improve performance on certain circuits by 15–30%. In the worst case it degrades performance, injects noise into the circuit (reducing the resolution of any Fourier analyses being performed), causes convergence problems, and wastes a considerable amount of memory. This dubious optimization was not implemented in Spectre. And unfortunately, the bypass algorithm cannot be disabled on most simulators.

4.3.5 Initial Conditions

When simulating a circuit with a transient analysis, a set of differential equations is formed and solved. However, differential equations have an infinite number of solutions and it necessary to specify a complete set of boundary conditions in order to identify the desired solution. Simulators generally allow you to specify the initial conditions or starting values for capacitor voltages and inductor currents. If the initial condition is not specified, the DC solution is used as the initial condition. Initial condition calculations, while not conceptually difficult, are surprisingly problematic to implement in the context of a nodal-analysis-based circuit simulator, and no SPICE-like simulator available today handles them in a satisfying manner.

While this situation could be remedied, it would require extensive rewrite of the simulator. Because most people are content to let circuits start from the DC solution, and those that use initial conditions are not overly concerned about the accuracy of the initial conditions, the problems are not fixed. These problems, along with several choices on how to specify initial conditions, lead to confusion on how to use initial conditions.

The difficulty in implementing initial conditions in a circuit simulator result because it must accept an incomplete set of initial conditions. Given an incomplete set, where do those state variables for which initial conditions were not specified start? The best solution would be to first eliminate conflicting initial conditions, then solve the circuit where all capacitors that have initial conditions are replaced with

voltage sources and all inductors that have initial conditions are replaced with current sources. The resulting node voltages and branch currents could be used as a complete and consistent set of initial conditions for the original circuit. However, this wholesale modification of the circuit's topology is difficult for most circuit simulators and so other, more approximate and easier to implement, approaches are used.

This section describes initial conditions as implemented so that you can use them and get what you want despite the deficiencies of the actual implementation.

4.3.5.1 Initial Conditions in SPICE

Initial Conditions with the UIC Option SPICE provides two different methods of supporting initial conditions. The original method is selected using the `uic` flag on the transient analysis. In this approach, the solution at $t = 0$ is never computed, it is simply inferred from the initial conditions. The value of the first time step is computed with backward Euler (4.9), and each capacitor and inductor uses the specified initial condition as its $t = 0$ value. SPICE is a component-based simulator, and so forces initial conditions on components. It also provides a mechanism whereby node-based initial conditions are transferred to the components. If the initial condition on a capacitor was not specified explicitly, then the capacitor computes its initial condition by subtracting the initial conditions from two nodes it is connected to. If initial conditions are not specified for the nodes, they are assumed to be zero. This approach is inconvenient because it forces all inductors and capacitors to have initial conditions. If initial conditions were not given for particular inductors or capacitors, and if they cannot be explicitly derived from initial conditions placed on nodes using the `.ic` statement, they are assumed to be zero.

If a loop of capacitors or a cutset of inductors exists, then the initial condition is over-constrained. For example, consider a circuit with two capacitors, C_1 and C_2 , in parallel. Since the capacitors are in parallel, they must share an initial condition. If different initial conditions are specified for both capacitors, then a conflict exists and it

is not clear how to choose the initial condition for the pair. Even if an initial condition is not specified on one of the capacitors, a conflict can still exist because zero is used for all unspecified initial conditions. With `uic` set, SPICE effectively forces the initial condition on each capacitor with the Norton equivalent of a voltage source in series with a small-valued resistor. SPICE forces the initial condition on each inductor with the Thevenin equivalent of a current source in parallel with a small-valued conductor. The value of the forcing source is set to the specified initial condition. The value of the resistance for the capacitor is set equal to $\frac{h}{C}$, where $h = t_1 - t_0$ is the first time step and C is the capacitance of the capacitor. The value of the conductance for the inductor is set equal to $\frac{h}{L}$, where L is the inductance of the inductor. For the situation where there are two or more capacitors connected in parallel, the initial condition used by the simulator is the weighted average of the initial conditions on each capacitor with the capacitance being used for the weights. Averaging of initial conditions occurs any time there is a loop of capacitors. For inductors, the dual situation occurs.

The main complaint that users have with the `uic` method for initial conditions is that any that are not specified are set to zero. Another complaint is that if the user wants to specify the initial condition on a particular capacitor, and it is a member of a loop of capacitors, then the actual initial condition used is modified by the initial conditions on the other capacitors. While it is conceivable that a clever user could run an experiment and determine how to specify the initial condition in order to get the desired value, this becomes very difficult if any of the capacitors in the loop are nonlinear. A solution to this last problem exists if the particular capacitor is grounded. If the initial condition is specified on the node rather than the capacitor itself using the `.ic` statement, and if initial conditions are not specified for any capacitor connected to the node, then SPICE computes the initial condition of every capacitor connected to that node such that the node starts at the desired value.

Initial Conditions without the UIC Option If initial conditions are specified using the `.ic` statement and the `uic` flag was not given on the transient analysis statement, the actual initial state of the

circuit is computed by performing a DC analysis while forcing the initial conditions on nodes using a the Norton equivalent of a voltage source in series with a 1 Ohm resistor (the resistance is given by the **rforce** option in Spectre). This mode allows the user to specify an incomplete set of initial conditions, and have the simulator calculate the others. Notice that no component-based initial conditions are supported in this mode. In particular, one cannot specify inductor initial conditions (these restrictions were eliminated in Spectre).

There are two problems with this approach. First, the 1 Ohm resistor may not be small enough to force the node to the desired voltage. Consider a simple grounded parallel *LC* circuit. Regardless of how small the forcing resistor is, the initial condition on the capacitor is always zero because of the parallel inductor. Interestingly, the voltage source/resistor combination simply forces a current through the inductor and result in a nonzero, and often huge, initial condition on the inductor.

It is also possible to specify an inconsistent set of initial conditions, by which we mean one that cannot be sustained by the capacitors and inductors. A situation that creates an inconsistent initial condition results when trying to set the voltage on a node with no path of capacitors and voltage sources to ground. If the initial condition is inconsistent, the resulting solution jumps discontinuously. That is, it changes instantly at the beginning of the simulation interval. This should be avoided, if possible, because the simulator can have convergence problems trying to make the jump.

If All Else Fails If the two methods of setting initial conditions in SPICE are not satisfactory, there is a more tedious approach that generally works. Duplicate the circuit, force all voltage-based initial conditions with voltage sources and all current-based initial conditions with current sources and perform a DC analysis to generate a complete set of .ic statements for capacitors and ic parameters for inductors. Then use these initial conditions with the original circuit. In this situation, use the **uic** mode of the transient analysis.

4.3.5.2 Spectre's Initial Conditions

Spectre's initial conditions are similar to the approach used by SPICE without the **uic** option. There are three important differences.

1. Initial conditions are supported on branches (capacitor and inductors) as well as nodes.
2. Initial condition specifications can be selectively enabled for nodes and branches. Do so by setting the transient analysis parameter **ic** to either **dc**, **node**, **dev**, or **all**.
3. The size of the resistor's used to force the value of the initial conditions can be scaled using the **rforce** option.

In addition, Spectre issues a warning if the initial conditions that are computed are different from those that are specified.

4.3.5.3 Initial Conditions Versus Nodesets

Nodesets and initial conditions are implemented in a similar manner but their effect is quite different. Initial conditions actually define the initial solution, whereas nodesets only influence it. Also, while nodesets affect both the DC and transient analyses, initial conditions only affect the transient analysis. The transient waveforms start from initial conditions. Nodesets are usually used only as a convergence aid and do not affect the final results (unless the circuit has more than one solution, such as a latch, in which case nodesets act to bias the simulator towards finding the solution closest to the nodeset values).

4.3.6 Remedies for Accuracy Problems

In general, if you would like your simulator to compute a more accurate solution, tighten **reltol**. Also, make sure **abstol** and **vntol** are reasonable. With Spectre, **errpreset=conservative** sets the most important error control parameters to emphasize accuracy over speed. If your circuit is sensitive to simulator errors, it is a good idea

to tighten `reltol` before you even start. In some situations tightening `reltol` may not help, or it may slow the simulator down more than necessary. So you might also consider the following suggestions.

The best way to reduce the error in the solution in most situations is to tighten `reltol`. Artificially shrinking the time step is often not a good idea because the Newton-Raphson convergence criteria are not affected and because not all time steps are shrunk, only the large ones. The one exception to this is circuits that do not contain dynamic components (such as capacitors and inductors). In this case, there is no local-truncation error and so tightening `reltol` does not affect the time step. In this case, the only way to force more time steps is to reduce the maximum time step (`maxstep` in Spectre). This does not make the results computed any more accurate, because if there are no dynamic components, accuracy is independent of the time step, but it does result in more pleasing and less misleading waveforms.

Particular situations where it is appropriate to use the maximum time step to control error is on sinusoidal circuits (to force at least 10–20 time points per period), oscillators (to accurately capture the onset of oscillation), and when performing Fourier analysis (prevent aliasing and reduces interpolation errors).

To resolve accuracy problems, try the following:

1. First verify that the circuit biased up properly. If not, there may be a problem with the topology, the models, or the power supplies.
2. Assure that models you are using are appropriate and that the model parameters are consistent and correct. Also check the operating point of each device.
3. Tighten `reltol`. Also make sure `abstol`, `vntol`, and `chgtol` are reasonable.
4. If you have a charge conservation problem, start by using a charge-conserving model. Then tighten `reltol` if more accuracy is needed.

5. Assure `gmin` is not affecting the solution. If possible, set `gmin` to zero.
6. If the solution exhibits point-to-point ringing, switch the integration method to Gear's second-order backward-difference formula or tighten `reltol` or `chgtol`.
7. If a low-loss resonator seems to exhibit too much dissipation or loss, switch the integration method to the trapezoidal rule or tighten `reltol`.
8. If the initial condition used by the simulator is not the same as the one you specified and you are using the `uic` option on the SPICE transient analysis, try using the `ic` statement instead. If you are using the `ic` statement and you can adjust the forcing resistance, decrease it until the initial conditions are correct. If you specify initial conditions of every capacitor and inductor, specify `uic` on transient analysis.

9. If the simulator is not accurately following the turn-on transient of an oscillator, set the maximum time step (`maxstep` in Spectre) to at most one tenth the size of expected period of oscillation.

4.3.7 Convergence in Transient Analysis

Once circuit simulators have applied an integration method to discretize time, it must then solve a large algebraic system of nonlinear equations at each time step. This is the same problem faced by DC analysis, and many of the issues are the same. You should review Chapter 2 before proceeding.

The transient analysis has similar convergence properties as a continuation method. In fact, it is a continuation method with time being the continuation parameter. If the transient analysis has convergence difficulties at a particular time point, as long as the solution waveforms are continuous, it should always be possible to achieve convergence by taking a smaller time step because eventually as the time step is reduced, the solution at the previous time point enters the region of convergence for the current time point. This is why, in

general, circuit simulators have fewer convergence problems in transient analysis than in DC analysis.

4.3.7.1 Discontinuities

There are two situations for which the simulator's natural tendency to reduce the time step does not improve convergence. First, if the models for the nonlinear capacitors are not continuous, Newton's method can get hung up on the discontinuities and never converge. Shrinking the time step actually makes things worse because it results in the discontinuous capacitors dominating over the presumably continuous linear and nonlinear resistors. Unfortunately, discontinuous models are a fault found in circuit simulators too often. The only real solution to this problem is for the simulator to be fixed so that it supplies continuous models.

The second situation for which reducing the time step does not improve convergence is if the waveforms exhibit discontinuous jumps. This most often occurs in circuits with overly simplified models that exhibit positive feedback, such as relaxation oscillators or Schmitt triggers. Jumps can occur when the circuit contains nodes that do not have a capacitive path to ground. In this case, shrinking the time step does not bring the previous time point into the region of convergence for the current time point because the transition is infinitely fast. In practice, waveforms generated by real circuits cannot jump discontinuously, and so the circuit must be incompletely modeled.

Generally, a small capacitor from the troublesome node to ground solves the problem. Spectre provides the transient parameter `cmin`, which when nonzero connects a small capacitor from every node to ground. This slows down the transitions enough so that Spectre can follow them without convergence problems.

Some situations that produce discontinuous waveforms include:

1. Using bipolar transistor models that include diffusion capacitors (nonzero `tf` and `tr`) without junction capacitors (zero or unspecified `cje` and `cjc`). Diffusion capacitors essentially disappear as the device turns off. Consider the BJT buffer

shown in Figure 4.42. As the input waveform finishes its rising transition, the voltage on the base of Q_2 begins to fall. It begins falling slowly because of the large diffusion capacitance. As the transistor turns off, the diffusion capacitance, which is proportional to the collector current, begins to disappear. In addition, the junction capacitances are not modeled, so there is not capacitance at the collector of Q_1 . The base voltage begins falling faster and faster, until it is moving so fast that the simulator has to take microscopic time steps and may result in convergence difficulties.

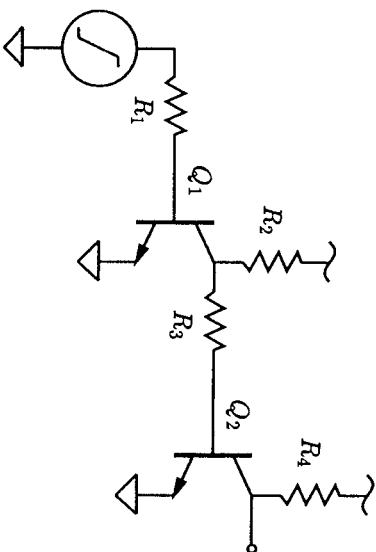
2. Similar problems occur when MOSFETs are specified without overlap or junction capacitors. When the channel disappears, so does the gate capacitance.

Particular care should be taken to avoid disappearing capacitors when they are attached to inductors. An inductor tries to maintain the current into a shrinking capacitor by increasing its voltage. If the increasing voltage causes the capacitor to shrink further, then there is positive feedback that causes the voltage to shoot to infinity. If this kind of nonlinear capacitor is combined with inductors in an LC ladder network, then this positive feedback is capable of generating a shock wave. This problem occasionally occurs in traveling-wave or distributed amplifiers.

4.3.7.2 Large Floating Capacitors

Another situation that causes problems in rare cases is when the time step is small is when there are large floating capacitors attached to nodes with no, or very small, capacitors to ground. With small time steps, this situation becomes similar to the small-floating-resistor problem discussed earlier in Section 2.2.3.

Consider the circuit shown in Figure 4.43, which contains a sinusoidal voltage source, a large floating capacitor, and a large resistor. Problems result from the use of a large floating capacitor to drive a large impedance. The floating capacitor is so large that with a small time step, there is no voltage representable by the computer that results in KCL being satisfied because of its finite precision. Consider the case



Step Response at Base of Q2 in BJT Buffer

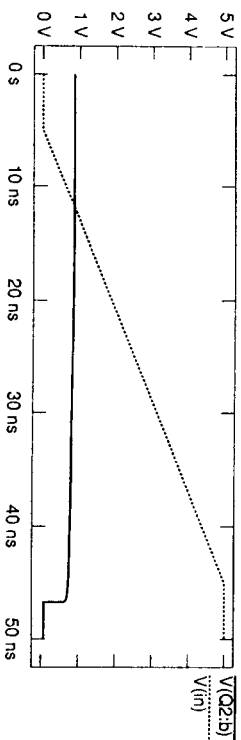


Figure 4.42: A buffer constructed from bipolar transistors and its response to a step input. As transistor Q_2 turns off, its diffusion capacitance disappears, causing the base voltage to drop at an ever increasing rate, which can cause convergence difficulties.

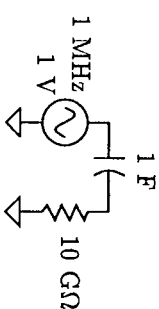


Figure 4.43: Large floating capacitor can cause convergence problems in transient analysis.

where the simulator takes an initial 20 ns time step. The precision of the computer is roughly 15 digits, so the resolution in voltage at the nodes of the capacitor is roughly 10^{-15} V. Thus, the resolution in current through the capacitor is $I = C \frac{dV}{dt} = 1 \times \frac{10^{-15}}{20 \text{ ns}} = 50 \text{ nA}$, which is over 50,000 times greater than abstol. Thus, the capacitor is so large, that given the finite resolution of the voltage, the simulator cannot satisfy KCL to the specified tolerances.

Another problem caused by the large floating capacitor is that when added to the small conductance of the resistor, it can swamp it out, which causes the Jacobian to be singular and the Newton iteration to fail.

These problems are easily avoided by using one of the following suggestions:

1. Use a smaller capacitor.
2. Rearrange the circuit so that the capacitor is grounded (swap the input voltage source and the capacitor).
3. Increase abstol.

Increasing abstol should be the last resort, because it loosens the tolerances over the entire circuit and does not avoid the ill-conditioning in the Jacobian.

4.3.8 Remedies for Convergence Problems

There are two strategies used to circumvent convergence problems in the transient analysis: reduce the effect of discontinuities in the nonlinear capacitors and eliminate discontinuous jumps in the solution.

These suggestions are given to help avoid convergence problems that occur in the middle of a transient analysis. If the convergence problem occurs when trying to compute the initial starting point or initial condition, refer to the suggestions given in Section 2.3.1 on page 35.

1. Heed any warnings the simulator gives. Whenever a problem occurs with a simulation, go back and scan the messages generated by the simulator and look for clues as to why the problem occurs. With Spectre, enable the diagnosis mode by setting `diagnose = yes` on the options statement.
2. When specifying the nonlinear device model parameters, be sure to give a complete capacitance model. Do not use simplified device models that do not have capacitances.
3. Be sure to give the source and drain areas for all MOSFETs. This results in the junction capacitors being modeled. Also, give all overlap capacitances.
4. In Spectre, set `cmin` to a physically reasonable nonzero value. When `cmin` is nonzero, Spectre connects a small capacitor from every node to ground.
5. Follow the suggestions given above to avoid large floating capacitors.
6. Increase SPICE's `itl4` option, or Spectre's `maxiters` transient option, to 50 or more to increase the possible number of iterations on each time step.
7. Try different integration methods, particularly second-order Gear.
8. Loosen the absolute tolerances, particularly `abstol` (`iabstol` with Spectre). Absolute tolerances should not be more than 9

orders of magnitude smaller than the largest signal present in the circuit of the same kind.

9. If, by a process of elimination, you identify a nonlinear capacitance that seems to have a discontinuity, try to simplify or even just modify the nonlinear capacitor model. This can sometimes eliminate the discontinuity.

10. If all else fails, loosen `trtol` (`lteratio` in Spectre) or `reltol` and widen transitions in the stimulus waveforms. This sometimes causes the simulator to just jump past the convergence difficulties (good luck).

4.4 Applications of Transient Analysis

In this, the last major section of this chapter the focus shifts from transient analysis algorithms to circuits. Particular types of circuits are used to illustrate nonobvious issues involved when applying transient analysis to real circuits.

4.4.1 Computing the DC Operating Point

If it is not possible to use DC analysis to compute a DC operating point because of convergence problems, then on most simulators transient analysis can often be used instead. Even on those simulators that employ a pseudo-transient analysis in their DC analysis it is still worthwhile to try using transient analysis. In the pseudo-transient analysis the simulators often do not use the capacitors and inductors specified for the circuit. Instead they generally install 1 F capacitors from every node to ground, or across every nonlinear component. The idea is that the solution process is faster if all capacitors are linear and the same size. However, the 1 F capacitors may cause a circuit to oscillate that would not normally oscillate. In this case, transient analysis is able to find the equilibrium point, where the pseudo-transient is not. Even if the circuit oscillates during transient analysis, because the real circuit is being analyzed and the waveforms are available, it is often possible to study the circuit, determine what is causing the oscillation, and disable it.

To compute a DC operating point with a transient analysis, start by modifying all the independent sources to start at zero and ramp to their DC values. Then, run the transient analysis well beyond the time when all the sources have reached their final value. Using a huge stop time is not be expensive as long as the circuit does not oscillate because the simulator takes large time steps once the solution reaches the equilibrium point. With SPICE, set `uic` on the transient analysis statement to avoid having the transient analysis perform a DC analysis in order to compute its initial conditions. With Spectre, use the transient analysis option `skipdc` for the same purpose (see below for more information on this option).

Use the solution at the final time point to generate a set of node-sets. To make the transient analysis more efficient, set the integration method to backward Euler if possible and eliminate the local-truncation error criterion by setting the option `lv1tim=1` (this is the *only* time you should use this option). Spectre does not allow you to disable the local-truncation error criterion, but you can de-emphasize it by increasing `lteratio`, say to 50 or even 500. To tell SPICE2 to use backward Euler, set the undocumented option `mu` to zero. With Spectre, set the transient analysis options `method=euler`.

Occasionally, this approach fails or is very slow because the circuit contains an oscillator. Try to disable the oscillator before using this approach.

Spectre provides several capabilities that make using transient analysis to compute a DC operating point considerably easier. First, the transient analysis provides the `skipdc` option. If `skipdc=yes`, then the transient analysis avoids using the DC analysis to compute unspecified initial conditions. Instead transient analysis simply starts from the specified initial conditions. Any unspecified initial conditions are taken to be zero. With this setting, the independent sources act the same as they would in a normal transient analysis. However, `skipdc` can also change the behavior of the independent sources. For example, with `skipdc=waveless`, the initial DC analysis is skipped and the independent source values are fixed to their initial values (not their DC values). With `skipdc=rampup`, the independent source values start at 0 and ramp up to their initial values

in the first 10% of the analysis interval. After that their values remain constant. The ramp-up option is best if no initial condition is given, and `waveless` is best when a non-zero initial condition is specified. If `skipdc=autodec` then Spectre automatically selects either `waveless` or `rampup` depending on whether a non-zero initial condition is specified.

When Spectre finishes the transient analysis, it can be directed to automatically write the state of the circuit at the final time point to a file so that it can be recalled to act as a nodeset for a later DC analysis. To do so, use the `writetfinal` transient analysis option. In addition, the final point of the transient analysis can be directly used as an operating point for a subsequent analysis. To use it as an operating point for a small signal analysis, the `smallsignal` analysis must follow the transient analysis and the `prevoppoint` option on the small signal analysis must be set to `yes` (see Section 3.4.4 on page 113). It is also possible to print operating point information about the final transient analysis point by simply following the transient analysis by an `info` statement.

4.4.2 Oscillators

There are two issues that must be considered when simulating oscillators. First, it is necessary to manually start the oscillator. Second, oscillators are more sensitive to simulator error than most nonautonomous circuits, and so it is necessary to be more careful.

Starting the Oscillation The DC analysis finds an equilibrium point for the circuit. By definition, the circuit does not drift away from an equilibrium point unless it is perturbed. If the equilibrium point is stable and the perturbation is small, the circuit drifts back to the equilibrium point afterwards. Oscillators have unstable equilibrium points. Unlike the latch of Figure 2.1 on page 17, when an oscillator is perturbed from its equilibrium point it breaks into an oscillation rather than drifting to a stable equilibrium point. However, a perturbation must be supplied to cause the oscillator to drift away from its equilibrium point and oscillate. In a physical circuit, thermal noise or the turn-on transients are sufficient to start the

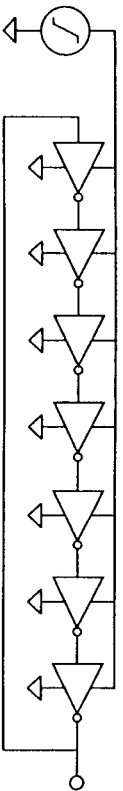


Figure 4.44: A 7-stage ring oscillator used to explore characteristics of various approaches to simulating oscillators.

oscillator. In a circuit simulator, neither stimulus exists and so an explicit perturbation is needed and must be supplied by the user.

One starts an oscillator by either inserting a time-varying independent source, such as one that generates a short pulse, or by specifying an initial condition on a capacitor or inductor. In addition there are usually several places where sources can be installed in the circuit, and several components for which initial conditions can be specified. In general, it is best to choose the method and location that provides the best coupling into the mode of oscillation, and the least coupling to other undesirable response modes. In particular, try to avoid exciting the very slow time constants associated with any bias circuitry.

When using sources to start an oscillator, there is a tendency to use a convenient source such as a power supply. Often these sources are poorly coupled to the mode of oscillation. For example, consider ramping the supply voltage in an attempt to start a 7-state ring oscillator, such as the one shown in Figure 4.44. If all 7 stages are identical, then the pulse would couple equally and identically to each stage. This balanced perturbation is orthogonal to the mode of oscillation in which the outputs of the stages alternate between high and low. A balanced stimulus on a balanced circuit is unlikely to generate an unbalanced response. Figure 4.45 on the facing page shows the response of the 7-stage ring oscillator when its power supply is quickly ramped to its final value.

A better approach for this circuit would be to use an initial condition

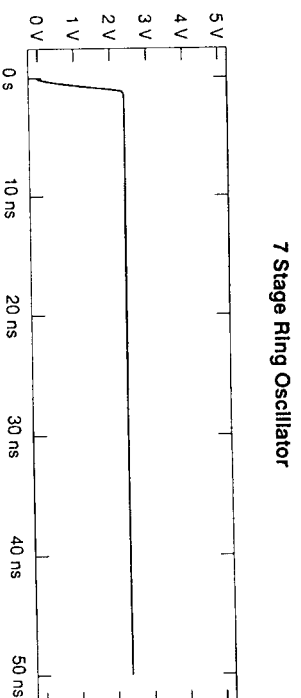


Figure 4.45: Response of the 7-stage ring oscillator when its power supply is quickly ramped to its final value. Oscillator does not start because the perturbation affects all stages identically and so does not couple into the mode of oscillation.

to set on of the nodes in the ring to either ground or to the supply voltage. The actual value used for the initial condition is not very important as long as it is in the normal operating range and it is sufficiently far from the equilibrium point. In this case, an initial condition couples strongly to the mode of the oscillation and does not excite any significant undesirable modes. In addition, using initial conditions has the advantage of not adding any new components to the circuit.

A circuit where initial conditions are not as effective is shown in Figure 4.46 on the next page and Netlist 4.2. Using uic-based initial conditions requires that you specify initial conditions for every capacitor and inductor in the circuit. For this circuit alone that would be easy. However, it this circuit were just a piece of a much larger circuit being simulated, specifying initial conditions for every capacitor and inductor in the circuit could easily be impractical.

When uic is not specified, SPICE uses voltage sources and 1 Ω resistors to force initial conditions. Consider trying to apply an initial condition of 1 V to C_1 . The simulator connects the series combi-

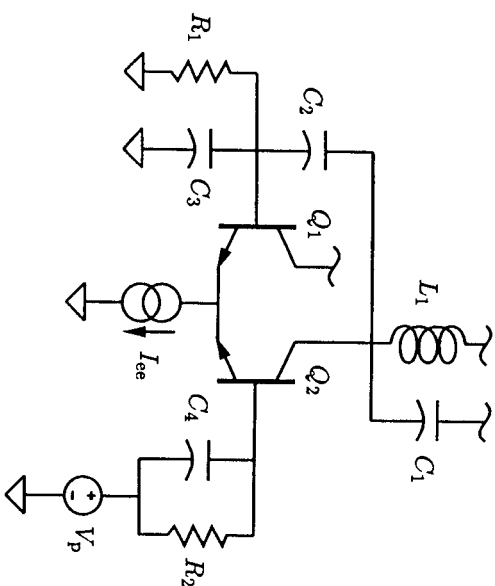


Figure 4.46: An emitter-coupled pair oscillator.

nation of a 1 V source and a 1 Ω resistor in parallel with C_1 . The simulator then computes the DC solution to this modified circuit and uses it as the initial condition for the original circuit. Unfortunately the inductor shorts the source and resistor used to force the initial condition and so $V_{C_1} = 0$. In addition, the current from the source and resistor used to force the initial condition travels through L_1 , and so it starts with an initial condition of 1 A. The huge initial current for L_1 cause the oscillation to start with a very large amplitude. This is shown in Figure 4.47 on page 214. On some circuits, it takes a long time for the amplitude to decay to the point where the oscillator functions normally and the large amplitude can cause a severe loosening of tolerances (see Section 4.3.2.1 on page 183). In general, having the simulator compute non-uic-based initial conditions on an LC resonator (series or parallel) is likely to result in huge oscillations.

```
// BJT ECP Oscillator
global gnd
simulator lang=spectre

Vcc (vcc gnd)      vsource dc=5 type=dc \
                    va10=0 va11=5 rise=10ns delay=10ns
Iee (e gnd)         isource dc=1mA
Q1 (vcc b1 e)       npn
Q2 (out b2 e)       npn
L1 (vcc out)         inductor l=1uH
C1 (vcc out)         capacitor c=1pf ic=1
C2 (out b1)          capacitor c=272.7pF
C3 (b1 gnd)          capacitor c=3nF
R1 (b1 gnd)          resistor r=10k
C4 (b2 in)           capacitor c=3nF
R2 (b2 in)           resistor r=10k
VP (in gnd)          vsource mag=1 type=dc \
                    amp1=0.1 freq=10MHz damp=5kneppers
```

```
ic b1=1
model npn bjt type=npn bf=80 rb=100 va=50 \
      cjs=2pf tf=0.3ns tr=6ns cje=3pf cjc=2pf
osc1 tran stop=4us ic=dev maxstep=10ns
osc2 tran stop=80us ic=node maxstep=10ns
EnablePerturbation alter dev=VP param=type value=sine
osc3 tran stop=10us ic=dc maxstep=10ns
```

Netlist 4.2: Emitter-coupled pair oscillator and the three analyses that generate the results for the following three figures.

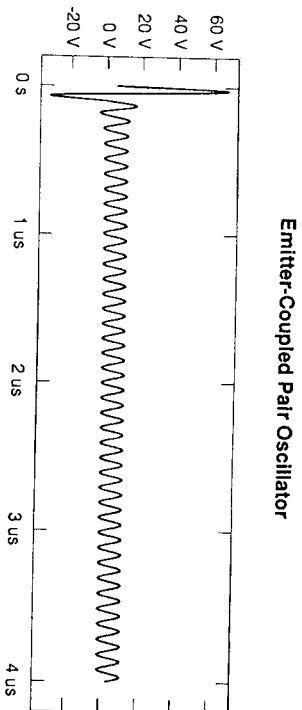


Figure 4.47: Assigning a non-*uic*-based initial condition to a capacitor in the *LC* resonator of Figure 4.46 on page 212 results in a huge response because of the large current forced through the inductor.

Assigning initial conditions to C_2 , C_3 , or C_4 trades the problems just discussed with another serious problem. Initial conditions on C_2 , C_3 , or C_4 disturb the bias point of the oscillator, and the slow time constants at the base results in the circuit taking a very long time to recover. For example, setting an initial condition of 1 V on C_3 results in all bias current initially going through Q_1 with Q_2 turned off completely. The time constant at the base of Q_1 is roughly 30 μ s. It takes several time constants for the circuit to become balanced enough to support oscillation. This is shown in Figure 4.48.

A better approach to starting this oscillator is to use an independent source (V_P) to couple a signal into the base of Q_2 . Exciting the circuit with a small damped sinusoid at the estimated frequency of oscillation is particularly appropriate and effective in this case because

1. It couples strongly to the mode of oscillation, increasing the likelihood of starting the oscillation.
2. The stimulus has a small DC level relative to its amplitude and so does not excite the slow time constants in the bias circuitry

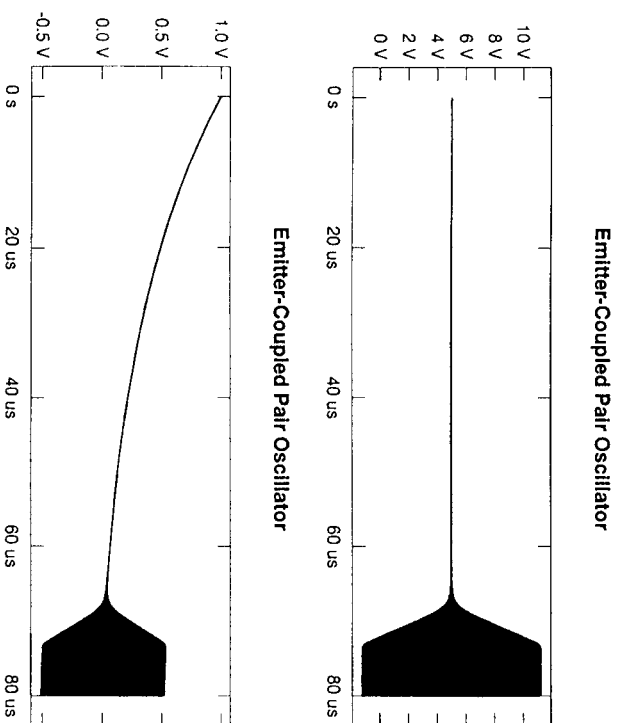


Figure 4.48: Assigning an initial condition to C_2 , C_3 , or C_4 in the oscillator of Figure 4.46 on page 212 disturbs the bias point. The slow time constants present in the bias circuitry result in the circuit taking a long time to settle into steady state. The top waveform is the output of the oscillator (collector of Q_2) and the bottom waveform is the base of Q_1 .

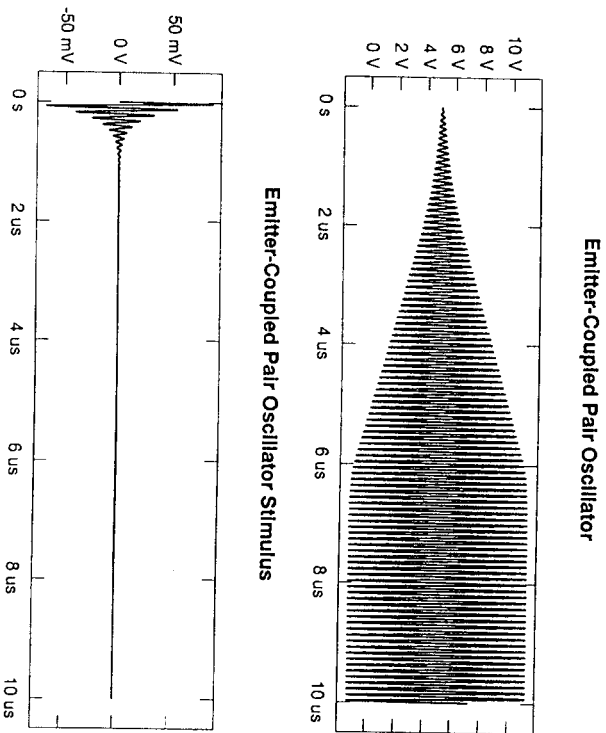


Figure 4.49: Injecting a small damped sinusoid at the estimated frequency of oscillation (bottom) into the base of Q_2 is particularly effective at starting the oscillator (top).

to any significant degree.

The response of the circuit to a small damped sinusoid at the base of Q_2 is shown in Figure 4.49.

Build-up of Oscillation Once an oscillator is perturbed, the oscillation grows and approaches its steady-state amplitude asymptotically. If the oscillator is started with a very small perturbation, then during the initial portion of the growth phase, the oscillation could be quite small. If the signal amplitudes are much smaller than the

DC levels in the circuit, they may not be of sufficient size to force the time-step control algorithm to take smaller time steps, then the simulator could portray the initial growth phase poorly. It is even possible that large time steps could result in the effect of the perturbation being lost and having the oscillator fail to move away from its equilibrium point. Remember that some integration methods exhibit artificial numerical damping that can quench the oscillation if the time step is large enough. In order to accurately predict the solution during its initial growth phase, it is necessary to force the simulator to use small time steps using the maximum time-step parameter on the transient analysis (tightening `reltol` is not as effective in this situation). Typically, a reasonable value for the maximum time step would be about one-twentieth of the expected period of oscillation. In addition, you should consider using trapezoidal rule when simulating an oscillator because it does not exhibit artificial numerical damping like Gear2 or backward Euler. Failing to force the simulator to use a small time step during the build-up phase results in

1. Ragged waveforms.
2. Grossly inaccurate prediction of the time constants associated with the turn-on transient.
3. Extremely inaccurate prediction of the delay between when the oscillator is first perturbed and when it settles.
4. The simulator may not even predict oscillation because
 - (a) The delay induced by the simulator is longer than the transient analysis interval.
 - (b) Artificial numerical damping of the integration method quenches the oscillation.

These points are illustrated in Figures 4.50 through 4.53.

Steady State Oscillators generally have very high- Q resonators by design. High- Q resonators by their very nature store energy for very long periods of time. Thus, errors created during a simulation tends to be stored and accumulated in the resonator.

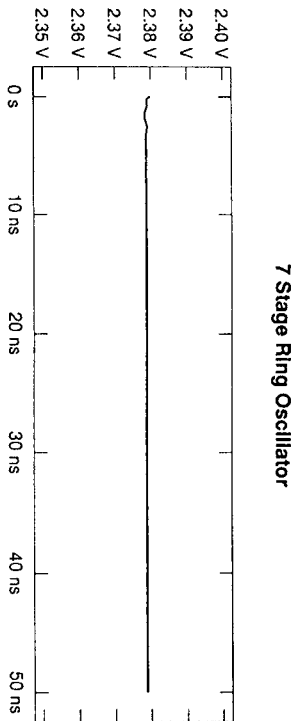


Figure 4.50: Response of the 7-stage ring oscillator to a small initial condition simulated with Gear2. When used with a large time step, Gear2 (and backward Euler) has sufficient artificial numerical damping to quench the oscillation. The initial condition is not large enough to shrink the time step, so the oscillation fizzes.

For oscillators there is an infinitely long time constant associated with the phase because phase is the integral of frequency. Thus, errors in the simulator cause the phase to drift (see Section 4.2.4.1 on page 160 for a discussion on how error accumulates with long time constants). In addition the low order polynomial used by the integration method consistently underestimates the curvature of a sine wave because the sine wave has an infinite number of nonzero derivatives. This results in a systematic bias that results in the phase drifting slightly on each step in such a way that the frequency is too low. Thus, it is best when simulating oscillators to be conservative and tighten `reltol`, say to 0.0001.

Relaxation oscillators cause another difficulty. Because they generate sharply discontinuous signals, but do not generate break points, you may find the corners are missing on the waveforms, as shown in Figure 4.39. This problem cannot be eliminated entirely, but it can be reduced by tightening `reltol`, which serves to reduce the time step.

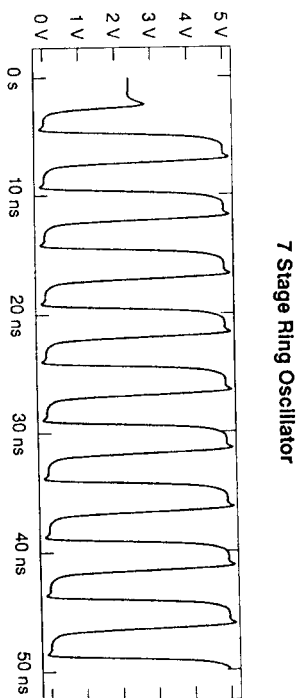


Figure 4.51: Response of the 7-stage ring oscillator to the small initial condition of Figure 4.50 on the preceding page simulated with Gear2. The oscillation starts this time because the maximum time step was specified to be no larger than $T/20$. With the smaller time step, Gear2 did not exhibit enough damping to quench the oscillation and the initial growth phase is portrayed accurately.

4.4.3 Unstable Circuits

Unstable circuits present special challenges to circuit simulators that you need to be aware of to avoid unpleasant surprises. Without additional care, the simulator may not give any indication that the circuit is unstable.

DC analysis computes equilibrium points and does not distinguish between stable and unstable equilibrium points. If a simulator finds an unstable equilibrium point, it will not move away from that point unless the circuit is somehow perturbed. This is why simulated oscillators do not self start. See Section 4.4.2 on page 209 for more information on how to trigger unstable modes.

Some integration methods exhibit artificial numerical damping that sometimes causes stable responses being computed for unstable circuits. Consider the circuit of Figure 4.54 on page 221 [kinge94]. The positive feedback causes this circuit to be unstable. The trans-

7 Stage Ring Oscillator

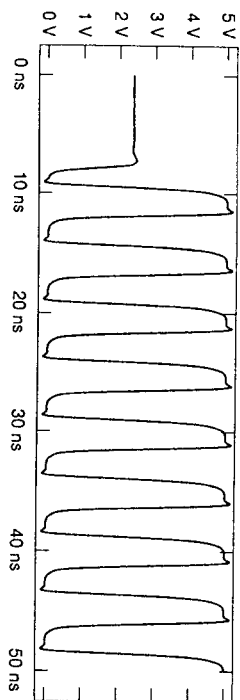


Figure 4.52: Response of the 7-stage ring oscillator to the small initial condition of Figure 4.50 on page 218 simulated with trapezoidal rule, which does not exhibit artificial numerical damping, even with large time steps. It takes a long time for the oscillation to build up because the oscillation is too small compared to the DC level to be noticed by the time-step control algorithm. This delay is not representative of what would happen with the real circuit. But even with the large time steps initially, the oscillation starts with trapezoidal rule as long as the initial delay is not longer than the transient analysis interval.

fer function of this circuit is easily computed to be:

$$\frac{V_3(s)}{V_1(s)} = \frac{\frac{A}{A-1}}{1 - s \frac{RC}{A-1}} \quad (4.61)$$

This transfer function shows that the positive feedback has two effects. Not only does it move the pole due to RC into the right-hand plane, but it also moves it up in frequency by a factor of $A - 1$. The circuit is driven with a unit step that has a $20\mu\text{s}$ transition time. The simulator takes about $1\mu\text{s}$ time steps in order to follow the transition. These steps are considerably larger than the $RC/A - 1 = 5\text{ns}$ unstable time constant, and there lies the problem. Substantial artificial numerical damping is generated by the integration method when the time step is much longer than the time constant.

7 Stage Ring Oscillator

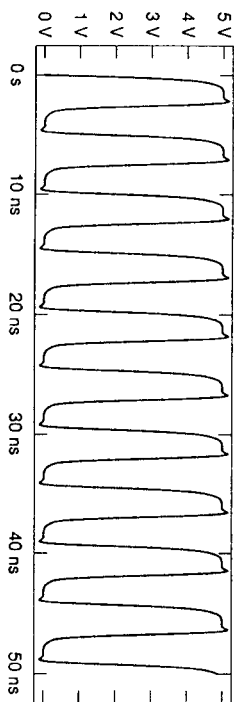


Figure 4.53: Response of the 7-stage ring oscillator to a large initial condition simulated with Gear2. The large initial condition generated an initial response that is large enough to immediately cause the simulator to shrink the time step, which reduces Gear2's artificial numerical damping to the point where it does not quench the oscillation.

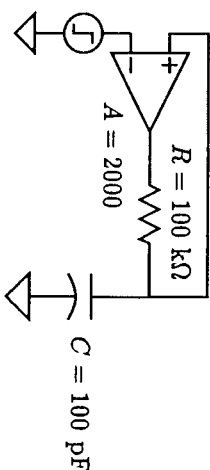


Figure 4.54: The large amount of positive feedback in this circuit causes the pole at $1/RC$ to be moved up in frequency and into the right-hand plane. The resulting fast unstable time constant is easily damped out by artificial numerical damping in the simulator.

Incorrect Step Response for Positive Feedback Amplifier

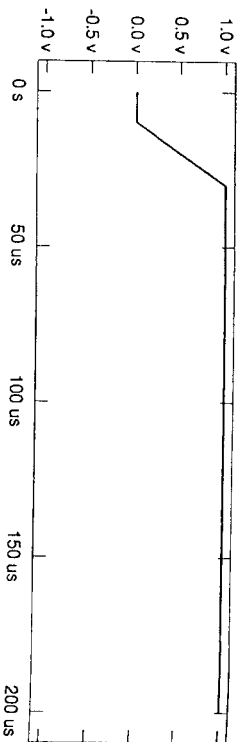


Figure 4.55: The response computed for the circuit of Figure 4.54 on the preceding page using trapezoidal rule. The unstable mode that results from the positive feedback in the circuit is completely damped by the backward Euler steps taken after each break point.

The circuit in Figure 4.54 was simulated with trapezoidal rule and the resulting output is shown in Figure 4.55. Notice that even though the circuit was driven with a step change on the input, the output shows no trace of a growing exponential that is characteristic of a unstable pole. Trapezoidal rule does not exhibit artificial numerical damping, so it is surprising that the pulse stimulus does not excite the unstable mode. However, the simulator places break points at both corners of the pulse stimulus, and backward-Euler is always used on the time step that immediately follows a break point. The large backward Euler step completely damps the unstable mode at each break point.

There is no way to prevent the simulator from using backward Euler after a break point, and there is no way to avoid the break point. However, this problem is easily prevented by simply causing the simulator to take a smaller time steps. Tightening `reltol` is an easy way to do that, as is reducing the transition time of the pulse. It is also possible to force the simulator to use a smaller time step by reducing `Tmax`, but that is not particularly efficient in this case be-

Correct Step Response for Positive Feedback Amplifier

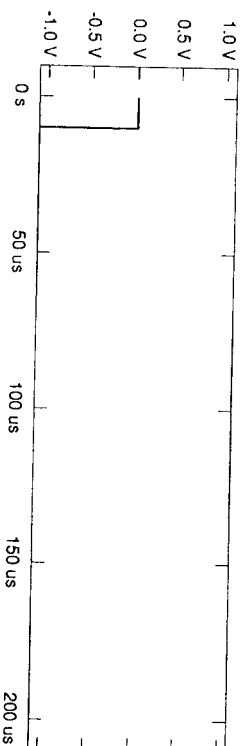


Figure 4.56: The response computed for the circuit of Figure 4.54 on page 221 using trapezoidal rule with `reltol = 10-4`. Tightening `reltol` results in smaller time steps and less artificial numerical damping. This response clearly indicates that the circuit is unstable.

cause it would force a small time step everywhere even though the small time step is only needed immediately after the break points. Figure 4.56 shows the results computed with `reltol` tightened to 10^{-4} .

While it is clearly possible for circuit simulators to generate ‘stable’ results for unstable circuits, it is rare. Two things must occur, first a high-frequency unstable pole must exist, and second, the smallest time step taken during the transient analysis must be long compared to the time constant of the unstable pole. Unexpected unstable poles are themselves rare, but when they occur they are usually detected eventually because designers generally drive their circuits with wide-band signals to determine their dynamic response. The wide-band signals force the simulator to take small time steps, which usually reduces the artificial numerical damping to the point where the unstable response is not damped out.

The few cases where the simulator does generate stable responses to unstable circuits tend to occur in overly simplified circuits, such as Kinget’s circuit above. It is also a problem for macromodels, partic-

ularly opamp macromodels. In fact, some of the opamps in a suite of macromodels released by a major semiconductor vendor contain negative capacitors, which directly cause unstable poles. These unstable poles have gone unnoticed by the developers of the macromodels because the poles are usually at such high frequency, that they do not cause a problem. However, if the opamps are driven with pulses that have especially fast transition times, if the tolerances are unusually tight, or if the circuit exhibits convergence problems, then the simulator can take time steps small enough so that the unstable mode is not damped. In this situation, the waveforms shoot off to infinity so fast that the simulator cannot follow. In this situation, the transient analysis would fail is such a way to make it seem as if the simulator is having convergence difficulties. Inspect the computed waveforms carefully, particularly at the last instant of the simulation, and look for a rapid exponential growth. If you find one, it likely your circuit contains an unstable pole.

To reduce the chance of an unstable pole escaping notice, it is always a good idea to drive your circuit with a fast rise time step and simulate with tight tolerances.

4.4.4 Charge-Storage Circuits

Charge-storage circuits are an important and difficult class of circuits to simulate. They include dynamic memories, switched-capacitor and switched-current circuits, and charge-coupled devices (CCDs). The distinguishing feature of charge-storage circuits is that they store charge on capacitors for periods of time and the leakage of charge from the capacitors is expected to be minimal. The circuits are difficult to simulate because of the long time constants associated with the circuitry that stores charge. Any errors made in the charge computed by the simulator are accumulated on the charge-storage capacitors. Thus, the key to accurate simulation of charge-storage circuits is reducing the error in charge that is stored on the capacitor.

As described in Section 4.2.5 on page 167, the nonlinear capacitors associated with MOSFET models that do not conserve charge are the dominant source of charge error when used in charge-storage circuits. In particular, the Meyer capacitance model used in MOS levels

1, 2, and 3 suffers from charge conservation problems, though level 2 also supports the charge conserving Ward-Dutton model. In addition, Spectre provides the Yang-Chatterjee and BSIM charge models, which are both charge conserving. However, even when charge conserving models are used, charge conservation errors still exist because the circuit simulator itself does not conserve charge.

On each time step, the node voltages are computed with Newton's method, which is an iterative method that continues to refine the solution until the convergence criteria are met. The criterion that is germane to charge conservation is the one that insists that Kirchhoff's current law is sufficiently satisfied. The simulator insists that KCL be satisfied to within `reltol` and `abstol`. If `reltol` = 0.001 and `abstol` = 1pA, then there could be as much as a 0.1% + 1pA mismatch of the current flowing on to each node (this assumes that (2.7) and (2.8) are used as the convergence criteria, if not, the error can be larger). One way to think of this, is that the simulator exactly solves a slightly modified circuit exactly. The circuit the simulator is solving exactly is the original circuit with small current sources connected to each node. The value of each current source must be less than 0.1% + 1 pA, and varies on each time step. Thus, in the original circuit, it is as if a small amount of current (or charge) is being created or destroyed on each time step. This is an unavoidable source of charge error. However, it is much less significant than the charge error caused by capacitance-based nonlinear capacitor models.

This source of charge error can be reduced by tightening `reltol` and `abstol`. Notice that this error is a function of the convergence criteria, not the time step. Thus if the normal accuracy of the simulator is acceptable and you are using a charge-conserving model, but you can tolerate very little charge error, it is possible to tighten the convergence criteria without tightening the truncation error criterion, allowing you to get much better charge conservation, with little increase in simulation time. Roughly, the charge error is proportional to `reltol` whereas the time step is proportional to `reltol` × `trtol`. Decreasing `reltol` by a factor of ten while simultaneously increasing `trtol` by a factor of ten causes a reduction of charge error while keeping the time step size roughly the same. In Spectre, use `Iteratio` rather than `trtol`.

Another source of charge error in a simulator is due to `gmin`, which is a small conductance (1pSiemens) installed across each nonlinear device. Spectre allows `gmin` to be set to zero, eliminating this error (see Section 2.3.2.1 on page 39 for additional information). Lastly, normal circuit behavior can masquerade as a charge conservation problem. For example, the normal leakage paths inside the semiconductor devices should be examined to make sure they are not the source of the presumed charge conservation problems. MOSFETs have junction leakage current from the source and drain to the bulk. This leakage current roughly doubles for every 10 Celsius rise in temperature.

4.4.5 Sinusoidal Circuits

The sinusoid is a particularly useful analog test signal, and so circuit simulators are often called upon to compute sinusoidal signals. Occasionally, they do so poorly, and so it is worth discussing what goes wrong and how to avoid it.

The most common problem results when the simulator uses too few time points during each period. In particular, if time points are not placed near the peaks of the sine wave, then when many periods are plotted, as in Figure 4.57, the amplitude seems to vary randomly. This problem is easily resolved by simply specifying the maximum time step to assure at least 10 to 20 time points per cycle.

A more extreme example of this occurs when simulating the circuit of Figure 4.58 on page 228. SPICE2 generates the signal shown. Notice that the apparent frequency of the waveform is 2 MHz, not the 102 MHz of the source. SPICE severely undersamples the waveform, causing it to alias down to 2 MHz. Normally truncation error causes SPICE to take time steps small enough to avoid severe undersampling. In this case however, the size of the 102 MHz signal on the capacitor is very small because at this frequency the impedance of the capacitor is very much smaller than the impedance of the resistor. The magnitude of the signal across the capacitor is very small, and therefore so is the truncation error. Without truncation error, there is no control on the time step. As above, the remedy for undersampling is to specify the maximum time step to be no larger than the period over 10 to 20. For more examples of undersampling with SPICE, see Section 5.3.2.

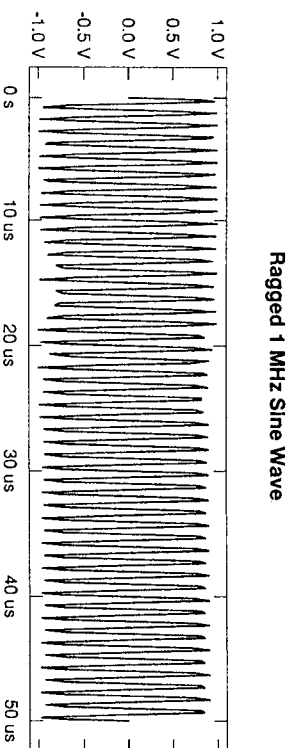


Figure 4.57: Constant amplitude sine wave whose amplitude seems to vary because too few time points were used by simulator.

Severe undersampling occurs mostly in circuits that contain few, or no, capacitors or inductors. This most often occurs in circuits that have been heavily simplified, such as circuits that contain macromodels or functional blocks. When simulating such circuits it is generally a good idea to specify a reasonable maximum time step.

Spectre automatically limits the size of a time step when a sinusoidal source is present and active. It forces between 4 and 20 time points per cycle, depending on the setting of the `errpreset` transient analysis parameter.

4.4.6 Macromodels

Macromodels pose a particularly difficult challenge for circuit simulators because many are so poorly designed that they can actually run more slowly and have more convergence problems than the circuits they model. In addition, they may cause the simulator to fail and occasionally contain instabilities that cause the solution to explode.

Problems with macromodels can be put into three categories:

1. Macromodels can be strongly discontinuous, which causes con-

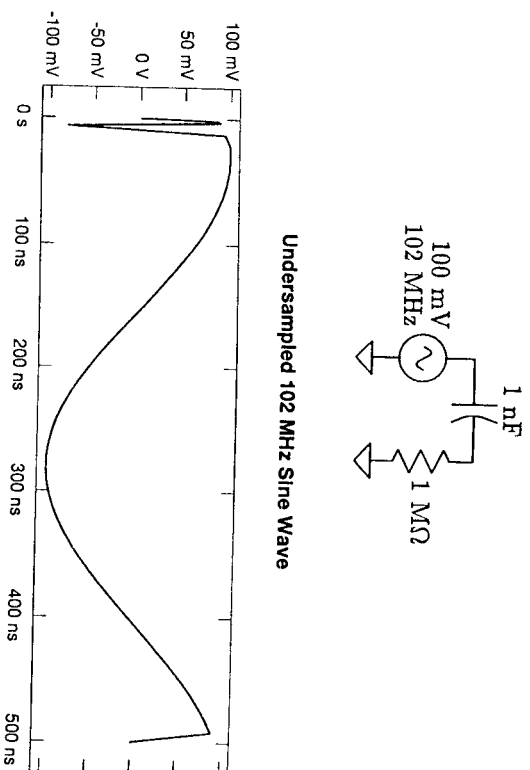


Figure 4.58: The figure at the top is a simple RC circuit driven at 102 MHz. The waveform below is the response computed by SPICE. The waveform as plotted appears to be at 2 MHz because of severe undersampling. The abrupt transitions at the beginning and end of the transient analysis interval result from SPICE taking smaller time steps at the ends of the interval.

- vergence problems.
2. Some macromodels contain unstable time constants, which if excited cause the internal voltages or currents to shoot towards infinity, causing the simulator to fail.
3. Macromodels are routinely written in such a way that they generate extremely large internal voltages or currents. This is usually not a problem with SPICE, but it can be disturbing to find 100 kV or 1 kA signals in an integrated circuit. It can also cause some newer simulators, such as Spectre, some problems because their algorithms are more highly optimized for real

4.4. Applications of Transient Analysis

229

circuits.

In addition, macromodels have a very limited range of effects that they model accurately. For example, few macromodels model power dissipation, noise performance, or power supply rejection accurately. Unfortunately, few macromodels come with documentation indicating what effects are modeled, and what are not. When in doubt, only count on effects being modeled accurately if they are documented as being modeled, or you have verified the models.

Abrupt Discontinuities Developers of macromodels often get into trouble and create models that are difficult to simulate when they stray from the basic topology of the circuit being modeled. In the name of simplicity, some extremely convoluted macromodels have been written, and the area that seems to generate the most problems is modeling the output stages of common operational amplifiers. For example, Figure 4.59 on the following page shows an output stage found in a typical opamp macromodel. The disconnected pieces of circuitry that contain points *C* and *D* are current steering circuits used to accurately model the current in the power supplies. They assure that the macromodel appears to source current from the positive supply and sink current to the negative supply. Current steering circuits constructed from ideal current sources and diodes almost always cause convergence difficulties because of the abrupt change in voltage across the diodes as the current through the source goes through zero. In this circuit, the voltage on points *C* and *D* jump almost instantaneously between $V_{cc} - 0.7V$ and $V_{ee} + 50V$ at the current zero crossings. The abrupt changes in the voltage on points *C* and *D* cause two problems. First, convergence is troublesome because of the rapid change in voltage across the strongly nonlinear diodes. Second, the simulator has to take infinitesimally small time steps during the current zero crossings if it is controlling the time steps to accurately follow the signals on points *C* and *D*. It is important to realize that this type of circuit causes the simulator to run slowly as the best case, and may cause the simulator to fail in the worst case.

A better approach is to model that output stage using a topology similar to that of the actual circuit. Doing so results in the model

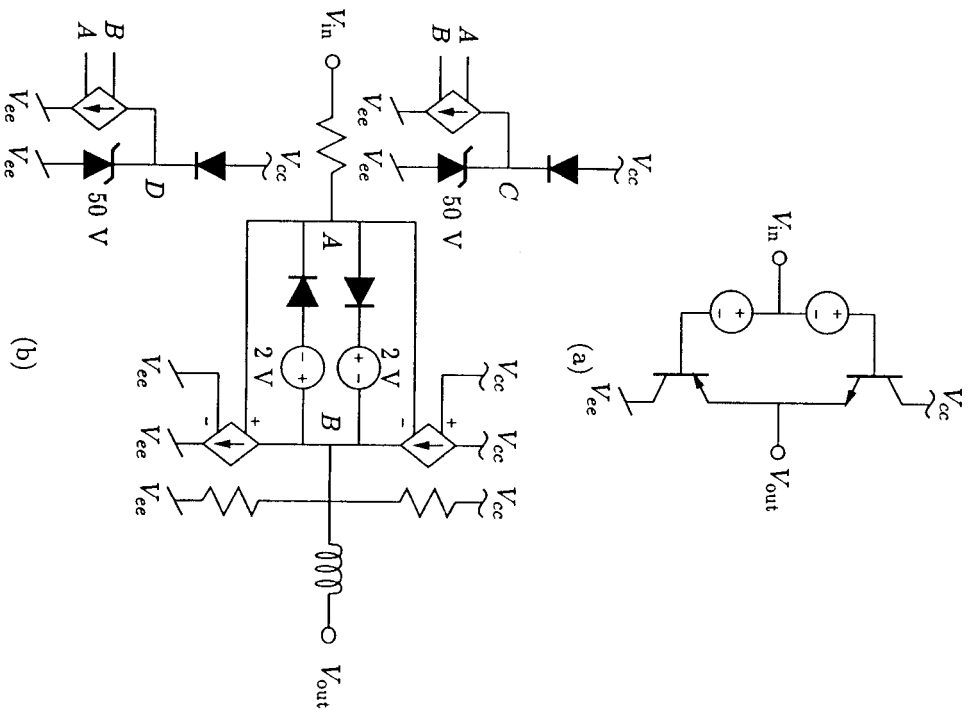


Figure 4.59: An idealized output stage for a bipolar operational amplifier (a) and one possible macromodel implementation (b). The macromodel causes problems for circuit simulators because of the strongly discontinuous nature of the circuit at points C and D.

naturally modeling the power supply current behavior without additional dangerous circuitry. Some variation on the circuit shown on the left side of Figure 4.59 would work best. The voltage sources could be replaced with diodes and constant current sources, to prevent the bias current from running away as temperature increased. Or, the bipolar transistors could be replaced with large JFETs.

Unstable Time Constants In general it is dangerous to use negative resistors, capacitors, and inductors in a macromodel. Though some simulators refuse to allow resistance, capacitance and inductance to specified to be negative, there is no fundamental reason why the simulator needs to do so. In fact, it is usually done to try to keep the user out of trouble. When a negative resistance is combined with a positive capacitance or inductance, or when a positive resistance is combined with a negative capacitance or inductance, an unstable circuit is created. If the time-constant of the unstable mode is small compared to the transient analysis interval, then the solution can grow so large that the simulator fails due to numerical overflow. If the time-constant is very small compared to the smallest time step used in a transient analysis, then the instability may be damped by artificial numerical damping of the integration method (see Section 4.4.3 on page 219). However, such situations should be avoided because something may force the simulator to take a small time step, which would eliminate the damping and cause the simulation to fail.

Macromodels typically use negative component values when modeling the intermediate stage of opamps. The intermediate stage of an opamp acts like a transconductance amplifier with feedback provided by the compensation capacitor, as shown in Figure 4.60 on the next page. At intermediate frequencies the main signal path is through the transconductance amplifier, with a feedback signal flowing backward through the capacitor. At these frequencies, the circuit is acting like an integrator. Thus, there is a left-hand plane pole at $f = 0$ that causes the output appears to lead the input signal by 90° (because the transconductance amplifier inverts the signal, which introduces -180° of phase shift, and the pole introduces another -90° for a total of -270° , which is equivalent to 90°). At high frequencies, the capacitor shorts out and the input signal feeds forward through the

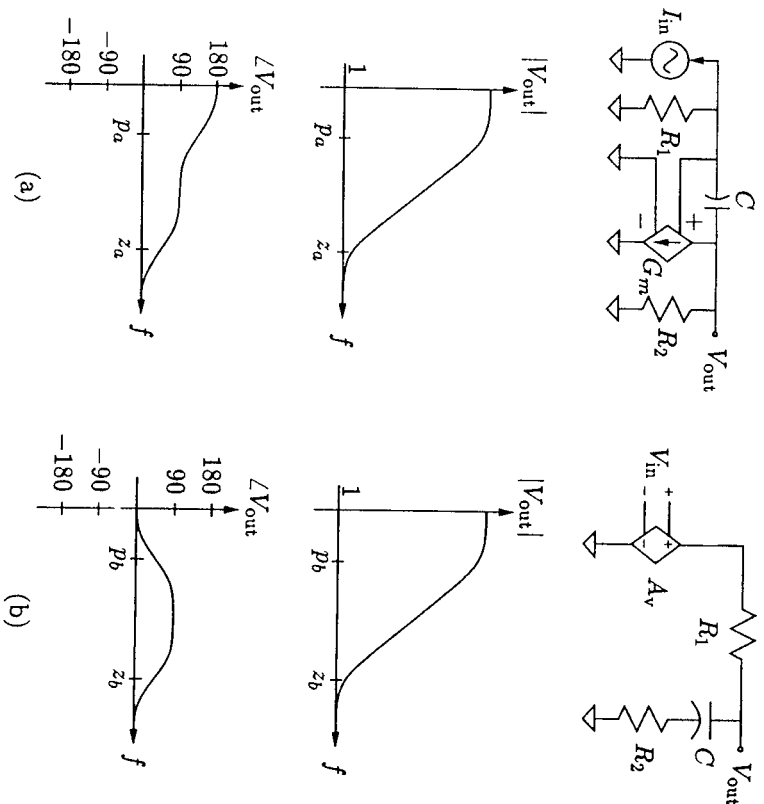


Figure 4.60: On the left (a) is a simplified transconductance amplifier with a compensation capacitor that is representative of the intermediate stage of an opamp. This circuit has a left-hand plane pole at $p_a = -2\pi/(R_1 + R_2 + G_m R_1 R_2)$ and a right-hand plane zero at $z_a = -2\pi G_m / C$. A common macromodel for the intermediate stage is shown on the right (b). This macromodel has a pole at $p_b = 2\pi/(C(R_1 + R_2))$ and a zero at $z_b = 2\pi/C R_2$. This circuit uses a negative capacitor to implement the right-hand plane zero, which also causes the pole to fall in the right-hand plane, making the macromodel unstable.

capacitor and causes the output to follow the input with no phase shift. Thus the capacitor introduces a zero with -90° of phase shift. Such a zero is a *right-hand plane* zero.

Figure 4.60b shows a macromodel implementation of the intermediate stage of Figure 4.60a. This circuit implements both a pole and a zero. If R_1 is assumed to be much larger than R_2 , then the pole frequency is $p_b \approx 2\pi/C R_1$ and the zero frequency is $z_b = 2\pi/C R_2$. A right-hand plane zero is modeled by making C negative. However, this also moves the pole into the right-hand plane, making the circuit unstable.

A better implementation would be to actually implement the topology of Figure 4.60a. No negative components would be needed to implement the right-hand plane zero. Another approach would be to use the circuit of Figure 4.60b, except make R_2 negative rather than C . Since R_1 and R_2 are in series, and assuming that $R_1 \gg R_2$, then the total resistance seen by the capacitor is positive, making the circuit stable. But R_2 negative does result in the zero being in the right-hand plane.

Large Internal Voltages and Currents Some macromodels have extremely high internal voltages or currents present internally. While SPICE has no difficulty with large internal voltages and currents, it is still not recommended to implement macromodels in such a way that they exhibit extremely large voltages or currents if no other reason than some users find it disturbing to see such voltages and currents. In addition, the newer circuit simulators, such as Spectre, have been optimized to be very efficient, and the mix of large and small voltages or currents can violate the assumptions used to make the optimizations, causing the simulator to be inefficient, or the results to be inaccurate. If you need to use Spectre on macromodels that generate huge internal voltages or currents, set the `relref` transient analysis option to `alllocal`.

Simulating Macromodels with Spectre Spectre is highly tuned for simulating real electrical circuits. Macromodels typically violate many of the assumptions made when tuning Spectre's algorithms,

and so without intervention by the user, Spectre can run slowly or generate results with degraded accuracy on circuits containing some macromodels. Spectre supports an option that adjusts its default settings in order to perform more efficiently and more accurately on circuits that contain macromodels. When simulating a circuit containing a macromodel, it is recommended that the `macromodels` option be set if the macromodel exhibits the problems described above.

4.4.7 Distribution Networks

One situation where artificial numerical damping (Section 4.2.2.3 on page 147) might have significant impact is when simulating power and clock distribution networks. Distribution networks consist of long, inductive lines terminated with capacitors. They are generally underdamped and often quite underdamped. The underdamped nature of distribution networks coupled with the abruptly changing currents that flow into them cause ringing. Excessive ringing on a power of clock distribution line is cause for concern because it results in the entire circuit misbehaving in a mysterious manner.

In this situation, the artificial numerical damping exhibited by backward Euler or Gear's method damps out some or all of the ringing on a power of clock distribution network. This damping is made more likely because the bias level on the distribution network is large compared to the ringing. The large bias level desensitizes the time-step control algorithm to the much smaller ringing, resulting in the simulator taking large time steps relative to the ringing. Thus, when simulating inductive distribution networks, where ringing is a concern, use of Gear's methods, and especially backward Euler should be avoided. Use trapezoidal rule instead. If trapezoidal rule cannot be used, then it may be necessary to tighten `reltol` or the maximum time step in order to reduce the artificial damping to negligible levels.

4.4.8 Large Circuits

It is difficult to find problems in large circuits simply because of the large amount of data one must wade through. Some suggestions

on how to efficiently find problems on large circuits were given in Section 2.4.2 on page 45. Additional suggestions that are specific to transient analysis are given here.

If a circuit seems to run very slowly or seize up in transient analysis, an explanation is sometimes found by looking at the supply currents. For example, consider the waveforms shown in Figure 4.61 on the following page. When computing this waveform, the simulator progressed quickly through the first 10% of the transient analysis interval and then appeared to hang. Careful inspection at the right end of the waveform shows a very fast, and very small, trapezoidal rule ringing that is the cause of the problem. The rapid trapezoidal rule ringing causes the simulator to take infinitesimally small time steps, and so makes it run very slowly. Switching to Gear2 solves the problem.

It is also possible to find unexpected oscillations by inspecting the supply currents. If the supply currents show steady undulations even after the circuit should have reached an equilibrium point, the circuit probably contains an oscillator.

Autonomous Clock Generators Often times large digital circuits contain oscillators that act as clock generators. These oscillators can significantly slow the simulation because the simulator must keep the time step small to accurately follow the constantly changing waveforms in the oscillator, even though the rest of the circuit may be quiescent and waiting for the next clock transition. To accelerate the simulation in this case, remove the oscillator from the large circuit and simulate it alone. Import the oscillator output waveform into the large circuit (now with the oscillator removed). This often dramatically reduces the number of time points needed to simulate large circuit, and can speed-up the simulation by a factor of 2 or 3.

4.4.9 Accurate Current Measurements

As a rule, currents are not calculated as accurately as voltages by circuit simulators. One reason is that the current through capacitors is proportional to the derivative of the voltage across the capacitor.

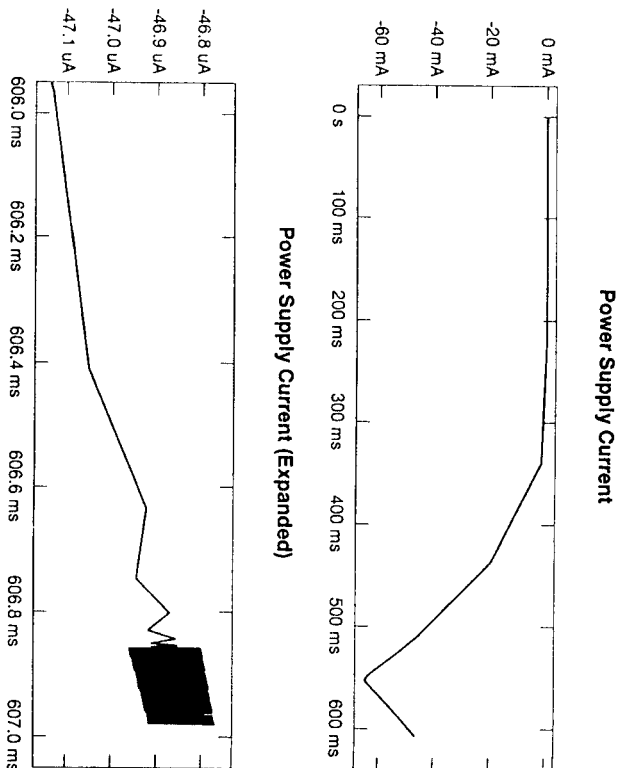


Figure 4.61: The circuit simulator seemed to seize up when simulating a large circuit and a quick check of several node voltages failed to turn up a clue of what was going wrong. Plotting the power supply current (above) did show that a severe trapezoidal rule ringing problem existed somewhere in the circuit. Without pinning down where the problem was occurring, it was possible to avoid it by simply switching the integration method to Gear2.

Any error present in the voltage appears amplified in the current (because the derivative amplifies change). Another reason why currents are computed less accurately than voltages is that in general the time step is not chosen to maintain the accuracy of current waveforms. Errors in currents through components other than inductors do not accumulate. Therefore, it is not essential to choose the time step to control the error in the currents. Because currents are rarely needed, because controlling error in current requires smaller time steps and so is more expensive, and because error in current does not accumulate, SPICE and its progeny do not control the time step to maintain the accuracy of current waveforms.

Unlike SPICE, Spectre chooses the time step to control error in the current. However, in its default configuration, it uses rather loose tolerances. Besides the normal parameters that affect the time-step control algorithm, such as `reltol` and `maxstep`, two other parameters directly affect how tightly error is controlled in currents, `errpreset` and `step`, the suggested time step. `errpreset` takes three possible values, `liberal`, `moderate`, and `conservative`. As shown in Table 4.1 on page 188, `errpreset` presets the values of several error control parameters to allow a user to quickly choose a tradeoff between accuracy and speed that favors accuracy (`conservative`), favors speed (`liberal`), or balances the two (`moderate`). When `errpreset=liberal`, error in current is ignored completely when choosing the time step. With `errpreset=moderate`, it is taken into account, but the error is controlled to rather loose tolerances. When `errpreset=conservative`, the error in currents is controlled tightly.

The suggested time step (`step`) also plays an important role, though perhaps not the one you might expect. The suggested step is the minimum time step that is taken simply to control the error in a current not associated with an inductor and to control the error in a voltage not associated with a capacitor. Thus, when using Spectre, the suggested time step is the minimum time step that will be used to follow rapid changes in a current waveform. This is a necessary parameter because currents can jump discontinuously, such as when a voltage source and capacitor are connected in parallel and the voltage on the source starts to ramp up. Without the `step` parameter, Spectre would be forced to take infinitesimally small time steps in a

futile attempt to accurately follow the infinitely fast transition.

Two important points should be recognized. First, it is not often when a very small time step would be needed to control error in a current. Second, setting the suggested step in no way affects the maximum time-step size. So choosing a small suggested step does not appreciably slow the simulator in most cases. In this way Spectre is different than SPICE. Spectre uses the suggested time step as a form of lower bound on the time step, while SPICE uses it as an upper bound.

4.4.9.1 Power Dissipation of a CMOS Gate

Consider trying to measure the power dissipation of a CMOS nand gate. Since the gate is driven by a constant 5V supply, measuring the power dissipation is equivalent to measuring the supply current. The only significant amount of supply current flows during the transitions. The current pulses are very short, and the current waveform must be computed carefully to accurately compute the power. Figure 4.62 on the next page shows the current computed by SPICE with `reltol=10-3` and `reltol=10-6`. SPICE is not controlling the error in the current, and so the waveform is not accurate. Figure 4.62 also shows the current waveform computed by Spectre with `errpreset=conservative` and `step=1 ps`. Spectre produces better results because it chooses the time step to directly control the error in the current. It is also using second-order Gear rather than trapezoidal rule, and so avoids the trapezoidal ringing.

4.4.10 Strobing

Many types of analog circuits are driven with high frequency repetitive signals, but their responses must be observed over a very long interval. This results either from very long time constants in the circuit or from slow repetitive signals present either as stimulus or as response. Circuits that fit this description include switched-capacitor and switched-current circuits, switching-power converters, mixers, and modulators, as well as circuits that contain these circuits, such as phase-locked loops.

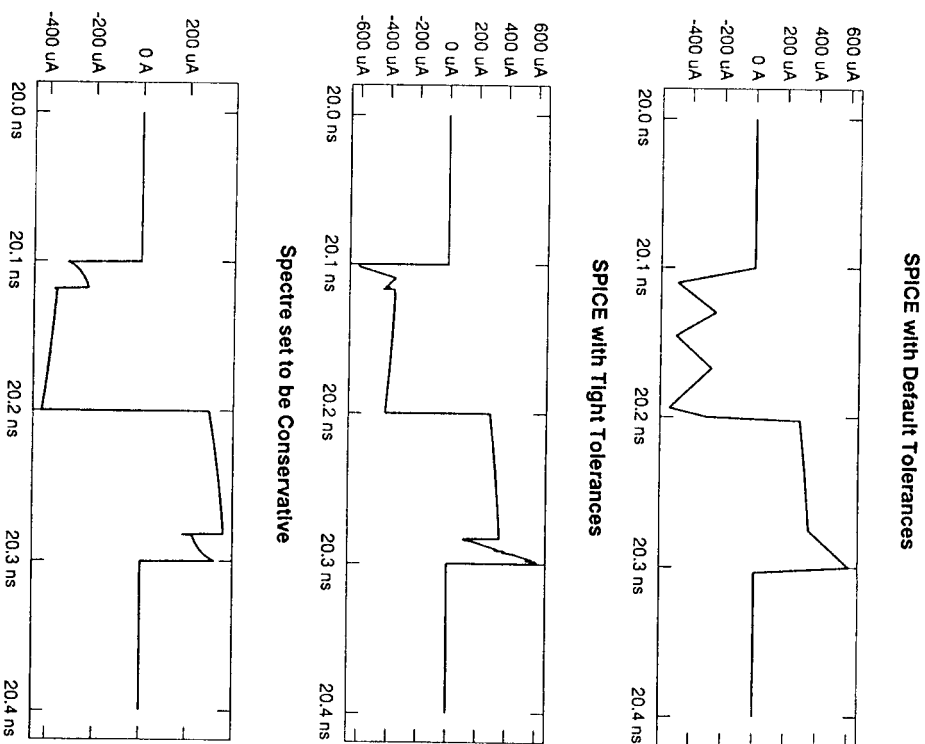


Figure 4.62: Supply current in a 2-input Nand gate was computed by SPICE with default tolerances, SPICE with tight tolerances (`reltol = 10-6`). Finally, as computed by Spectre with `errpreset=conservative` and `step=1ps`. The results computed by Spectre are the most accurate. The odd notches in the waveform results from jumps in the gate capacitance as the FET changes regions.

Generally with such circuits, the long term or low frequency response is more interesting than in the high frequency response, usually because the high frequency signals are not desired and is removed eventually by filters. However, in the mean time, the high frequency signals can obscure the information in the low frequency response. In addition, the large number of time points needed to accurately represent the high frequency information can make plotting or printing the signals a time consuming or cumbersome affair.

Spectre provides an option for its transient analysis called strobing that is completely analogous to a strobe light. Strobe lights are invaluable when trying to observe the motion of rapidly rotating machines. They emit a short powerful burst of light at exactly the same rate at which the machinery rotates. Since the flash occurs at exactly the same point in every cycle of rotation, the machinery *appears* to stop rotating and allows non-ideal behavior such as vibrations or flexings to be easily seen. In a circuit simulator context, strobing or sampling is used to demodulate signals. Sampling at a given frequency eliminates that frequency from the output.

Spectre implements strobing by computing the entire response of the network as usual, but only outputting the results once per cycle or period. Spectre places a time point at the sample time, so interpolation is not used as it is with SPICE's print and plot interval. In addition, the phase of the sample can be specified.

Strobing has several useful characteristics:

1. It completely removes any frequencies at the sample rate. In other words, if sampled at the frequency of the repetitive signal, then that frequency is removed from the output.
2. It hides uninteresting detail in the waveform and reveals patterns.
3. It reveals small nonideal behavior that would be obscured by the high frequency signal.
4. It greatly reduces the number of points in the waveform making it much easier to store and display.

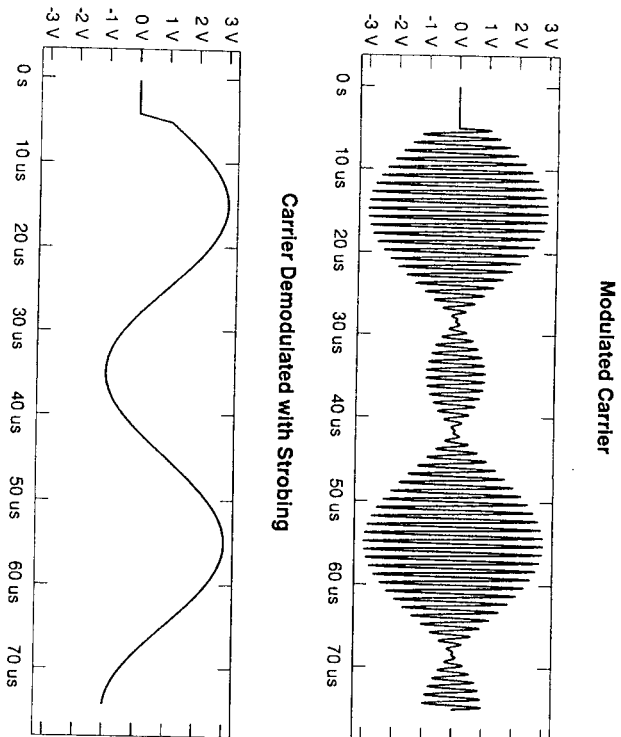


Figure 4.63: Amplitude modulated carrier that is demodulated using strobing.

5. When strobed waveforms are used in *XY* plots (with time as hidden parameter), the constant time between points can be used to make timing estimates.
6. When used with a DFT or FFT, it eliminates interpolation error and greatly improves the accuracy of the Fourier analysis.

The ability of strobing to demodulate AM signals is shown in Figure 4.63. This signal is a 1 MHz carrier that is amplitude modulated at 25 kHz. The circuit was strobed at 1 MHz with a delay of 250 ns to assure that the samples were taken at the peak of the carrier. All trace of the 1 MHz is eliminated from the demodulated waveform by strobing.

Strobing can also be used to crudely demodulate FM signals. Consider the waveform shown in Figure 4.64 on the next page. It is a 100 MHz constant amplitude carrier frequency modulated at 100 kHz. If the waveform is strobed (or sampled) at a 100 MHz rate, the 100 kHz signal is clearly seen. However, it is distorted. The reason for the distortion is seen by writing the equation for an FM signal modulated by a sinusoid.

$$v(t) = \sin(\omega_C t + A \cos(\omega_M t)) \quad (4.62)$$

When sampled at the carrier frequency, the $\omega_C t$ drops out and the strobed signal becomes

$$v(2\pi n/\omega_C) = \sin(A \cos(\omega_M 2\pi n/\omega_C)). \quad (4.63)$$

The modulation signal ($A \cos(\omega_M t)$) suffers a sinusoidal distortion. Besides this distortion, FM demodulation using strobing also suffers because strobing is sensitive to AM modulation or amplitude variations in the FM modulated signals.

An example of a circuit for which it is useful to strobe at two different rates is now considered. Figure 4.65 on page 244 shows the block diagram of a 16 state quadrature-amplitude modulator (16-QAM) and communications channel. At the output of the demodulator, the 16-QAM signal is ideal. The channel adds dispersion and delay. Figure 4.66 on page 244 shows the waveforms at the output of the modulator and channel. The two waveforms show the signals on either side of the channel in continuous time. Little information can be extracted from this presentation.

Figure 4.67 on page 245 shows the constellation diagrams that were generated by strobing the waveforms at the carrier frequency and plotting the output of the demodulators. The quadrature (or imaginary) output Q is plotted versus the in-phase (or real) output I . This presentation conveys more information on how the output is affected by the channel and what the maximum data rate is. The bottom pair of waveforms shows the same constellation diagrams, except the waveforms are strobed at the data rate rather than the carrier frequency. The timing information, or the information on how long it takes to get from one symbol to another, is lost because only the final settled value is shown. However, the intersymbol interference created

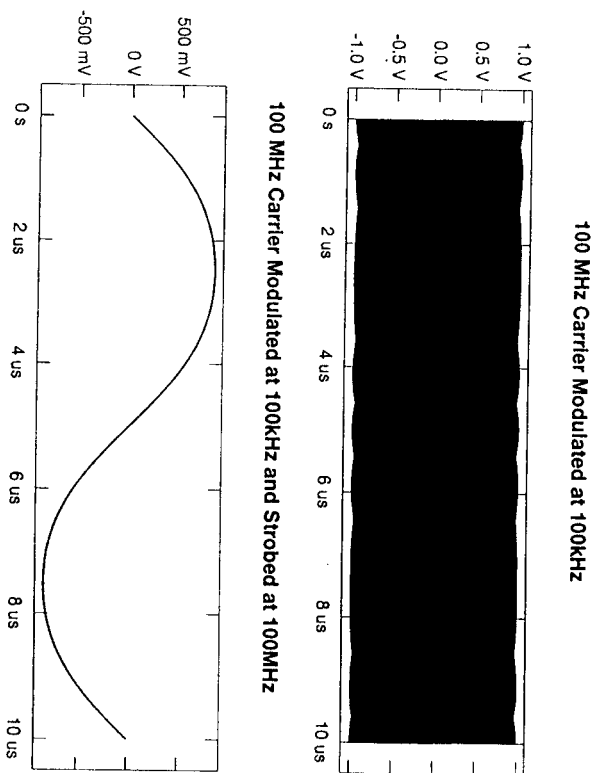


Figure 4.64: Frequency modulated carrier that is crudely demodulated using strobing. The amplitude variations seen in the top waveform are not real, they result from the simulator not placing time points at the precise top of every undulation. The top waveform is strobed at a 100 MHz rate to produce the bottom waveform, which is a distorted sinusoid. The distortion results because the voltage at the sample point is nonlinear function of the phase of the signal.

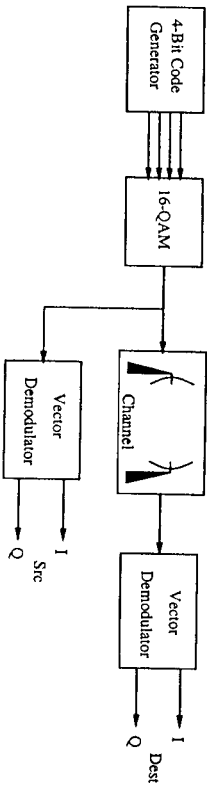


Figure 4.65: Block diagram of a test circuit for a 16-QAM modulator. The vector detectors are the sampling detectors shown in Netlist 5.6 on page 325.

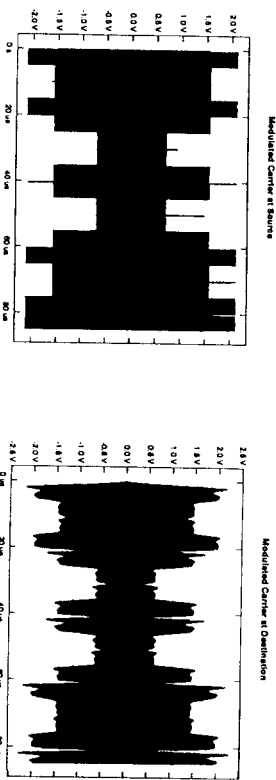


Figure 4.66: Response of 16-QAM modulator at the source (on the left) and at the destination (on the right) to a sequence of 16 symbols.

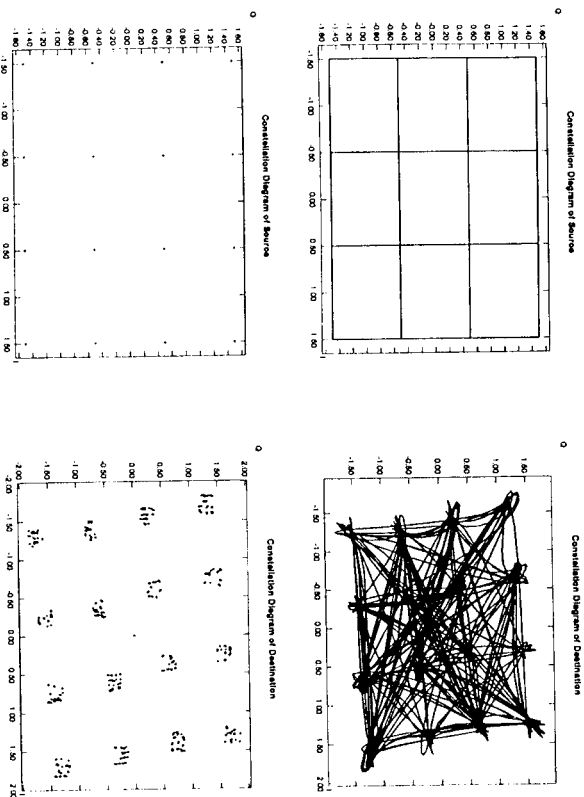


Figure 4.67: Constellation diagrams for the 16-QAM modulator at the source (on the left) and at the destination (on the right). The top figures are Q-vs-I sampled at the carrier frequency, and so include the transitions between symbols. The bottom figures are Q-vs-I sampled at the modulation clock rate, which eliminates the transition.

by the dispersion of the channel is clearly visible. It is the reason why the samples form clouds around the desired symbol value. The radius of the cloud is a measure of the expected bit error rate.

For another example of when strobing or sampling is desirable, see Section 5.4.6 on page 313 on clocked analog circuits in the Fourier analysis chapter.

4.5 Summary

This is a list of the key points presented in this chapter along with the section and page numbers where they were presented.

- For a fixed time step, the most accurate second-order method in a local sense is the trapezoidal rule. However, circuit simulators automatically control the time step to maintain an acceptable level of error, so it is not possible to say in advance which method is more accurate for a particular circuit, but the trapezoidal rule normally allows larger time steps and so requires a shorter run time. Section 4.2.2 on page 133.
- All stable differential equations, and some unstable ones, are stable when simulated with backward Euler. Section 4.2.2.2 on page 141.
- Stable differential equations are stable and unstable differential equations are unstable when simulated with trapezoidal rule. Section 4.2.2.2 on page 141.
- All stable differential equations, and some unstable ones, are stable when simulated with Gear2. Section 4.2.2.2 on page 144.
- Some integration methods exhibit an artificial numerical damping which manifests itself by artificially increasing the damping in the circuit. The trapezoidal rule does not exhibit numerical damping. However, Gear's second-order backward difference formula does exhibit some damping and backward Euler exhibits heavy damping. Thus, backward Euler, and to a lesser degree, Gear2, appear to increase the loss, or decrease the Q , of

low-loss resonators. This is important when trying to determine the amount of noise or ringing on an inductive clock or power supply net. Section 4.2.2.3 on page 147.

- On stiff circuits, trapezoidal rule generates solutions where the sign of the error alternates on every time step. The solution is within specified limits, but exhibits a point-to-point ringing, often referred to as *trapezoidal ringing*. Section 4.2.2.4 on page 150.
- Currents through capacitors and voltages across inductors are generally not computed as accurately as voltage across capacitors and current through inductors. In addition, some artifacts of the numeric integration algorithm are more extreme when computing capacitor currents and inductor voltages. Section 4.2.2.5 on page 152.
- The simulator does not choose the time step to control errors on voltage, except those across capacitors, and or currents, except those through inductors. Unlike the voltage across a capacitor or current through an inductor, errors in other signals do not accumulate, and therefore they are ignored by the time-step control algorithm. Therefore, these signals can look rough when plotted. In particular, current through capacitors and voltage across inductors are the derivatives of controlled variables, which magnifies error. Section 4.2.2.5 on page 153.
- Analog circuits, particularly oscillators and integrators are very sensitive to truncation error. These circuits exhibit very long time constants that retain the effect of any perturbations for a very long time. Section 4.2.4.1 on page 161.
- There is no truncation error when solution waveforms are constant valued. Thus, truncation errors affect time constants, such as settling times and periods of oscillation, but do not affect DC values or settled values unless they depend on time constants. Section 4.2.4.2 on page 166.
- SPICE chooses the time step to control error of capacitor charge. Spectre chooses the time step to control the error of node voltages. For linear capacitors, the two approaches are similar ex-

cept that in SPICE, `chgtol` determines the minimum interesting signal level, whereas in Spectre it is determined by `vabstol`. Section 4.3.2 on page 182.

- There are several mechanisms that affect charge conservation. The most important is that nature of semiconductor models (capacitance-based versus charge-based). However, even when charge-conserving models are exclusively used, there can still be some problems with charge conservation. Section 4.2.5 on page 167.
- Capacitance-based models do not result in charge conservation problems when the capacitor is linear and the capacitance is exact. Section 4.2.5 on page 170.
- If `uic` is specified on the transient analysis line, SPICE never computes the solution at $t = 0$, it is simply inferred from the initial conditions. This approach is inconvenient because it forces all inductors and capacitors to have initial conditions. If initial conditions were not given for particular inductors or capacitors, and if they cannot be explicitly derived from initial conditions placed on nodes using the `.ic` statement, they are assumed to be zero. Section 4.3.5.1 on page 196.
- Nodesets and non-`uic` initial conditions are implemented in a similar manner but their effect is quite different. Initial conditions actually define the initial solution, whereas nodesets only influence it. Also, while nodesets affect both the DC and transient analyses, initial conditions only affect the transient analysis. The transient waveforms start from initial conditions. Nodesets are usually used only as a convergence aids and do not affect the final results (unless the circuit has more than one solution, such as a latch, in which case nodesets bias the simulator towards finding the solution closest to the nodeset values). Section 4.3.5.3 on page 199.
- Transient analysis convergence problems are often caused by jumps in the solution. This most often occurs in circuits with overly simplified models that exhibit positive feedback, such as relaxation oscillators or Schmitt triggers. Jumps can occur when the circuit contains nodes that do not have a capacitive

path to ground. In practice, waveforms generated by real circuits cannot jump discontinuously, and so the circuit must be incompletely modeled.

Generally, a small capacitor from the troublesome node to ground solves the problem. Spectre provides the transient parameter `cm1n`, which when nonzero connects a small capacitor from every node to ground. This slows down the transitions enough so that Spectre can follow them without convergence problems. Section 4.3.7.1 on page 202.

- Oscillators generally do not start oscillating by themselves. Oscillators are started by specifying initial conditions or by inserting a short duration pulsed source. Choose a method that couples strongly into the mode of oscillation and weakly into other undesirable modes, such as long time constants associated with the bias circuitry. Section 4.4.2 on page 210.
- Oscillators are more sensitive to errors than many other circuits and simulators tend to underestimate the frequency of oscillation. Section 4.4.2 on page 218.
- When simulating inductive power of clock distribution networks where ringing is a concern, use of Gear's methods or especially backward Euler should be avoided because they provide additional damping to the circuit. Use trapezoidal rule instead. Section 4.4.7 on page 234.
- Suggested remedies for convergence problems that occur while computing the initial time point or the initial conditions are given in Section 2.3.1 on page 35. Suggested remedies for accuracy problems that occur while computing the initial time point or the initial conditions are given in Section 2.3.3 on page 41. Suggested remedies for convergence problems that occur during the transient analysis are given in Section 4.3.8 on page 206. Suggested remedies for accuracy problems that occur during the transient analysis are given in Section 4.3.6 on page 199.

Chapter 5

Fourier Analysis

5.1 Introduction

The theory, practice, and application of Fourier analysis are presented in the three major sections of this chapter. The theory includes a discussion of Fourier transforms, including the Fourier integral and the discrete Fourier transform, along with suggestions on computing the spectrum of quasiperiodic waveforms. The section on practice presents details on error mechanisms as well as on how Fourier analysis is implemented in SPICE, Spectre, and external Fourier analyzers. Suggestions are given on how to achieve accurate results. The chapter concludes by presenting various types of circuits that present challenges to Fourier analysis and how those challenges might be addressed.

5.1.1 Distortion Metrics

Fourier analysis is almost exclusively used to measure distortion. When measuring distortion, there are two important cases to consider: wide-band and narrow-band circuits.

5.1.1.1 Harmonic Distortion

To measure the distortion of wide-band circuits, one simply applies a pure single-tone sine wave to the input, and determines how much

the output deviates from being a pure sinusoid after the circuit has reached steady state. The distortion products in the output signal fall at frequencies that are integer multiples (harmonics) of the input frequency. In this case, the distortion measure of choice is *total harmonic distortion* or *THD*.

Total Harmonic Distortion

THD is a measure of the amount of energy in the harmonics relative to the energy in the fundamental.

$$\text{THD} = \sqrt{\frac{\sum_{k=2}^{\infty} |X_k|^2}{|X_1|^2}} \cdot 100\% \quad (5.1)$$

where $|X_k|$ is the magnitude of the k^{th} harmonic.

5.1.1.2 Intermodulation Distortion

If this same technique were applied to measure the distortion of a narrow-band amplifier, the harmonic distortion products would normally fall outside the bandwidth of the amplifier and so would be attenuated. The amount of distortion calculated using (5.1) would be much too low.

Distortion is measured in narrow-band amplifiers by applying two pure sinusoids with frequencies well within the bandwidth of the amplifier (call these frequencies f_1 and f_2). The harmonics of these two frequencies are outside the bandwidth of the amplifier, however there are distortion products that fall at the frequencies $2f_1 - f_2$, $2f_2 - f_1$, $3f_1 - 2f_2$, $3f_2 - 2f_1$, etc. As shown in Figure 5.1 on the next page these frequencies should also be well within the bandwidth of the amplifier and so can be used to measure accurately what is referred to as the *intermodulation distortion*, or *IMD*, produced by the amplifier.

5.1.1.3 Compression and Intercept Points

Harmonic and intermodulation distortion both characterize signals rather than the underlying circuit. At low frequencies, it is common

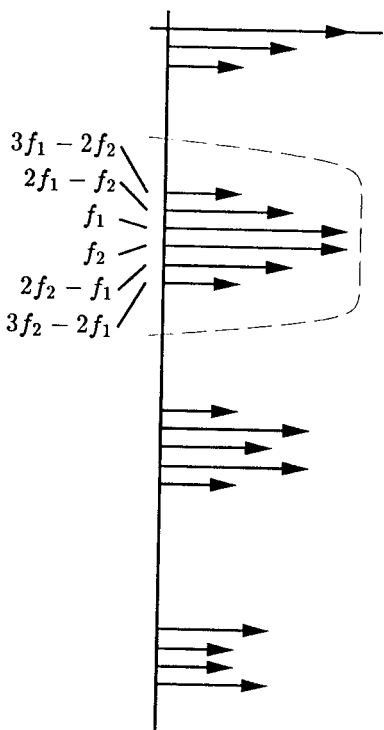


Figure 5.1: Two-tone intermodulation products superimposed on a narrow-band filter's passband.

to describe the distortion of a circuit by indicating the distortion in the output signal when driven by a sinusoid to achieve a certain output level. For example, an audio power amplifier might be described as having 0.1% THD at 100 Watts. At high frequencies it is more common to characterize the distortion produced by a circuit in terms of a compression point or an intercept point. These metrics characterize the circuit rather than the signal, and as such it is not necessary to specify the signal level at which the circuit was characterized.

In order to more easily understand the definition of a compression point and an intercept point, consider the output power of the first three harmonics for a GaAs traveling-wave amplifier, as shown in Figure 5.2 on the following page. The 1 dB compression point is the value of the output power when the gain of the amplifier has dropped 1 dB from its small-signal asymptotic value, as shown in Figure 5.3 on page 255.

The compression point is a difficult measurement to make with a time-domain circuit simulator (SPICE) because one must measure the

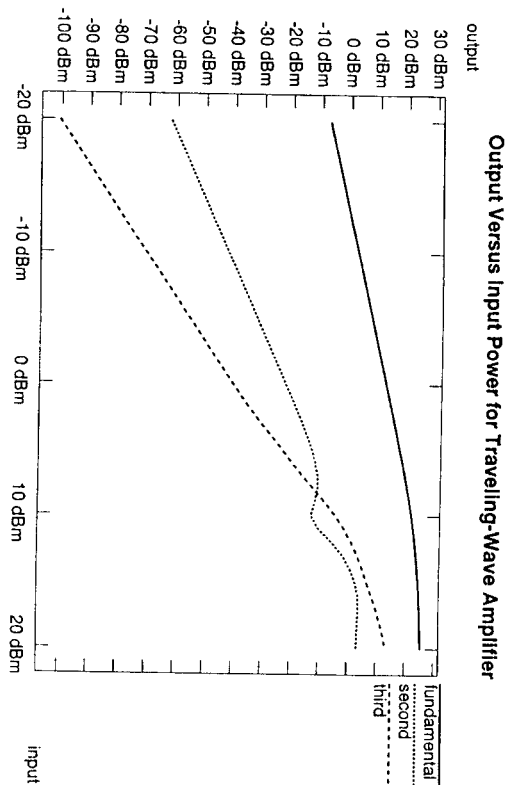


Figure 5.2: The power in the first three harmonics at the output of the travelling-wave amplifier versus the input power.

power in the fundamental of the output (using transient and Fourier analysis) while sweeping the input power. This generally requires many simulation runs. The actual compression point is found by drawing a figure similar to Figure 5.3 and determining the proper value graphically.

The intercept points are defined in terms of the power levels of the various harmonics as extrapolated from their asymptotic small-signal behavior. When the input signal is small, a doubling of the input power doubles the output power at the fundamental, multiplies the output power of the second harmonic by 4 (2^2), and multiplies the output power of the third harmonic by 8 (2^3). Thus, the asymptotic slope of the fundamental is 1 dB/dB, the asymptotic slope of the second harmonic is 2 dB/dB, and the asymptotic slope of the third harmonic is 3 dB/dB. The second-order intercept point (IP_2)

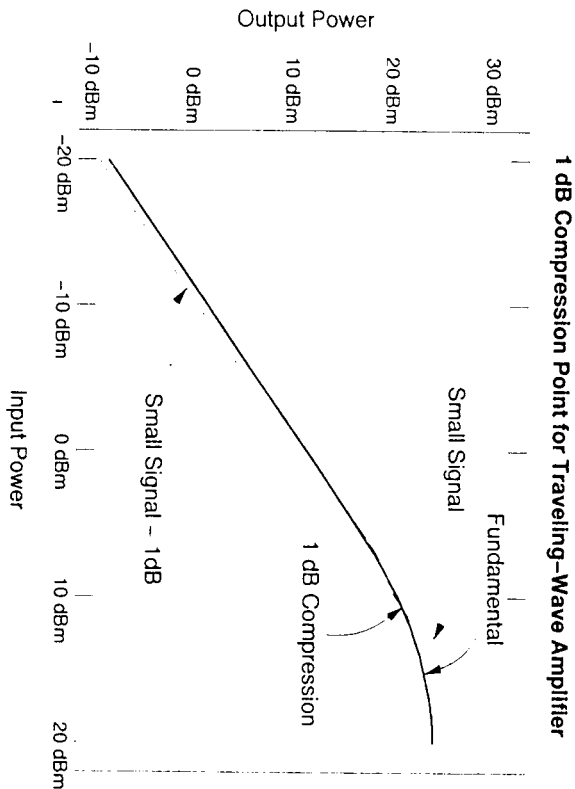


Figure 5.3: The 1 dB compression point is the value of the fundamental output power where it crosses the line that represents the output power extrapolated from small signal conditions minus 1 dB. In this example, the compression point is 22.3 dBm.

is where the asymptotes for the second harmonic and the fundamental cross. Similarly, the third-order intercept point (IP_3) is where the asymptotes for the third harmonic and the fundamental cross. The definitions of IP_2 and IP_3 are illustrated in Figure 5.4 on the following page.

The intercept point is easier to measure with a time-domain circuit simulator than is the compression point because it requires just one transient and one Fourier analysis. Simply choose an input power level that you are confident puts the circuit in the 'small-signal' regime and then measure level of the first, second, and third har-

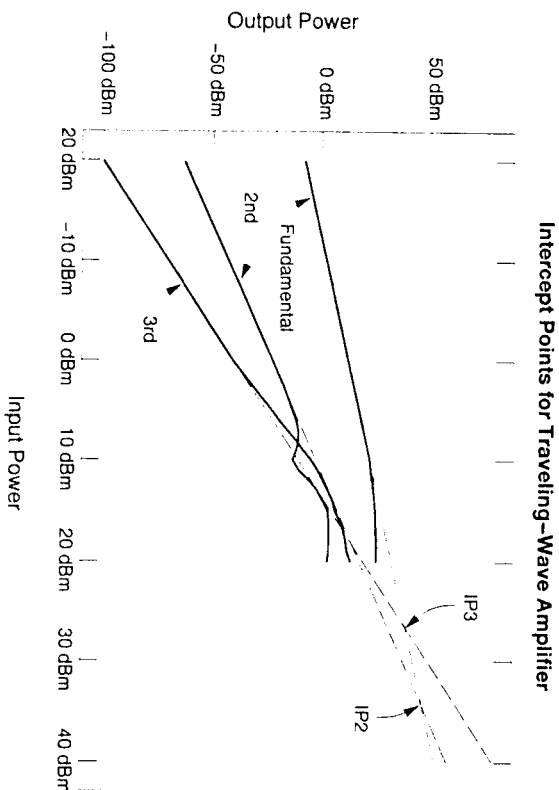


Figure 5.4: The second-order intercept point is the power where the second-order term as extrapolated from small-signal conditions crosses the extrapolated power of the fundamental. Similarly, the third-order intercept point is the power where the third-order term as extrapolated from small-signal conditions crosses the extrapolated power of the fundamental. In this example, IP_2 is 47.6 dBm and IP_3 is 39.4 dBm.

monics. Let P_1 be the power of the input test signal and P_1 , P_2 , and P_3 be the power in the first, second, and third harmonics at the output. Assume that the power is measured in your favorite flavor of decibels (dBm, dBW, dBV, etc). Then,

$$IP_2 = 2P_1 - P_2 \quad (5.2)$$

$$IP_3 = (3P_1 - P_3)/2 \quad (5.3)$$

These calculations are only accurate if P_i is small enough so that the output power levels are in the asymptotic region for all harmonics, and if P_i is large enough so that the Fourier analysis accurately computes P_1 , P_2 , and P_3 .

In the intercept point definitions just given, harmonic distortion was used for illustrative purposes only. The same definitions are just as easily be applied to intermodulation distortion. In this case, the second-order distortion terms play the role of the second harmonic, and the third-order distortion terms play the role of the third harmonic.

5.2 Fourier Analysis Theory

It is a fundamental tenet in the theory of communications systems that waveforms can be decomposed into a possibly infinite sum of sinusoids or complex exponentials. In a circuit simulator, Fourier analysis is what is used to perform the decomposition. Fourier analysis takes a waveform, which is either a continuous or discrete real function of time, and transforms it into a spectrum, which is either a continuous or discrete complex function of frequency. The value of the spectrum at frequency f is the coefficient of the complex exponential of the same frequency.

5.2.1 The Fourier Series

5.2.1.1 Periodic Waveforms

Periodic waveforms are generated by circuits in steady-state undergoing autonomous oscillation or responding to periodic stimuli (assuming that all stimulus have the same period T). The frequency of

the response is usually equal to the frequency of the stimulus, though occasionally the frequencies of the two are multiples of some common frequency.

If a waveform is T -periodic and continuous, then the waveform can be written as a Fourier series.

$$x(t) = \sum_{k=0}^{\infty} \left[a_k \cos\left(\frac{2\pi kt}{T}\right) + b_k \sin\left(\frac{2\pi kt}{T}\right) \right] \quad (5.4)$$

A waveform $x(t)$ is T -periodic if $x(t) = x(t - T)$ for all t . The sequence of Fourier coefficients (a_k and b_k) is called the spectrum of x . Periodic waveforms consist of a sum sinusoids at integer multiples of the fundamental frequency $f = 1/T$. The signals at frequencies above the fundamental are called harmonics. Thus, the k^{th} harmonic is the signal at kf .

Given a T -periodic waveform $x(t)$, you compute the Fourier coefficients by evaluating the following Fourier integrals.

$$a_0 = \frac{1}{T} \int_t^{t+T} x(\tau) d\tau \quad (5.5)$$

$$a_k = \frac{2}{T} \int_t^{t+T} x(\tau) \cos\left(\frac{2\pi k\tau}{T}\right) d\tau \quad (5.6)$$

$$b_k = \frac{2}{T} \int_t^{t+T} x(\tau) \sin\left(\frac{2\pi k\tau}{T}\right) d\tau \quad (5.7)$$

The Fourier Integral as a Bank of Receivers In order to understand how the Fourier integral works, it is useful to view it as a bank of receivers, one for each coefficient that is to be computed, as shown in Figure 5.5 on the next page. Each receiver translates the frequencies of the input signal so that the frequency of interest is moved to DC, and then filters the results in order to eliminate the signals at other frequencies. A comb filter is used that has zeros at the frequency of the fundamental and every harmonic. The comb filter completely eliminates all frequencies that exist in the signal assuming that it is precisely T -periodic except the one at DC. For

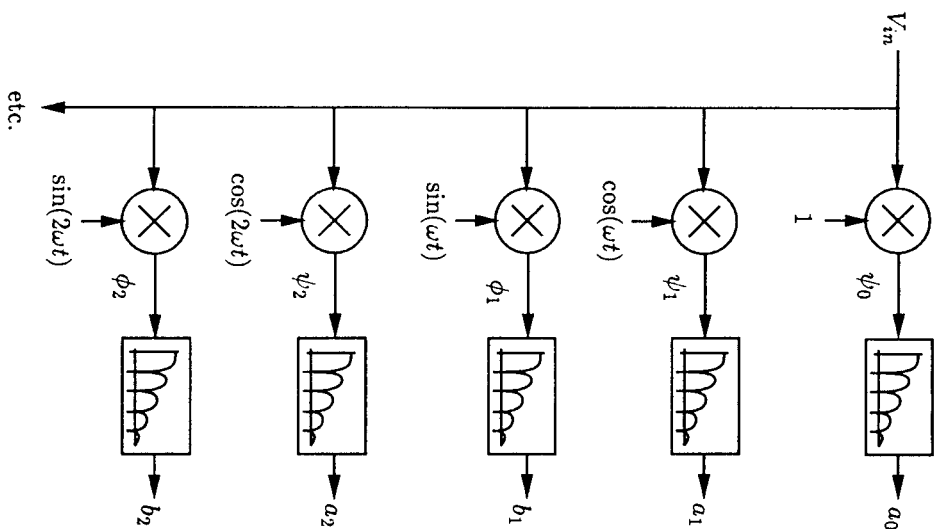


Figure 5.5: The Fourier integrals of (5.5–5.7) interpreted as a bank of receivers.

example, consider computing the coefficient a_k for the signal $x(t)$ of (5.4). To do so, apply (5.6).

$$a_k = \frac{2}{T} \int_t^{t+T} x(\tau) \cos(k\omega\tau) d\tau \quad (5.8)$$

where $\omega = 2\pi/T$. Partition the calculation into two pieces, the frequency translation and the filtering.

$$a_k = \int_t^{t+T} \psi_k(\tau) d\tau \quad (5.9)$$

where

$$\psi_k(t) = \frac{2}{T} \cos(k\omega t) x(t) \quad (5.10)$$

The filter is implemented by the integral of (5.9) and the frequency translation is implemented in (5.10). Now, replace $x(t)$ with its Fourier series.

$$x(t) = \sum_{\ell=0}^{\infty} [a_{\ell} \cos(\ell\omega t) + b_{\ell} \sin(\ell\omega t)] \quad (5.11)$$

$$\psi_k(t) = \frac{2}{T} \cos(k\omega t) \sum_{\ell=0}^{\infty} [a_{\ell} \cos(\ell\omega t) + b_{\ell} \sin(\ell\omega t)] \quad (5.12)$$

The multiplication of two sinusoids is the operation that performs the frequency translation, as can be seen with the following two trigonometric identities.

$$2 \cos \alpha \cos \beta = \cos(\alpha - \beta) + \cos(\alpha + \beta) \quad (5.13)$$

$$2 \sin \alpha \cos \beta = \sin(\alpha - \beta) + \sin(\alpha + \beta) \quad (5.14)$$

Multiplying $x(t)$ by $(2/T) \cos(k\omega t)$ effectively duplicates the signal and translates one copy up by $k\omega$ and translates the other copy down by $k\omega$.

$$\begin{aligned} \psi_k(t) &= \frac{1}{T} \sum_{\ell=0}^{\infty} [a_{\ell} \cos((\ell - k)\omega t) + b_{\ell} \sin((\ell - k)\omega t)] \\ &+ \frac{1}{T} \sum_{\ell=0}^{\infty} [a_{\ell} \cos((\ell + k)\omega t) + b_{\ell} \sin((\ell + k)\omega t)] \end{aligned}$$

Thus, to compute a_k , the Fourier integral simply multiplies $x(t)$ by $(2/T) \cos(k\omega t)$, which translates $x(t)$ both up and down in frequency by $k\omega$, and then low-pass filters the product to eliminate all but the DC term, which precisely equals a_k . To see this, separate out the term where $\ell = k$ (and for simplicity, assume $k \neq 0$).

$$\begin{aligned} \psi_k(t) &= \frac{a_k}{T} \cos((k - k)\omega t) \\ &+ \frac{b_k}{T} \sin((k - k)\omega t) \\ &+ \frac{1}{T} \sum_{\ell=0, \ell \neq k}^{\infty} [a_{\ell} \cos((\ell - k)\omega t) + b_{\ell} \sin((\ell - k)\omega t)] \\ &+ \frac{1}{T} \sum_{\ell=0}^{\infty} [a_{\ell} \cos((\ell + k)\omega t) + b_{\ell} \sin((\ell + k)\omega t)] \end{aligned}$$

$$\begin{aligned} \psi_k(t) &= \frac{a_k}{T} \\ &+ \frac{1}{T} \sum_{\ell=0, \ell \neq k}^{\infty} [a_{\ell} \cos((\ell - k)\omega t) + b_{\ell} \sin((\ell - k)\omega t)] \\ &+ \frac{1}{T} \sum_{\ell=0}^{\infty} [a_{\ell} \cos((\ell + k)\omega t) + b_{\ell} \sin((\ell + k)\omega t)] \end{aligned}$$

The second and third terms do not contain any DC components and are removed by the filter.

The integral operator of (5.9) acts like a low-pass comb filter. The impulse response of the integral is a unit pulse, and the transform of a unit pulse has the form of $\text{sinc}(f) = (\sin(\pi x))/(\pi x)$ as shown in Figure 5.6 on the following page. The filter has a gain of T at DC, and nulls at each harmonic. Thus, the frequency of interest, which is translated to DC when multiplying it by $(2/T) \cos(k\omega t)$, is passed without attenuation, but the other harmonics are completely eliminated because they fall at the nulls.

Consider the example of using (5.6) to compute a_2 of $x(t) = 2 \cos^2(t)$ where $T = 1/2\pi$ and so $\omega = 1$. Using (5.13), $x(t)$ can be analytically shown to be equal to $1 + \cos(2t)$, which implies that all Fourier coefficients are zero except a_0 and a_2 , which both equal one. First apply

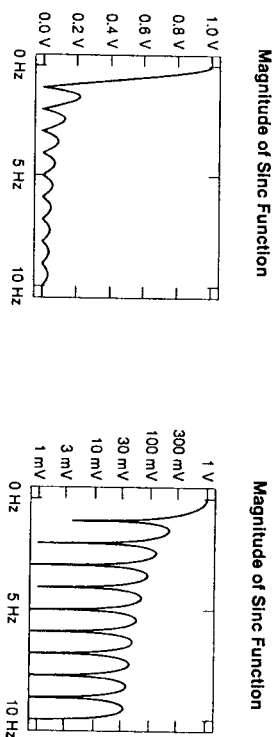


Figure 5.6: The frequency response of the filter used when computing Fourier coefficients using the Fourier integral. The two figures are the same, except that the left one uses a linear grid whereas the right one uses a semilog grid.

(5.10) to compute ψ_2 .

$$\psi_2(t) = 4\pi \cos(2t)(1 + \cos(2t)) \quad (5.15)$$

$$\psi_2(t) = 2\pi + 4\pi \cos(2t) + 2\pi \cos(4t) \quad (5.16)$$

Passing $\psi_2(t)$ through the filter shown in Figure 5.6 and defined by (5.9) results in the second and third terms being eliminated because:

Time-Domain Rationale: Both the second and third terms are cosines that are integrated over exactly an integer number of periods.

Frequency-Domain Rationale: The second and third terms fall exactly at the nulls of filter's transfer function.

These two explanations are simply two different ways of saying the same thing. In this example, the Fourier coefficients are computed exactly.

Now consider the example of using (5.5–5.7) to compute a_k of $x(t) = 2 \cos(1.5t)$ where $T = 1/2\pi$. In this case, the input signal is periodic,

5.2. Fourier Analysis Theory

263

but it is not T -periodic. First apply (5.10) to compute ψ_2 .

$$\psi_k(t) = 4\pi \cos(kt) \cos(1.5t) \quad (5.17)$$

$$\psi_k(t) = 2\pi \cos((1.5 + k)t) + 2\pi \cos((1.5 - k)t) \quad (5.18)$$

Passing $\psi_k(t)$ through the filter shown in Figure 5.6 and defined by (5.9) does not result in either term being eliminated for any value of k . The terms do not fall in the nulls of the filter. This phenomenon is referred to as spectral leakage and can be thought of as a particularly complex form of aliasing. It is discussed later.

5.2.1.2 Quasiperiodic Waveforms

Circuits in steady-state that are driven by several periodic sources at unrelated frequencies respond by generating quasiperiodic signals.

Quasiperiodic

A quasiperiodic signal consists of a linear combination of sinusoids at the sum and difference frequencies of a finite set of non-commensurate fundamental frequencies and their harmonics. The fundamental frequencies are non-commensurate if for every pair of fundamentals, there is no common frequency for which both fundamentals are harmonics.

If the number of fundamental frequencies is d , then the signals are said to be d -fundamental quasiperiodic or simply d -quasiperiodic. Quasiperiodic signals are not periodic if $d > 1$. For example, consider

$$x(t) = \cos t + \cos \sqrt{t}. \quad (5.19)$$

While $x(0) = 2$, for all other values of time $x(t) < 2$. As a result, $x(t)$ cannot be periodic because there is no T such that $x(0) = x(T)$.

A waveform that is d -quasiperiodic can be written as a Fourier series. For simplicity, let $d = 2$,

$$x(t) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} X_{k_1 k_2} e^{2\pi j(k_1 f_1 + k_2 f_2)t} \quad (5.20)$$

where $X_{k_1 k_2}$ is the complex Fourier coefficient. The spectrum of this signal is shown in Figure 5.1 on page 253. A numerical algorithm is available for computing the forward and inverse discrete Fourier transform for quasiperiodic signals. It is called the Almost-Periodic Fourier Transform, or APFT, but it is not very practical because it tends to be ill-conditioned [Kundert90].

Another Bank of Receivers Another approach for computing the Fourier coefficients of quasiperiodic signals for highly motivated individuals is to use the ideas presented above (pg. 258) to build a specialized bank of receivers to compute the Fourier coefficients of a quasiperiodic waveform. Thus, each filter in the bank translates its frequency to DC and then filters out the other frequencies. The Fourier integral assumes the input signals are periodic and uses a comb filter with zeros at each harmonic to precisely eliminate the unwanted frequencies. Fortunately, comb filters have an infinite number of equally-spaced zeros that filter out the infinite number of equally-spaced harmonics, which results in the complete elimination of spectral leakage when the input waveform is T -periodic.

The situation is more complicated for quasiperiodic waveforms because the unwanted frequencies are not equally-spaced, and the set of unwanted frequencies is different for each filter. One cannot hope to design a filter with an infinite number of precisely placed non-evenly-spaced zeros. The best approach is to use a low-pass filter with a large amount of attenuation everywhere except near DC. The filter can include a few strategically placed zeros to eliminate unwanted frequencies that promise to be large. Techniques for designing such filters are described by Zverev [Zverev67]. A technique for designing discrete-time FIR filters is the *Remez exchange algorithm*. FIR filters are particularly desirable because they have an impulse response that is finite in extent, so they settle completely in a finite time (assuming the input signal has settled).

Pseudo-Quasiperiodic Signals For a signal to be quasiperiodic, its fundamental frequencies must not be commensurate. In other words, for every pair of fundamental frequencies, they must not both

be integer multiples of some lower frequency. If they are, the fundamentals are referred to as being quasi-fundamentals, and the largest common frequency is a true fundamental. For example, 1 and π are not commensurate, whereas 7 and 11 are, because they are both integer multiples of 1. Hence, 7 and 11 are quasi-fundamentals and 1 is the true fundamental. If a signal consists of a linear combination of sinusoids at the sum and difference frequencies of d fundamentals, two of which are commensurate, then the signal is $(d - 1)$ -quasiperiodic. Signals that are 1-quasiperiodic are periodic.

A quasiperiodic signal can be approximated arbitrarily closely by a periodic signal by choosing the period sufficiently long. As a result, periodic Fourier analysis techniques can be applied to quasiperiodic signals. However, when the product of the highest fundamental frequency and the Fourier analysis period becomes very large, the analysis becomes very expensive. In this case, it is often more efficient to eliminate the largest fundamental by demodulation. This is true regardless of whether the fundamentals are commensurate. Thus, if a periodic signal is constructed from two or more quasi-fundamentals, or if a quasiperiodic signal is constructed from two or more true fundamentals, and if at least one of the fundamental frequencies is very high relative to the required Fourier analysis interval, then it is impractical to apply Fourier analysis directly and demodulation is preferred. Techniques for demodulating quasiperiodic signals are described in Section 5.4.7 on page 316.

5.2.2 The Discrete Fourier Transform

The Fourier integral computes Fourier coefficients for continuous-time waveforms. The discrete Fourier transform (or DFT) is used for computing the Fourier coefficients of sequences (discrete-time or sampled waveforms). There must be a whole number of sample intervals in the Fourier analysis period, and so $T_s = T/N$ where T_s is the time between samples and N is the number of samples in the Fourier analysis interval. In contrast to the Fourier series for continuous-time waveforms, there are only K sinusoid sequences with unity amplitude that

1. Are T -periodic, or in other words, are harmonics of the fundamental $f = 1/T$.
2. Are distinct at the sample points.

Thus, any T -periodic sinusoidal sequence with frequency larger than Kf is indistinguishable from a T -periodic sequence with a frequency less than Kf . This phenomenon, which results from the sampling (and not the DFT itself), is referred to as aliasing, and is discussed more fully in Section 5.3.1.3 (page 271). As a result, the Fourier series representation of a discrete-time waveform need only contain K frequencies.

$$x_n = \sum_{k=0}^{K-1} \left[a_k \cos\left(\frac{2\pi kn}{N}\right) + b_k \sin\left(\frac{2\pi kn}{N}\right) \right] \quad (5.21)$$

where $N = 2K - 1$ is the number of time points per period T .

The DFT can be derived by either applying the Fourier integral to a periodically sampled waveform where each sample is an impulse, or by simply applying the trapezoidal rule to numerically evaluate the Fourier integral at an evenly spaced set of N points.¹

$$a_0 = \frac{1}{N} \sum_{n=0}^{N-1} x_n \quad (5.22)$$

$$a_k = \frac{2}{N} \sum_{n=0}^{N-1} x_n \cos\left(\frac{2\pi kn}{N}\right) \quad (5.23)$$

$$b_k = \frac{2}{N} \sum_{n=0}^{N-1} x_n \sin\left(\frac{2\pi kn}{N}\right) \quad (5.24)$$

Together these three equations make up the trigonometric form of the Discrete Fourier Transform (DFT). These equations along with (5.21) make up a transform pair for the discrete Fourier series.

¹The trapezoidal rule is used for the DFT because it can be shown, with help from the Euler-Maclaurin formula, that the method gives exceptional accuracy when applied to periodic function over an entire period with equally-spaced sample points.

The fast Fourier transform, or FFT, is the discrete Fourier transform with its calculations rearranged such that it is very efficient at calculating all N Fourier coefficients. In fact, while the DFT requires N^2 multiply operations, the FFT involves only $N \log_2 N$.

5.3 Fourier Analysis Practice

Fourier analysis involves applying the Fourier integrals (5.5–5.7) or the DFT (5.22–5.24) to the signal computed by the simulator in order to determine the Fourier coefficients. Fourier analysis is not without its problems, and it is important to be familiar with them so that you can recognize them as artifacts of the analysis rather than the circuit and avoid them.

5.3.1 Errors Mechanisms

There are five significant sources of error that often corrupt the results computed by a Fourier analysis.

1. Incorrect period.
2. Transients.
3. Aliasing.
4. Interpolation.
5. Simulation noise.

These errors are described in the next few pages.

5.3.1.1 Error Due to Inaccurate Period

Fourier analysis inherently assumes waveforms are T -periodic, where T is the Fourier analysis interval. Waveforms that are not T -periodic are periodically extended by ignoring anything outside the Fourier analysis interval, and replicating the waveform within the interval. If the period of the signal does not match the Fourier analysis interval,

a discontinuity is generated such as the one shown in Figure 5.7 on the facing page. The discontinuity has a very broad spectrum that contaminates the results of the Fourier analysis through a process that is sometimes called spectral leakage. It greatly reduces your ability to resolve small signals as shown in Figure 5.8.

An inaccurate period is generally only a problem if the user is careless or if the signal is being generated by an autonomous circuit such as an oscillator. Before you can compute the Fourier coefficients of a signal generated by an oscillator, you must run at least one transient analysis without Fourier analysis to accurately determine the period. Remember, if the circuit changes in any way, the period of the oscillation may change.

5.3.1.2 Error Due to Transients

For the Fourier analysis to be accurate, the measured signal must be periodic. If the value at the beginning of the period does not equal the value at the end, then the periodic extension is discontinuous, in a manner similar to that shown in Figure 5.7 on the next page. As before, the broad spectrum of the discontinuity reduces the resolution of a Fourier analysis.

In particular, the circuit must have settled into periodic steady-state. The transient analysis must continue long enough to allow any start-up transients to die before the Fourier analysis begins. The Fourier analysis generally begins at $T_f - T$ where T_f is the final (or stop) time of the transient analysis and T is the period of the Fourier analysis. If the longest time constant excited by the initial turn-on of the circuit is τ , then simulating the circuit for an additional τ before starting the Fourier analysis reduces the error due to transients by 10 dB.

Determine whether the waveform being analyzed is really periodic by computing $v(T_f) - v(T_f - T)$ (which may require interpolation). However, when trying to accurately resolve very small Fourier coefficients, it is difficult to compute the nonperiodicity $(v(T_f) - v(T_f - T))$ with enough accuracy to determine whether a small transient is corrupting your results. The Fourier analysis in Spectre automatically computes and prints the nonperiodicity so you can easily identify or

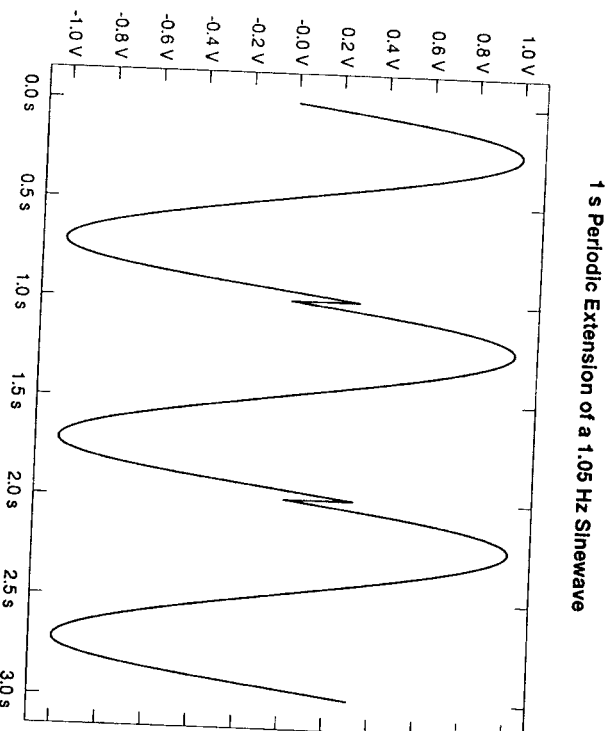


Figure 5.7: The 1 second periodic extension of a 1.05 Hz sinewave. A large difference between the period of the sinewave and the period of the extension was used to make the discontinuity visible in a time-domain waveform. The spectrum of a sinewave with a much smaller difference is shown in Figure 5.8 on the following page.

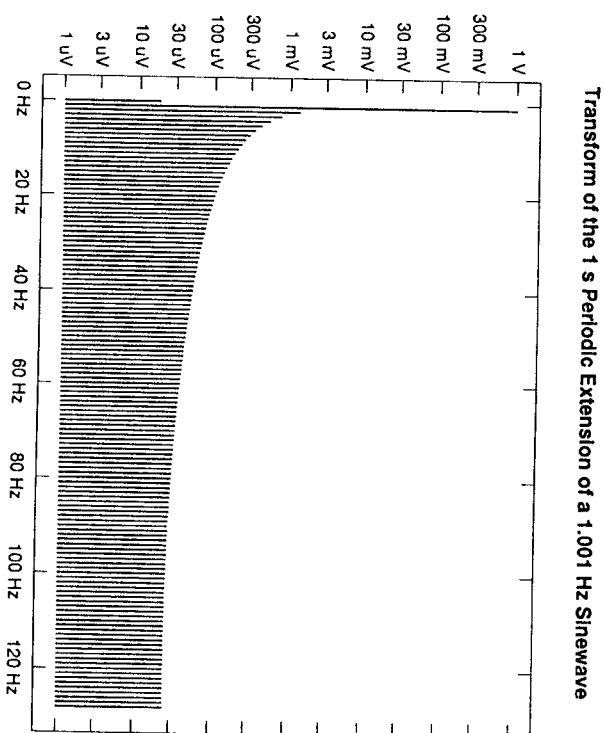


Figure 5.8: The spectrum of the 1 second periodic extension of 1.001 Hz sinewave. This illustrates how a discontinuity that is so small as to be invisible in the time-domain can severely contaminate a spectrum. Notice how slowly the spectrum drops off as frequency increases. Such discontinuities routinely result when Fourier analysis is applied to waveforms with periods different from the Fourier analysis interval or waveforms generated by circuits that have not completely settled.

Ninth Harmonic Aliasing as Seventh

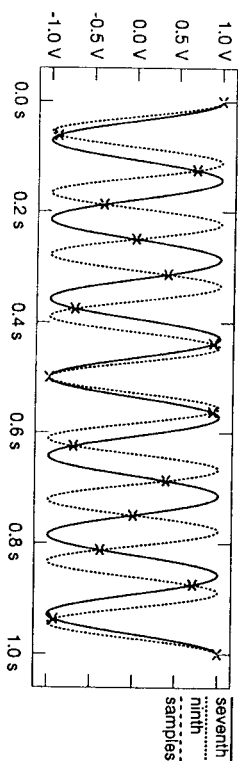


Figure 5.9: An example of aliasing. The seventh and ninth harmonics are sampled at a rate of 16 per period of the fundamental. Notice that the values of both waveforms are identical at the sample points.

rule-out transients as a source of error.

5.3.1.3 Error Due to Aliasing

The discrete Fourier transform is subject to an error called aliasing. The DFT only computes the Fourier coefficients at a finite number of frequencies. Aliasing occurs when energy in the Fourier coefficients not computed masquerades as energy at the harmonics being computed. Aliasing results because the DFT samples the waveform and computes the Fourier coefficients using the value of the waveform only at the sample points and does not use the entire signal. The problem is that once the signal has been sampled, it can represent one of many different signals. For example, assuming that 16 points are used with the DFT, Figure 5.9 shows signals at both the 7th and the 9th harmonic being sampled. Notice that for both waveforms, the values at the sample points are identical, making the 9th harmonic an alias of the 7th. Given a sampled waveform, the DFT assumes that the signal only contains energy at the harmonics of the fundamental

and at those harmonics below half the sample frequency. Energy at any other frequency masquerades as excess energy in one or more of the computed Fourier coefficients.

To avoid aliasing, the signal must be sampled at a rate faster than the Nyquist frequency. The Nyquist frequency is twice the highest frequency present in the signal being sampled. If the sample rate is less than the Nyquist frequency, as shown in Figure 5.10 on the next page, then the spectra overlap and aliasing occurs.

Aliasing is not generally a problem if the signal is nearly sinusoidal, because the energy at these high harmonics are negligible. However, if one wishes to accurately compute the low order harmonics of wide-band signals, such as those with abrupt discontinuities, using the DFT with a small number of points (< 100) gives incorrect answers and is inappropriate.

Fourier analysis based on Fourier integrals rather than the DFT accurately computes the Fourier coefficients for wide-band signals because it is not subject to aliasing. Furthermore, unlike with the fast Fourier transform (the FFT is an efficient implementation of the DFT), it is not necessary to compute many high order harmonics in order to accurately compute the low order harmonics. You only need to compute the harmonics that are of interest.

5.3.1.4 Error Due to Interpolation

The DFT requires equally spaced data points. Circuit simulators naturally generate unequally spaced points. SPICE, and all external Fourier analyzers based on the DFT or FFT, interpolate the waveforms computed by the transient analysis to equally spaced points. If the points computed by the transient analysis are widely separated, the interpolated points may contain significant errors, which would contaminate the results of the Fourier analysis. This process is illustrated in Figures 5.11 and 5.12. Even though the interpolated error is normally small, the Fourier analysis is generally called upon to resolve very small signals, and so it can significantly limit the resolution of the results. For this reason, you should tighten the simulator's tolerances and time step. It is a good idea to set the

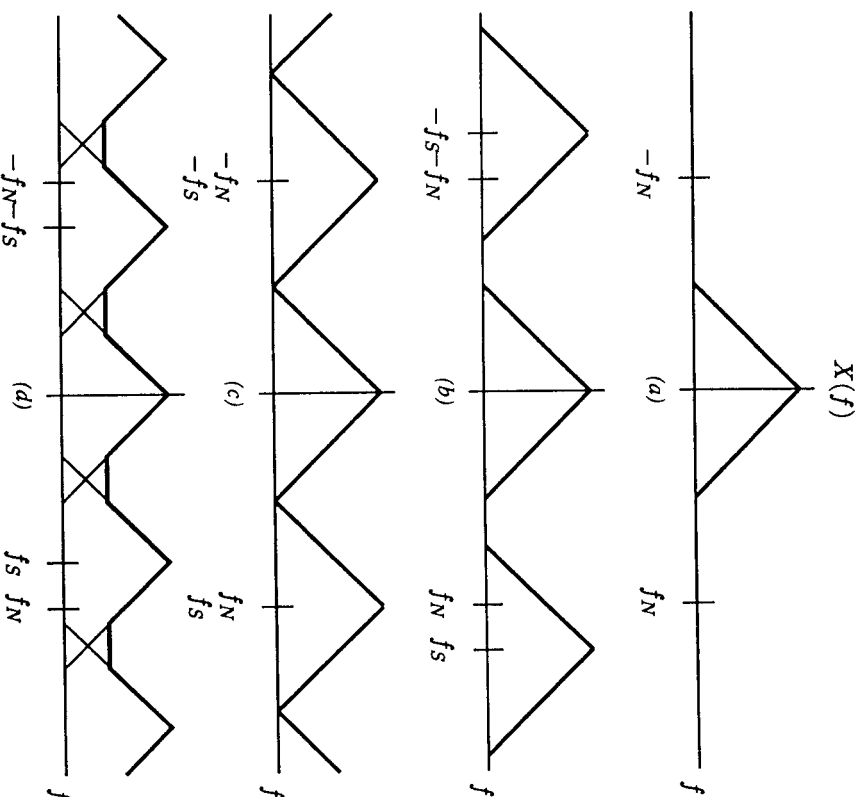


Figure 5.10: The spectrum of a signal before sampling is shown in (a). Sampling causes the spectrum to be replicated at the sampling frequency. There is no aliasing if the sampling frequency is greater (b) or equal (c) to the Nyquist frequency. When the sampling rate is less than the Nyquist frequency (d) the spectra overlap or alias. The Nyquist frequency is denoted f_N and the sampling rate is f_s .

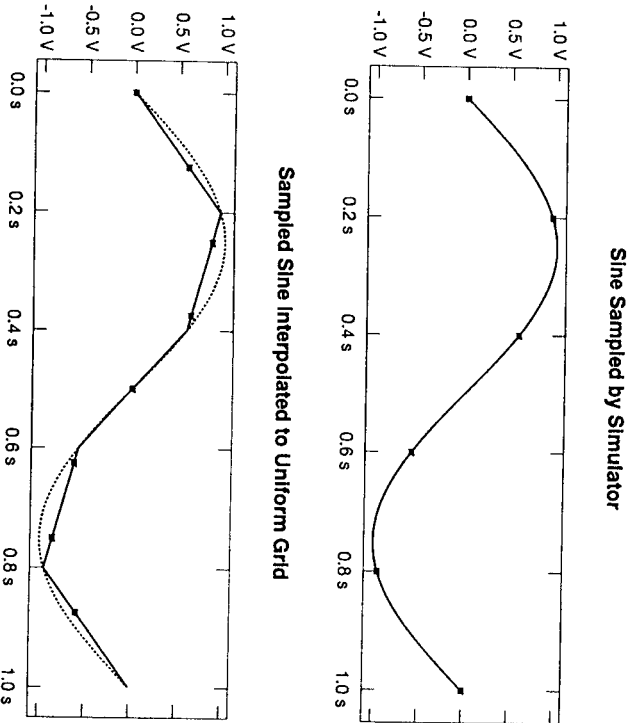


Figure 5.11: The top waveform is sampled directly at 6 equally spaced points. To illustrate the errors caused by interpolating, the 6 equally spaced sample points are linearly interpolated and sampled a 8 equally spaced points, as shown in the top waveform. The 8 sample points do not actually fall on the sine wave and so are in error.

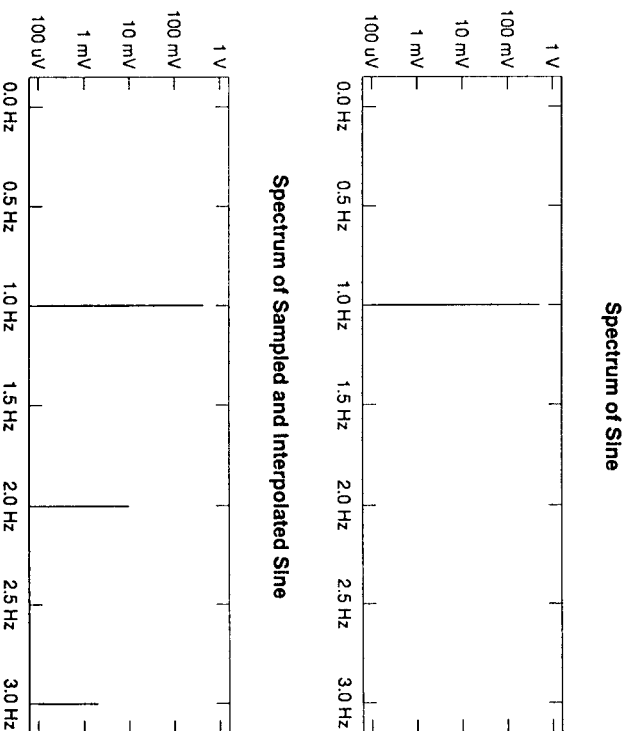


Figure 5.12: The spectrum of the 6 point sampled waveform from the previous figure is shown at the top. It exhibits no distortion. The spectrum of the 8 point interpolated and sampled waveform is shown on the bottom. It shows considerable distortion.

maximum time step so as get to at least 10–20 steps per period of the largest harmonic of interest.

Figure 5.13 on the facing page shows an exact cosine and one that is linearly interpolated to 55 roughly equally-spaced points. The spacing of the time points is typical of what would occur in a circuit simulation. Notice that the error is smaller near the ends of the interval. This results because the simulator typically takes smaller steps at the beginning and end of the interval. While the error due to interpolation is seemingly inconsequential in the time domain, it is quite obvious in the frequency domain. Figure 5.14 shows the spectrum of the interpolated cosine. The error due to interpolation is seen to be about 70 dB below the carrier.

5.3.1.5 Error Due to Simulator Noise

All simulators generate a certain amount of noise that, during a Fourier analysis, obscures very small signals. For example, the Newton convergence criteria and the numerical integration algorithm by their very nature generate a small amount of error. Both sources of error can be significantly reduced by tightening `reltol`.

Many simulators use bypass algorithms to get better efficiency on large digital circuits. With bypass, components that are considered to be latent are not evaluated. The currents computed on the previous time point are used instead. When components come out of bypass, there is a small discontinuity in the current waveforms. The aggregate effect of all the small discontinuities in the currents of components coming out of bypass is to create a noise floor in the Fourier analysis that prevents small signals from being resolved. Unfortunately, the bypass algorithm on most simulators cannot be disabled.

While Spectre does not use bypass algorithms, it does have a few things that can generate very small levels of noise (much smaller than the noise generated by SPICE's bypass algorithm, and generally smaller than the noise that results from the nonideal convergence criteria). If you are concerned with getting the very best performance, set `dskip=no` on all semiconductor model statements and set the option `approx=no`.

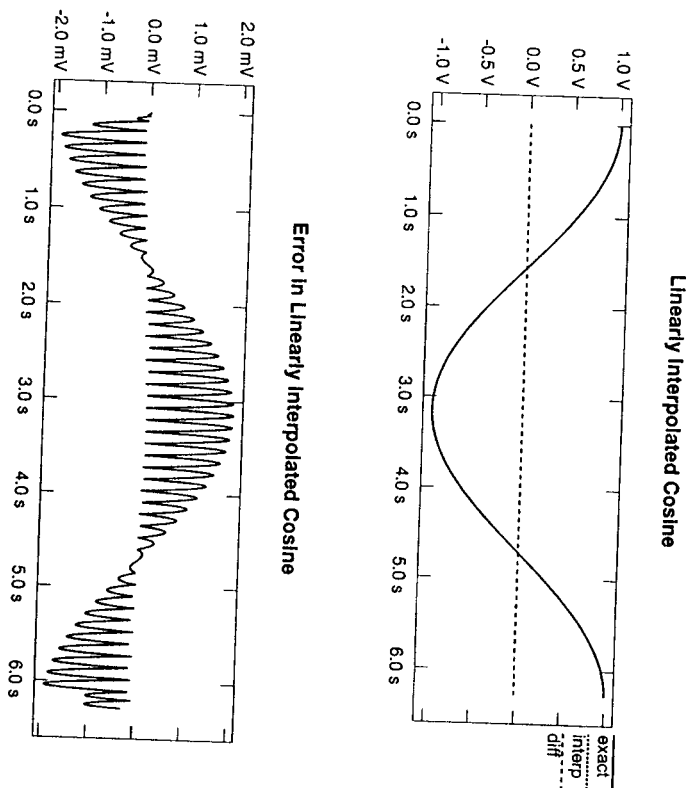


Figure 5.13: A cosine interpolated to 55 roughly equally-spaced points. The difference between the exact cosine and the interpolated cosine is shown in the lower graph.

Transform of Linearly Interpolated Cosine

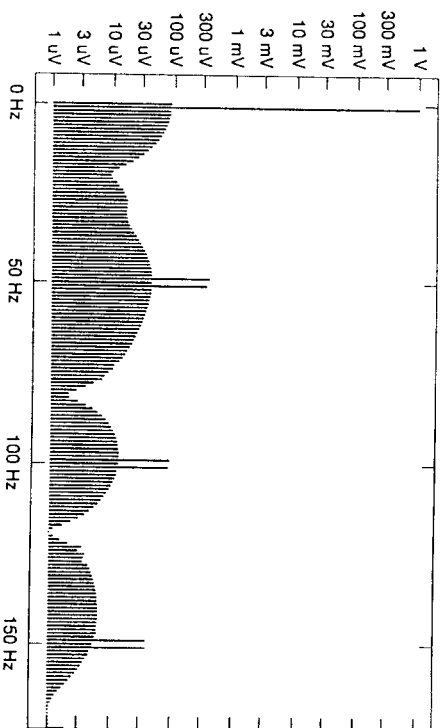


Figure 5.14: The spectrum of the interpolated cosine of Figure 5.13 on the preceding page.

Finally, remember that in general circuit simulators produce less accurate answers when computing currents through capacitors or voltages across inductors, and this could dramatically reduce the accuracy of a Fourier analysis of these quantities. Return to Section 4.2.2.5 on page 152 for a description of these problems.

5.3.2 SPICE's Fourier Analysis

SPICE2, as well as most other circuit simulators, provide a Fourier analysis (generally called the `.fourier` analysis) that can be used to compute the amplitude and the phase of the first nine harmonics and an estimate of the THD. The waveforms are analyzed for one period of the fundamental frequency. If the transient analysis interval is longer than the Fourier analysis interval, then the initial portion of

the transient analysis interval is ignored by the Fourier analysis.

SPICE's Fourier analysis is implemented using the discrete Fourier transform, which requires evenly spaced time points. Linear interpolation creates a set of evenly spaced points from the unevenly spaced points naturally generated by the transient analysis.

This approach to Fourier analysis can be made to accurately resolve harmonics over 100 dB below the fundamental on a good implementation of SPICE if the user is *very* careful and if the signal has very little energy at the harmonics. Without taking special precautions to achieve accurate results, SPICE can easily generate disappointing results.

The errors in SPICE's Fourier analysis spring mainly from interpolation and aliasing in the DFT. The `.fourier` command is intended to be used to compute the distortion of low-distortion sinusoids. Thus, as long as a reasonable number of time points are used, aliasing is not typically an issue (unless the periodic extension of the waveform is discontinuous due to incorrect period or incomplete settling). The DFT is applied to the interpolated signal, which differs from the true signal by an amount that is related to the curvature of the signal and the time between the interpolation point and the closest time points computed by the simulator. The greater the curvature and the farther the interpolation point is from the time points, the greater the error. In SPICE there are several reasons why interpolation can lead to unacceptable error:

1. In Fourier analysis, users are interested in resolving very small signals. Thus, even small interpolation errors are a problem.
2. SPICE's Fourier analyzer does not bound the time step during the analysis period in order to assure reasonable accuracy of the Fourier analysis results.
3. SPICE controls the time step to assure that second-order interpolation is accurate on capacitor charge waveforms and inductor flux waveforms, not to assure accurate Fourier analysis.
 - (a) The Fourier analysis uses first-order interpolation and so

requires a smaller time step to achieve the same accuracy as the second-order interpolation used by the simulator.

- (b) Fourier analysis is performed on node voltages and voltage source currents rather than on capacitor charges and inductor fluxes. Thus, the simulator is not choosing the time steps to control error in the signals being Fourier analyzed.

- (c) If a circuit contains only small capacitors, `chgtol` effectively loosens the local-truncation error criterion, allowing the simulator to take large time steps even though the Fourier analyzer requires smaller steps.

SPICE2 uses the values of N equally spaced points that are computed by linearly interpolating the unequally spaced data naturally generated in a transient analysis. N is computed from T_{step} , the printing or plotting increment given as the first argument on the `.tran` statement.

$$N = \frac{T_{\text{stop}} - T_{\text{start}}}{T_{\text{step}}} + 1 \quad (5.25)$$

T_{start} and T_{stop} are the start and stop times specified on the transient analysis statement. They are *not* the start and stop times for the Fourier analysis interval. From these N points, SPICE uses the DFT to compute 9 harmonics. Thus, energy in harmonics $N - 9$ to $N + 9$, $2N - 9$ to $2N + 9$, etc., masquerades or aliases as energy in the computed harmonics and represents error. Keep in mind that though SPICE interpolates to N points for the Fourier analysis, it may start from fewer points if the Fourier analysis interval is smaller than the transient analysis interval. SPICE may use as few as 50 points in a transient analysis, and of those, many may fall outside the Fourier analysis interval. To prevent aliasing that may be *much* worse than would normally be expected from N points, you should set the maximum time step so that there are at least 100 time points in the Fourier analysis interval.

By default, SPICE3 uses 200 points and linear interpolation to compute its DFT. When running interactively, the number of points can be specified by setting `fourgridsize` to the desired number of points. The interpolation order can be specified by setting `polydeg`.

5.3. Fourier Analysis Practice

281

Finally, the number of harmonics computed is specified by setting `nfreqs` to the desired number. If `nfreqs` is not set, 9 harmonics are computed.

Like SPICE2, the transient analysis in SPICE3 can compute many fewer than 200 points in the Fourier analysis interval, which would result in additional aliasing. You should set the maximum time step so that there are at least $10 \times K$ time points during the Fourier analysis interval where K is the highest harmonic of interest.

Other sources of error can also limit the resolution of the Fourier analysis. For example, discontinuities in the model equations due to incomplete models, errors, or bypass algorithms generate small jumps in the waveforms being analyzed. In addition, the small errors that result from incompletely converging Newton's method also contributes to reducing the resolution of the Fourier analysis. These errors are reduced by tightening `reltol`.

5.3.2.1 Remedies for Errors in SPICE's Fourier Analysis

There are two approaches to making SPICE's Fourier analysis more accurate. The most important thing to do is to set the maximum time step to avoid gross errors. To get the best performance, tighten `reltol` as well.

1. Shrink the maximum time step (the fourth parameter on the transient analysis statement). Set the maximum time step to be no larger than $T/(10 \times K)$ where K is the largest harmonic computed. Shrinking the maximum time step directly acts to reduce the interpolation error. If set sufficiently small it can eliminate the aliasing that occurs in the simulator before the waveform is passed to the Fourier analyzer. For the best resolution, it is also necessary to tighten `reltol` to reduce the error due to the Newton convergence criteria.

Setting the maximum time step is particularly important whenever the Fourier analysis interval is a small fraction of the total simulation interval. This usually occurs when it is necessary to simulate through many cycles to allow long initial transients to

settle out.

2. If the signal contains energy at many harmonics, set T_{step} small enough (or `fourgridsize` large enough in SPICE3) that the number of points used, N , is large enough to avoid aliasing. You must also set the maximum step size so that it is no larger than T/N , where T is the Fourier analysis interval.
3. Tighten `reltol`, which has the effect of reducing noise from the Newton convergence criteria. It also shrinks the time step, which reduces the interpolation error.
4. In SPICE3, increase `fourgridsize` and `polydeg`. Do not bother tightening `fourgridsize` unless you also shrink the maximum time step. There is no significant benefit in making the Fourier grid any finer than the maximum time step in the Fourier interval.

5.3.3 Spectre's Fourier Analysis

Spectre uses a new approach to Fourier analysis that computes the Fourier coefficients using the Fourier integral rather than the DFT. This approach has two important benefits. First, it uses the unequally spaced time points naturally generated by the simulator. As a result, fewer time points are used, making the simulation less expensive (assuming time required for computing the circuit response dominates over the time required for Fourier analysis). Second, this approach does not suffer from aliasing. It accurately computes the Fourier coefficients for a few harmonics, even though there is significant energy in the harmonics that are not being computed. This second advantage is an important one, because it allows the Fourier analysis to be performed on signals it would not normally be suited for, such as the waveforms generated by $\Sigma\Delta$ and pulse-width modulators, mixers, DACs and SC-filters.

5.3.3.1 The Fourier Integral

For a periodic waveform x of period T , the Fourier coefficients are computed with the following formulas,

$$a_0 = \frac{1}{T} \int_t^{t+T} x(\tau) d\tau \quad (5.26)$$

$$a_k = \frac{2}{T} \int_t^{t+T} x(\tau) \cos\left(\frac{2\pi k\tau}{T}\right) d\tau \quad (5.27)$$

$$b_k = \frac{2}{T} \int_t^{t+T} x(\tau) \sin\left(\frac{2\pi k\tau}{T}\right) d\tau \quad (5.28)$$

where $k = 1, 2, \dots$. The derivation that follows uses (5.27) to demonstrate the calculations. Similar steps should be taken with (5.26) and (5.28) to compute a_0 and b_k .

The circuit simulator discretizes time and solves the system of equations that describe the circuit at $N + 1$ time points, $t_0, t_1, \dots, t_N$. Equation (5.27) is rewritten as a sum of integrals over each time step as shown in Figure 5.15 on the next page,

$$a_k = \frac{2}{T} \sum_{n=1}^N \int_{t_{n-1}}^{t_n} x(\tau) \cos(k\omega\tau) d\tau \quad (5.29)$$

where $\omega = 2\pi/T$ and $t_N - t_0 = T$.

Between time points, x can be approximated with a low-order polynomial as shown in Figure 5.16. This is very natural because the circuit simulator makes a very similar approximation in order to perform numerical integration. During time-step n ,

$$x(\tau) \approx \sum_{m=0}^M c_{mn} \tau^m \quad (5.30)$$

for $t_{n-1} \leq \tau \leq t_n$, where M is the order of the approximating polynomial. Substituting into (5.29) gives

$$a_k \approx \frac{2}{T} \sum_{n=1}^N \int_{t_{n-1}}^{t_n} \sum_{m=0}^M c_{mn} \tau^m \cos(k\omega\tau) d\tau. \quad (5.31)$$

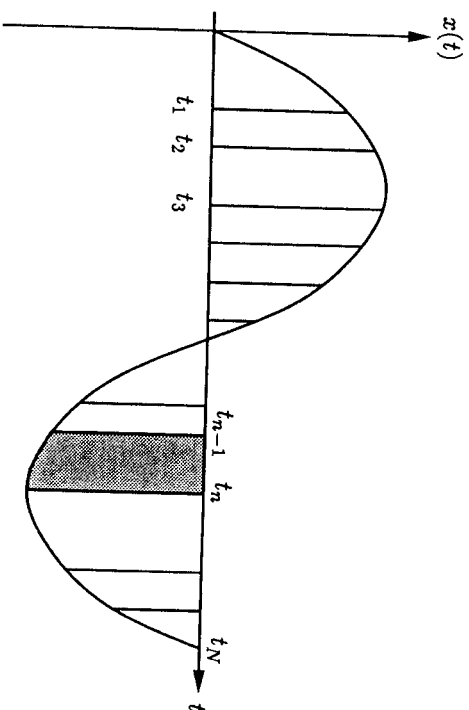


Figure 5.15: The Fourier integral is reformulated as a sum of integrals, where each integral spans a single time step.

Exchange the order of integration and summation,

$$a_k \approx \frac{2}{T} \sum_{n=1}^N \sum_{m=0}^M c_{mn} \int_{t_{n-1}}^{t_n} \tau^m \cos(k\omega\tau) d\tau. \quad (5.32)$$

Finally,

$$a_k \approx \frac{2}{T} \sum_{n=1}^N \sum_{m=0}^M c_{mn} \alpha_{kmn} \quad (5.33)$$

where

$$\alpha_{kmn} = \int_{t_{n-1}}^{t_n} \tau^m \cos(k\omega\tau) d\tau. \quad (5.34)$$

This integral is in a simple form that can be evaluated analytically using integration by parts and induction.

$$\int \cos(k\omega\tau) d\tau = \frac{1}{k\omega} \sin(k\omega\tau) \quad (5.35)$$

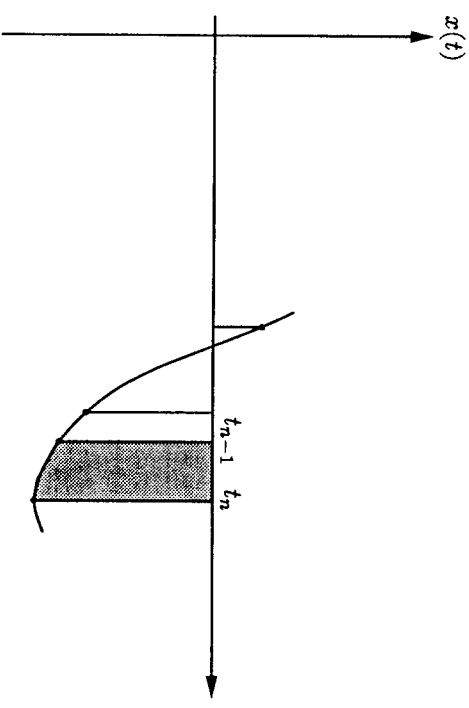


Figure 5.16: The waveform is approximated as a low-order polynomial over each time step.

$$\begin{aligned} \int \tau \cos(k\omega\tau) d\tau &= \frac{1}{(k\omega)^2} \cos(k\omega\tau) \\ &+ \frac{\tau}{k\omega} \sin(k\omega\tau) \end{aligned} \quad (5.36)$$

$$\begin{aligned} \int \tau^2 \cos(k\omega\tau) d\tau &= \frac{2\tau}{(k\omega)^2} \cos(k\omega\tau) \\ &+ \frac{(k\omega\tau)^2 - 2}{(k\omega)^3} \sin(k\omega\tau) \end{aligned} \quad (5.37)$$

$$\begin{aligned} \int \tau^3 \cos(k\omega\tau) d\tau &= \frac{3(k\omega\tau)^2 - 6}{(k\omega)^4} \cos(k\omega\tau) \\ &+ \frac{(k\omega)^2 \tau^3 - 6\tau}{(k\omega)^3} \sin(k\omega\tau) \end{aligned} \quad (5.38)$$

This can be carried on to arbitrarily high order using integration by parts, though in practice, third-order polynomials are generally sufficient.

To review, the procedure for computing the Fourier coefficients is:

1. Break the integral into a sum of integrals, one for each time step as in (5.29).
2. Approximate the waveform with a low-order polynomial over each time step as in (5.30).
3. Analytically evaluate the integral over each time step for each order of τ individually using (5.35–5.38).
4. Sum the integrals for each order and each time step, as in (5.33).

5.3.3.2 Interpolation Error

Approximating the waveform with a low-order polynomial over each time step introduces a small amount of error that is referred to as interpolation error. Interpolation error, along with errors generated by the transient analysis algorithms, limit the resolution of the Fourier analysis.

Unlike the Fourier integral approach, the DFT is not subject to interpolation errors as long as the sampled waveform that serves as the input for the DFT is not already contaminated with interpolation error. However, the DFT is equivalent to numerically integrating the Fourier integrals on a uniform grid using trapezoidal rule. As such, it is also equivalent to the Fourier integral approach on the same fixed grid using linear interpolation. And yet, the DFT is not subject to interpolation error, while with the Fourier integral approach, interpolation error is one of the dominant sources of error. This apparent contradiction was resolved by Euler and Maclaurin, who showed that when the trapezoidal rule is applied to a periodic waveform that is continuous in all derivatives the results are exact if uniformly spaced time points are used. In this case, the interpolation errors all cancel.

Euler-Maclaurin can be applied to the Fourier integral approach if the order of interpolation is limited to first order, and so it is also exact when uniformly-spaced time points are used. In addition, empirical

results show that the Euler-Maclaurin results also hold when second-order interpolation is used. Thus, the DFT and Spectre's Fourier integral approach are both exact when equally-spaced time points are used.

The interpolation error that results when Fourier analysis is performed by SPICE does not cancel even when equally-spaced points are used because the interpolation is applied in a different manner. With the Fourier integral and the DFT, the interpolation is part of the numeric integration algorithm. However, in SPICE the interpolation is performed in advance of the numeric integration. Thus, the interpolation error is injected before the DFT is applied.

To avoid interpolation error, use small time steps. If possible, reduce the maximum time step to the degree that all of the time points used by the simulator during the Fourier analysis are equally spaced. Doing so results in a tremendous increase of accuracy.

5.3.3.3 Implementation in Spectre

Spectre is different from SPICE in a key way that helps to improve the resolution of the Fourier analysis. Rather than choosing the time step to control the error in polynomial approximation of the *charge* waveforms on capacitors only (neglecting inductors), Spectre chooses the time step to control error in the polynomial approximations of the *voltage* waveforms on capacitors. With the addition of the Fourier analyzer, Spectre also chooses the time step to control error in the polynomial approximations of the voltage waveforms at the input of the Fourier analyzer. As a result, simply tightening `reltol` directly acts to improve the accuracy of the Fourier analysis.

Several additional methods are applied simultaneously in Spectre to reduce error from interpolation.

1. Higher order interpolation methods are used. Typically, second-order interpolation is used, though in special circumstances first order interpolation is used on at most a few time points.
2. The Fourier analyzer forces Spectre to place a time point at

both the beginning and the end of the Fourier analysis period (using Spectre's break-points).

3. The Fourier analyzer limits the maximum size of the time step during the Fourier analysis (only) to assure that there are at least 10 time points per period of the largest harmonic being computed. The Fourier analyzer also provides the points parameter to allow you to increase that number. The time step is only controlled in the Fourier analysis interval, not before. This allows you to quickly simulate through an initial transient without sacrificing accuracy in the Fourier analysis.

Finally, Spectre does not use bypass algorithms, thereby eliminating a numerical noise source present in SPICE.

5.3.3.4 Remedies for Errors in Spectre's Fourier Analysis

Though Spectre's Fourier analyzer is considerably more accurate than SPICE's, there are still times when you would like it to be more accurate still. As with SPICE, to increase the accuracy of Spectre's Fourier analyzer, you should tighten the simulator tolerances, and shrink the maximum time step. Spectre allows you to specify the maximum time step for the Fourier analysis alone. This is useful because it allows you to bound the time step during the Fourier analysis when it is useful, while not forcing the simulator to take small time steps before the Fourier analysis. Generally, the simulation interval before the Fourier analysis is used to allow any initial transients to decay. Small time steps during this initial interval are unnecessary.

1. Tighten `reltol`, which has the effect of reducing noise from the Newton convergence criteria. It also shrinks the time step, which reduces the interpolation error.
2. Do not start the Fourier analysis at the first point of a transient analysis. Allow the time step to stabilize by waiting for at least 5–10% of the transient analysis to complete before starting the Fourier analysis.

3. Shrink the maximum time step. Shrinking maximum time step directly acts to reduce the interpolation error. Use the Fourier analysis points parameter to avoid shrinking the time step outside the Fourier analysis interval. If possible, use enough time points to make them equally spaced, which eliminates interpolation error.
4. Specify `method=traponly` or `method=gear2only` to try to prevent the simulator from unnecessarily changing the integration method.
5. Assure input signal is periodic. The Fourier integral approach employed by Spectre may not be sensitive to aliasing, but it still requires signals to be periodic.
6. Try to avoid 'events' such as break points during the Fourier analysis period.

Be aware that abrupt activity even in a completely isolated section of the circuit can cause degradation of the noise floor of the Fourier analysis because of its affect on the time step and integration method.

5.3.4 External Fourier Analysis

Most circuit simulators are used with some kind of program that processes and plots simulation results. Most of these provide Fourier analysis in the form of a windowed FFT. The windowing allows these Fourier analyzers to be applied to non-periodic signals or periodic waveforms with unknown periods, such as those created by autonomous circuits (oscillators).

External Fourier analyzers are applied to waveforms after the transient analysis has been run. Unlike Fourier analyzers built into the simulator, they cannot control the placement of the time points and must take whatever points the simulator gives them. To control error, you can follow the suggestions given for SPICE. That is, to tighten the simulator tolerances (`reltol` and the maximum time step), or you can force the simulator to place time points exactly where the Fourier analyzer samples the waveform. The best way to do so is to

use strobing or a component that precisely samples the waveform at the proper points (such as one of Spectre's z-blocks). However, with SPICE one is usually forced into using a round-about approach that involves using a piecewise-linear or pulse source to generate break points at the desired sample points.

5.3.4.1 Windowing

Window functions are provided to allow you to analyze waveforms that are not periodic or have unknown periods. Windows address these problems by modifying the signal to make it smoothly T -periodic and thereby eliminate the discontinuities. Windows are waveforms that are zero up to the start time of the Fourier analysis and zero after the stop time. During the analysis they generally move smoothly from zero at the beginning of the interval up to some maximum value and then back to zero at the end of the interval. There are any number of ways to accomplish this.² Several common windows are shown in Figures 5.17–5.23. Not using a window is equivalent to using a rectangular window. A rectangular window is zero everywhere outside the Fourier analysis interval, and one everywhere inside. See the comprehensive survey article written by Harris for more information on windows [harris78].

Window functions are multiplied by the input waveform before the Fourier analyzer is applied. The spectrum of the product is equal to the spectrum of the input signal convolved with the spectrum of the window. The convolution acts to spread out any narrow peaks in the spectrum of the input and fill in any deep valleys. Both of these artifacts are undesirable. To minimize them, you must choose a window function that has a spectrum with a narrow main lobe and drops quickly to zero. The width of the main lobe determines the frequency resolution of the results, and the level of the response away from the main lobe determines the amplitude resolution. Choosing a good window involves trading off frequency and amplitude resolution. Here are some useful guidelines.

²At one point, developing a window function was seemingly the easiest way to be immortalized. A list of window functions reads like a roll-call of dead mathematicians.

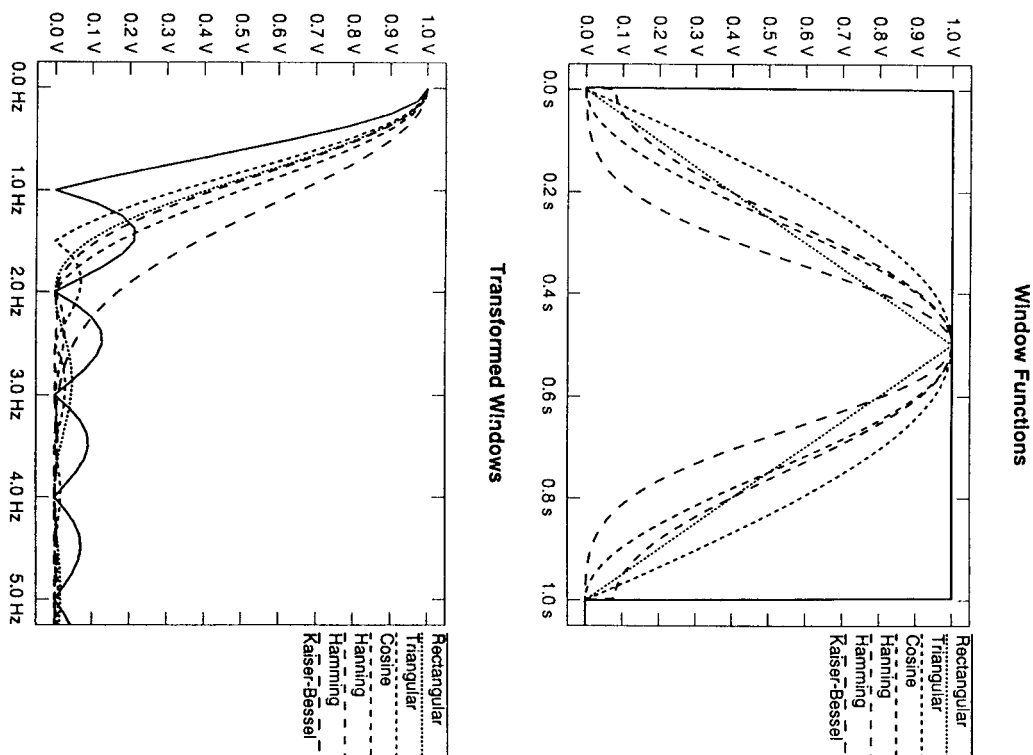


Figure 5.17: Several common window functions and their spectra.

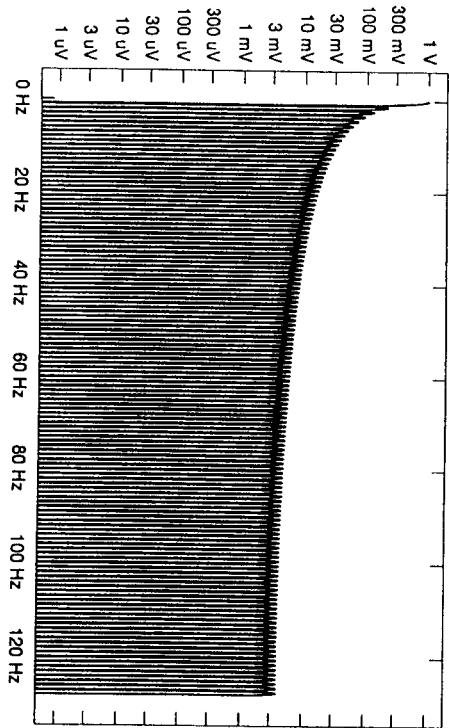
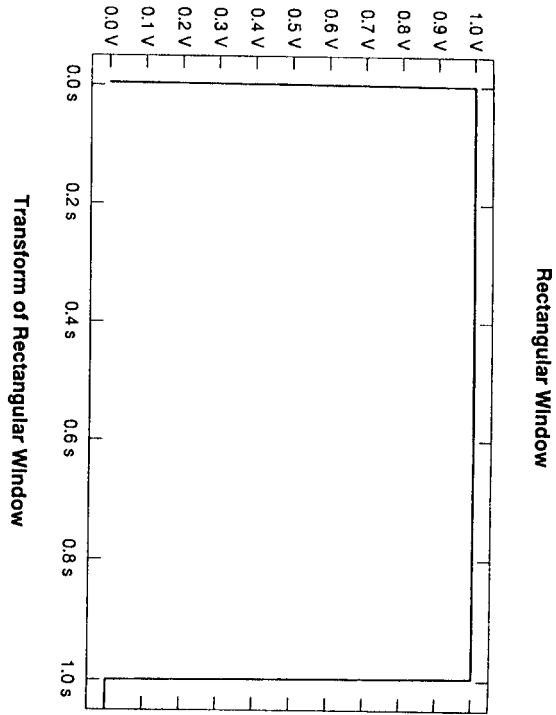


Figure 5.18: Rectangular (or Dirichlet) window function and its transform.

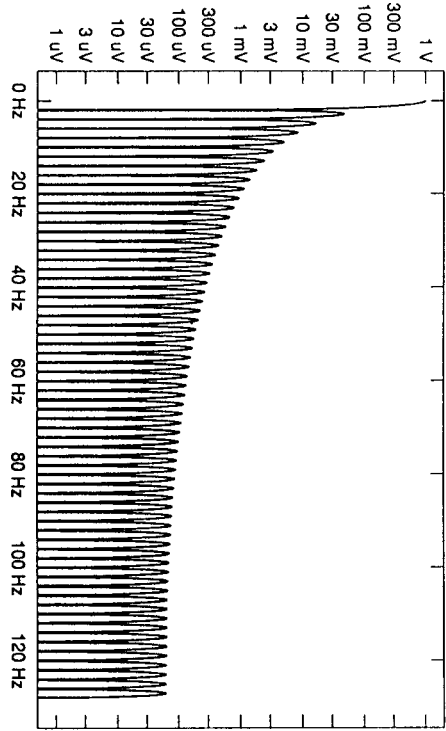
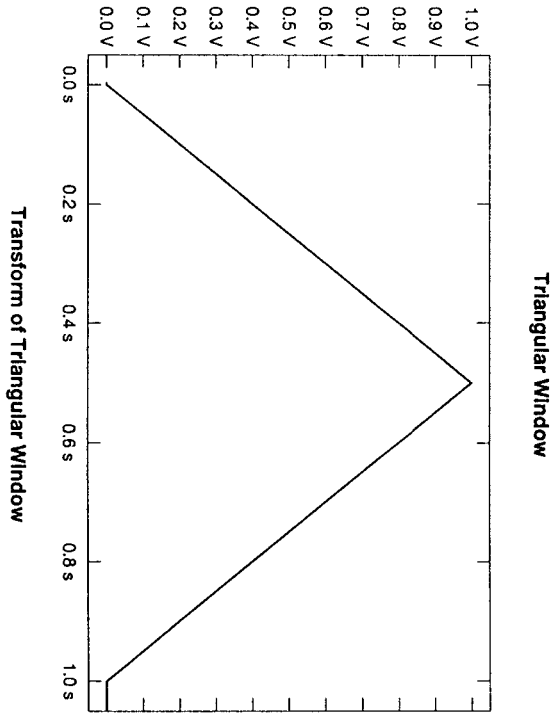


Figure 5.19: Triangular (or Bartlet) window function and its transform.

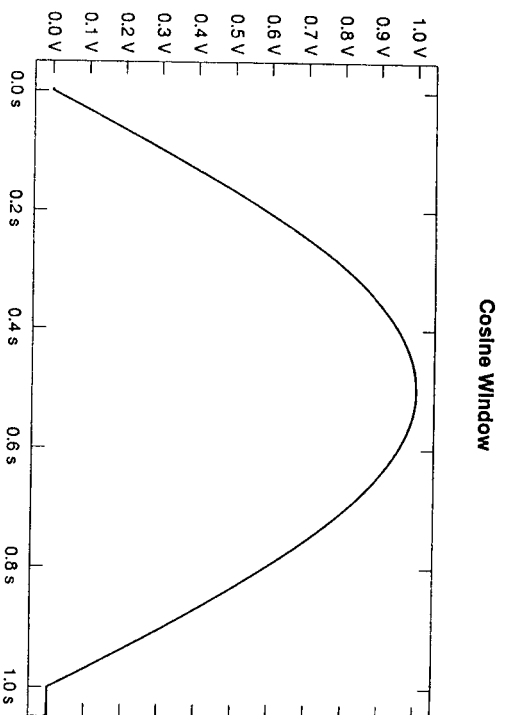


Figure 5.20: Cosine window function and its transform.

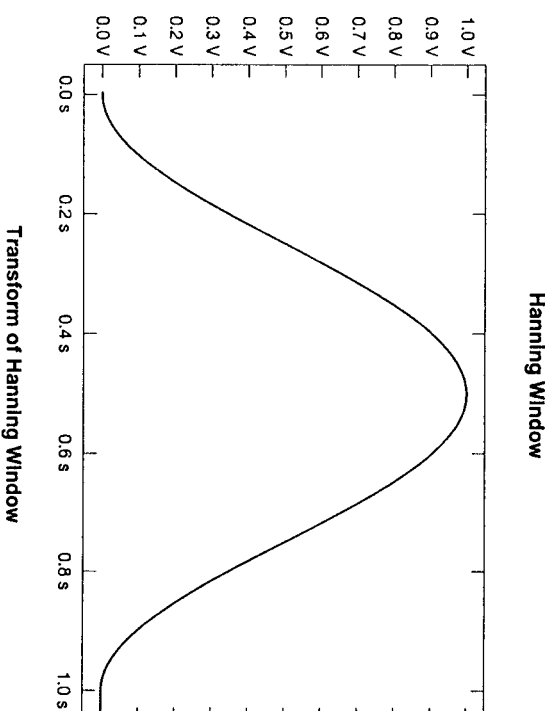


Figure 5.21: Hanning (or $\cos^2$) window function and its transform.

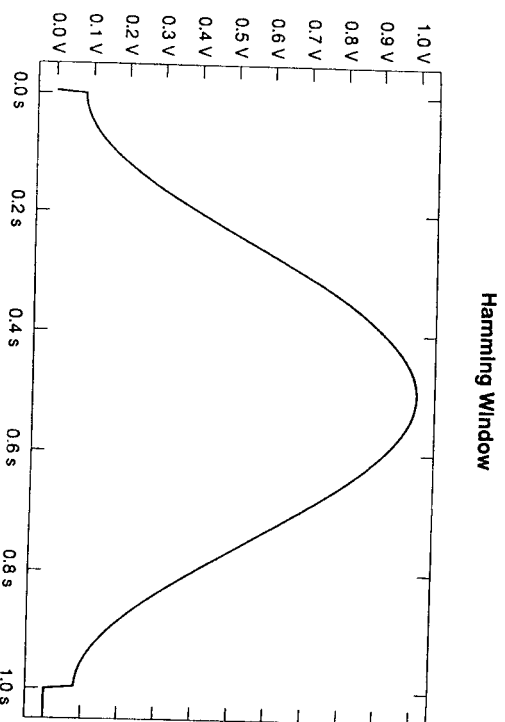


Figure 5.22: Hamming window function and its transform.

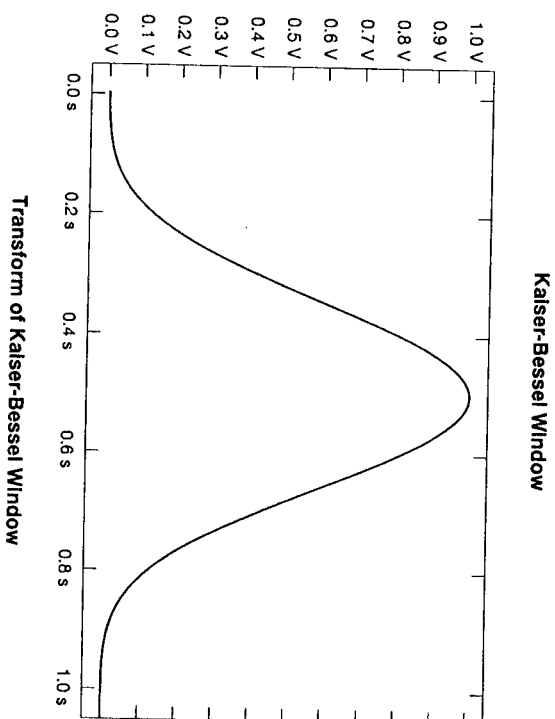


Figure 5.23: Kaiser-Bessel window function with parameter $\alpha = 3.5$ and its transform.

Improving frequency resolution: The frequency resolution is improved in general if the effective width of the window is made as wide as possible. Thus, flat-topped windows that quickly fall to zero near the edge of the interval result in the narrowest main lobes and so give the best frequency resolution. The rectangular window has the best frequency resolution.

Improving amplitude resolution: The amplitude resolution is improved in general by using the smoothest possible window function. The periodic extension to the window function, and as many of its time derivatives as possible, should be zero at the boundaries. The Hanning and Kaiser-Bessel windows have good amplitude resolution.

Generally the ratio between the best and the worst frequency resolutions is about a factor of 2–4, whereas the ratio of the best to worst amplitude resolution can be 60 dB–80 dB. Thus, it is usually best to choose a window function that emphasizes amplitude resolution.

One can improve both the frequency and amplitude resolution by choosing a window that provides good amplitude resolution and then increasing the number of points used.

5.3.4.2 Remedies for Errors in an External Fourier Analysis

1. Shrink the maximum time step. Shrinking the maximum time step directly acts to reduce the interpolation error. For the best resolution, it is also necessary to tighten `reltol` to reduce the error due to the Newton convergence criteria. Set maximum time step to be no larger than $T/100$.
2. Tighten `reltol`, which has the effect of reducing noise from the Newton convergence criteria. It also shrinks the time step, which reduces the interpolation error.
3. Use a sampler in the simulator to avoid interpolation error. Of course, the sampler must be constructed so that it forces the simulator to place a time point at each sample point. This approach is particularly appropriate if the output would be

5.4. Applications of Fourier Analysis

299

naturally sampled anyway, such as if it were followed by a sample-and-hold or an ADC. Spectre provides several components that accept z-domain transfer functions. Specifying a frequency independent transfer function (such as unity) to one of these components is a good way of implementing an ideal sampler. One can also use strobing, which is discussed in Section 4.4.10 on page 238.

4. An alternative approach to 3 is to add a triangle wave source in a subcircuit that is disconnected from the rest of the circuit with corners evenly spaced along the Fourier analysis interval (put a corner at each end of interval and corners at each of the times inside the interval that are used as sample points by the Fourier analysis). Each corner generates a break point which forces the simulator to place a time point exactly on the corner, and so on the Fourier analysis sample point. This eliminates interpolation as a source of error. However, it is not an ideal solution because the break points cause the simulator to switch to backward Euler, which reduces the resolution of the Fourier analysis.

5. If signal is not periodic or if period is unknown, use a window function to improve the resolution of the analysis. Good windows to use are those that are very smooth and so have spectra that drop quickly to zero.
6. If aliasing is a problem, increase the sampling frequency.
7. If there is insufficient amplitude resolution, use a smoother window.
8. If there is insufficient frequency resolution, use more points while keeping the sampling rate constant.

5.4 Applications of Fourier Analysis

5.4.1 THD of Low Distortion Amplifier

Low distortion amplifiers are found in demanding applications where linearity is crucial. Examples of such applications include high res-

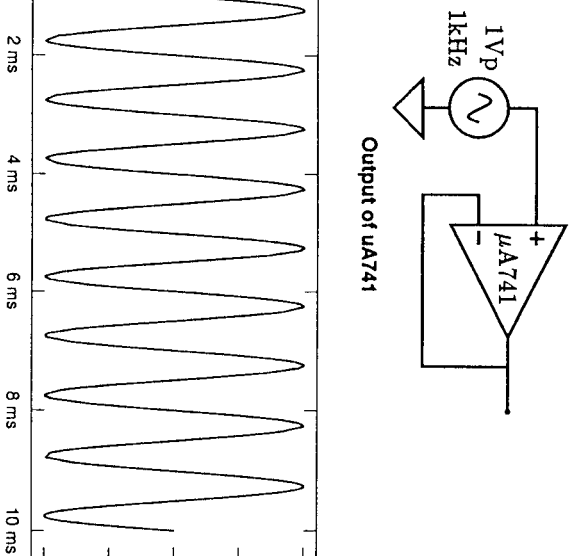


Figure 5.24: The unity gain test circuit for the $\mu A741$ and its response.

olution data-acquisition systems and high-fidelity audio equipment. When driven by a sinusoid, the distortion products generated by these amplifiers is often required to be 80–120 dB ($10^4 - 10^6$) below the fundamental. In order to measure this level of detail accurately, a transient simulator must place its time points very closely. It also must compute the time-domain response until the initial transient has decayed to a level well below the expected distortion products. Both of these contribute to the expense of the simulation.

In order to illustrate the process of computing the distortion of a highly linear circuit, Fourier analysis is applied to a low distortion feedback amplifier. The amplifier is the $\mu A741$ opamp in unity-gain configuration shown in Figure 5.24. The input frequency is low at

harmonic	SPICE2	Spectre	Harm. Bal.
fund	877 mV	1 V	1 V
2nd	-46.8 dB	-98.6 dB	-98.5 dB
3rd	-48.4 dB	-116 dB	-120 dB
4th	-24.2 dB	-125 dB	-169 dB
5th	-51 dB	-124 dB	-143 dB
6th	-31.2 dB	-125 dB	-196 dB
7th	-57.7 dB	-125 dB	-165 dB
THD	6.9 %	0.0012 %	0.0012 %

Table 5.1: Results from $\mu A741$ using default tolerances and time steps. SPICE gives poor results because it places only a few time points in the Fourier analysis interval.

1 kHz and the input amplitude is small at 1 Vp, so the distortion is very low. The distortion was computed from 9 harmonics using the Fourier analysis in SPICE2 and in Spectre on a 10ms transient analysis. Nothing was done in either simulation to improve the results (default values were used for both `reltol` and the maximum time step; in SPICE, T_{step} was set to 1 ms). Harmonic balance was also run on the circuit as a control (periodic harmonic balance has the property of becoming extremely accurate as signals approach being pure sinusoids [Kundert90]). The results, given in Table 5.1, show that because SPICE does not control the time step to accurately compute Fourier coefficients, its results are often inaccurate. Table 5.2 shows how the Spice results improved as the print interval (T_{step}), the maximum time step, and `reltol` were all tightened.

Spectre with `reltol` = 10^{-3} resolves signals down to about -120 dB. This “noise” floor results from errors in the waveform as computed by the simulator rather than from errors in the Fourier calculations. This is confirmed by running the simulators with `reltol` = 10^{-6} (no bounds were placed on the time steps; in SPICE, T_{step} = 1 ms), as shown in Table 5.3.

T_{step}	100 μs	100 μs	100 μs	10 μs	10 μs
T_{max}	100 μs	100 μs	10 μs	10 μs	10 μs
reltol	10^{-3}	10^{-6}	10^{-3}	10^{-3}	10^{-6}
fund	968 mV	996 mV	1 V	1 V	1 V
2nd	-69 dB	-75 dB	-79 dB	-98 dB	-99 dB
3rd	-73 dB	-58 dB	-90 dB	-118 dB	-122 dB
4th	-74 dB	-76 dB	-97 dB	-140 dB	-138 dB
5th	-74 dB	-61 dB	-101 dB	-134 dB	-141 dB
6th	-74 dB	-74 dB	-104 dB	-131 dB	-148 dB
7th	-75 dB	-64 dB	-108 dB	-138 dB	-146 dB
8th	-81 dB	-70 dB	-110 dB	-132 dB	-158 dB
9th	-38 dB	-67 dB	-114 dB	-135 dB	-151 dB
THD	1.23%	0.175%	0.012%	0.0013%	0.0011%

Table 5.2: Results from SPICE on $\mu\text{a}741$ using various tolerances and time steps. T_{step} is the print interval and T_{max} is the maximum time step.

harmonic	SPICE2	Spectre	Harm. Bal.
fund	996 mV	1 V	1 V
2nd	-75.4 dB	-98.5 dB	-98.5 dB
3rd	-58 dB	-120 dB	-120 dB
4th	-74.6 dB	-159 dB	-169 dB
5th	-61.1 dB	-142 dB	-143 dB
6th	-73.7 dB	-164 dB	-196 dB
7th	-63.4 dB	-165 dB	-165 dB
THD	0.18 %	0.0012 %	0.0012 %

Table 5.3: Results from $\mu\text{a}741$ with reltol = 10^{-6} .

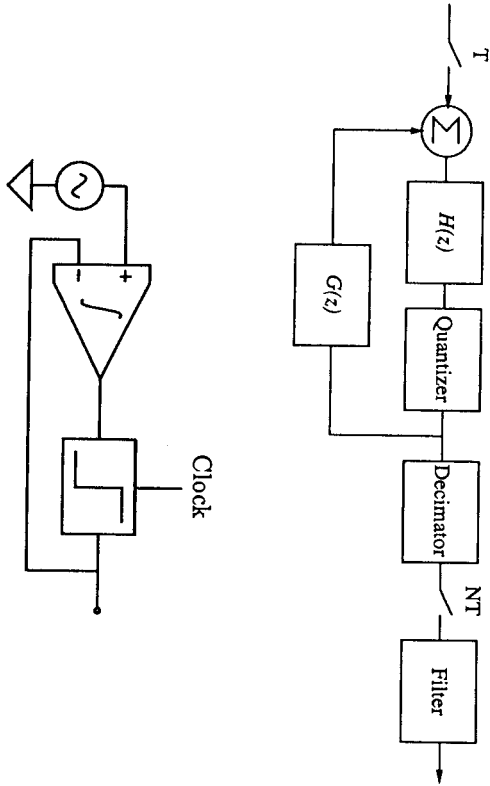


Figure 5.25: The block diagram of a $\Sigma\Delta$ converter (top), and a the block diagram of a simplified first-order $\Sigma\Delta$ modulator (bottom).

5.4.2 Resolution of a $\Sigma\Delta$ Modulator

The block diagram of a generic $\Sigma\Delta$ modulator is shown in Figure 5.25. Also shown is the block diagram of a simplified first-order $\Sigma\Delta$ modulator where $H(z)$ has been replaced by an integrator ($1/(z - 1)$) and $G(z)$ has been set to 1. $\Sigma\Delta$ modulators can be understood at a simple level by considering a first-order modulator, by initially ignoring the quantizer, and by recognizing that the resulting circuit is simply a feedback loop. The feedback is such that the output is forced to be the same as the input. If the integrator is ideal, and if the input signal is constant, then eventually, the output is exactly equal the input. However, with the quantizer present, the output can only take one of a small number of values. For example, a two-level quantizer might force the output to be either 1 or -1. Thus, with the quantizer present, the feedback forces the average

value of the output to be equal to the average value of the input. For example, if the input were a constant 0 volts, then with our simple two-level quantizer the modulator would output an endless sequence alternating between 1 and -1 . If the input were 0.5 volts, the output sequence would be $\{1, 1, 1, -1\}$ repeated endlessly. Notice that in these two examples, the period of the repeating sequence is different. There are some inputs for which the output sequence never repeats. Consider an input of $\sqrt{2}/2$. Since it is irrational, it cannot be exactly represented as the ratio of two integers and so also cannot be represented as the average of a finite number of integers. Thus, to exactly represent $\sqrt{2}/2$, the $\Sigma\Delta$ modulator has to average an infinite number of samples. In practice, only a finite number of samples can be averaged, which leads to a small amount of error. This error is generally referred to as quantization noise because it varies in a pseudo-random way and so behaves like noise. The longer the period of averaging, the smaller the error. The averaging is done by the decimator and filter, both of which are generally implemented as finite-impulse response (FIR) filters. Thus the period of averaging is finite and known.

As can be seen by the behavior described above, the noise pattern is dependent on the input data. So use a realistic signal, because an overly simplified waveform, such as a constant, would give numbers that are much different for the real modulator responding to real signals.

At any point in time the output of a $\Sigma\Delta$ modulator is a very sensitive function of its current state. As a result, the circuit is considered to be chaotic. While chaotic circuits are deterministic, their outputs seemingly vary randomly. Indeed, the statistics, such as the average value, of the waveform generated by a $\Sigma\Delta$ modulator is very predictable at any point in time, but the actual value of the output at any point in time is not. Even if the clock to the $\Sigma\Delta$ modulator and the input signal are commensurate, there is no guarantee that the output signal is periodic. Indeed, even constant-valued inputs may generate non-periodic output waveforms. Designers treat this quasi-random behavior as noise, and design their circuits using multiple loops to minimize this noise. It is not noise in the classic sense, because it is deterministic. However, it behaves very much like random

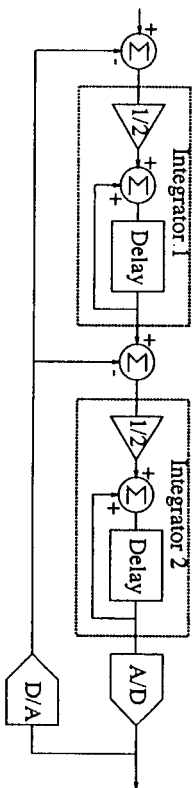


Figure 5.26: Boser and Wooley's second-order $\Sigma\Delta$ modulator architecture.

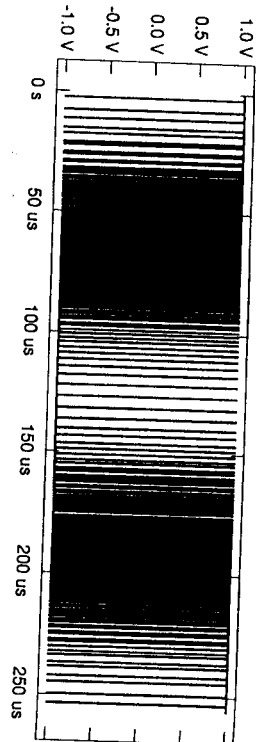
noise and can be treated as such. Even though the output of the $\Sigma\Delta$ modulator is not always periodic, the Fourier analysis can be applied (the waveform already has a large set of discontinuities, what is one more?).

5.4.2.1 A Second-Order $\Sigma\Delta$ Modulator

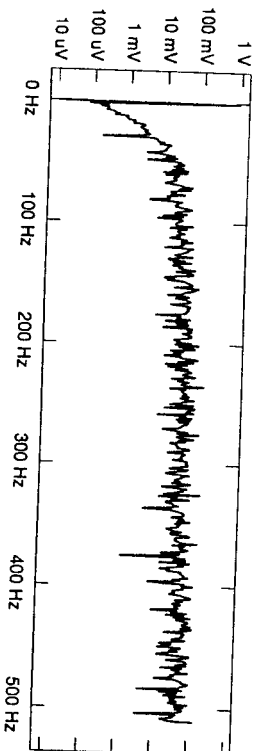
Higher-order modulators are generally preferred over first-order modulators because the quantization noise generated by the higher-order modulators exhibits significantly less correlation with the input signal, and therefore needs a smaller clock rate to achieve a given resolution. Figure 5.26 shows the architecture of a second-order $\Sigma\Delta$ modulator [Boser88]. The circuit is clocked at 4 MHz. The circuit was driven with a 4 kHz sine wave and was simulated for 1024 clock cycles. The output waveform is shown in Figure 5.27

Using an External Discrete Fourier Transform The spectrum of the output signal is shown in Figure 5.28. It was computed by sampling the output signal once per clock period and using the FFT. The transient analysis interval was carefully chosen to generate 2^k samples for the FFT. In addition, the period of the input was chosen so that an even number of periods fit into the simulation interval. The modulator was simulated for about 400 clock periods before

Response of Sigma-Delta Modulator to a Sinusoidal Stimulus

**Figure 5.27:** The response of Boser and Wooley's $\Sigma\Delta$ modulator to a 0.9 volt sine wave.

Spectrum of Sigma-Delta Modulator Output

**Figure 5.28:** The spectrum of the response of Boser and Wooley's $\Sigma\Delta$ modulator to a 0.9 volt sine wave. Notice that the quantization noise drops at low frequencies. This is because the feedback loop acts to attenuate the quantization noise at low frequencies.

any data was taken for the FFT to allow the initial transients to decay. The data for the FFT was taken before the decimator and filter. The decimator acts to reduce the clock rate of the signal. Typically the decimator reduces the clock rate by a factor of 64 to 256. By deleting the decimator and filter, it becomes unnecessary to simulate for a long period simply to get enough samples out of the decimator to determine the statistics of the output. In addition, it is not necessary to simulate long enough for the output filter to settle.

Once the spectrum has been computed, it is possible to compute the resolution of the $\Sigma\Delta$ converter if the characteristics of the decimator and filter are known. Simply compute the total RMS noise over the bandwidth of the filter using

$$\overline{v_{out}^2} = \int_0^\infty |H^2(f)v_{mod}^2(f)|df, \quad (5.39)$$

where $H(f)$ is the transfer function of the filter as a function of frequency, and $v_{mod}(f)$ is the quantization noise at the output of the modulator.

Figure 5.28 shows a drop in noise level at low frequencies that is a characteristic of $\Sigma\Delta$ modulators. The drop in noise results from the action of the feedback. The decimator and filter are used to eliminate the noise at the upper frequencies. Typically, all but the bottom 1% or less of the frequency range is filtered out.

Using Spectre's Fourier Analyzer An important feature of the Fourier integral is that it is not subject to aliasing, allowing it to be applied to broad-spectrum signals. This is illustrated by computing the first 7 harmonics of the signal shown in Figure 5.27.

The output of the $\Sigma\Delta$ modulator is converted into integers by the decimator, which is implemented as a digital signal processor. Thus, the distortion of the modulator is determined solely from the value and the order of the integers, not the details of the waveform such as ringing or slew limiting. Unlike the DFT, which samples the output once per clock period, the Fourier integral computes the spectrum of the entire waveform, ringing and all. It is necessary to pass the signal at the output of the modulator through an ideal sample and

harmonic	Fourier Integral	FFT
fund	0.90 V	0.90 V
2 nd	-82.2 dB	-82.2 dB
3 rd	-71.5 dB	-71.5 dB
4 th	-76.2 dB	-76.2 dB
5 th	-71.3 dB	-71.3 dB
6 th	-71.8 dB	-71.8 dB
7 th	-72.9 dB	-72.9 dB

Table 5.4: The magnitude of the lowest harmonics computed from the $\Sigma\Delta$ modulator response by the Fourier integral and FFT.

hold to eliminate these imperfections. This is easily accomplished in Spectre using a unity gain z-block (the `zrcvs`).

The Fourier integral does not constrain the number of clock cycles, but the Fourier analysis interval should be a multiple of both the clock period and the period of the input signal. Often the period of the input signals is the same as the Fourier analysis period. Again, the modulator is simulated long enough for any initial transients to settle out before the Fourier analysis interval begins.

The magnitudes of the lowest harmonics computed from the second-order two-level $\Sigma\Delta$ modulator waveform by the Fourier integral and FFT are shown in Table 5.4. In this example, the clock was 4 MHz, the input signal was a 0.9 V, 3906.25 Hz sine wave, the Fourier analysis interval was 256 μ s, with 25.6 μ s allotted to allow the modulator to settle.

5.4.3 Distortion of a Pulse-Width Modulator

It is difficult to measure the distortion of a pulse-width modulator using the DFT because of its limited resolution in time. The DFT periodically samples the waveform, and any event that occurs between sample points is missed. Consider the response of a pulse-width modulator to a sinusoidal stimulus that is shown in Figure 5.29. The

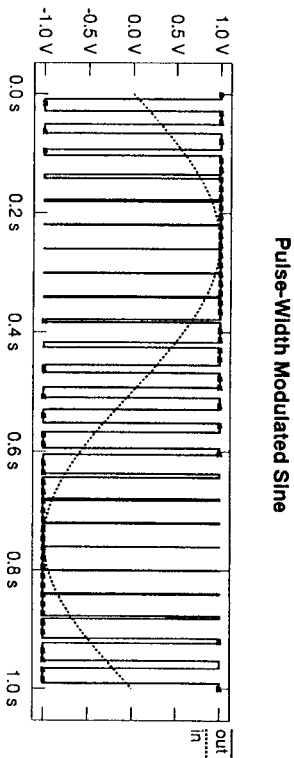


Figure 5.29: Response of a pulse-width modulator to a sinusoidal stimulus. The sample points used by a 64-point FFT are shown. Notice that many of the pulses are missed by the FFT.

sample points for a 64-point FFT are shown. The spectrum computed by the FFT is clearly inaccurate because several pulses have been missed. The accuracy of the FFT is improved by increasing the number of points used by the FFT, as shown in Figure 5.30. But even with 64K points, the accuracy is only marginal. In this example, the modulator generates only 25 pulses per period of the sine wave. If the pulse rate was higher, which would not be unusual, then the performance of the FFT would be even worse.

The Fourier integral method provided by Fourier analysis in Spectre accurately resolves the transition times without missing any pulses. It is considerably more accurate and more efficient than the FFT on this type of waveform. The Fourier coefficients computed by the Fourier integral and a 64K-point FFT for the waveform in Figure 5.29 are given in Table 5.5. The FFT is able to resolve harmonics approximately 85 dB below the fundamental whereas the Fourier integral resolves harmonics over 200 dB below the fundamental.

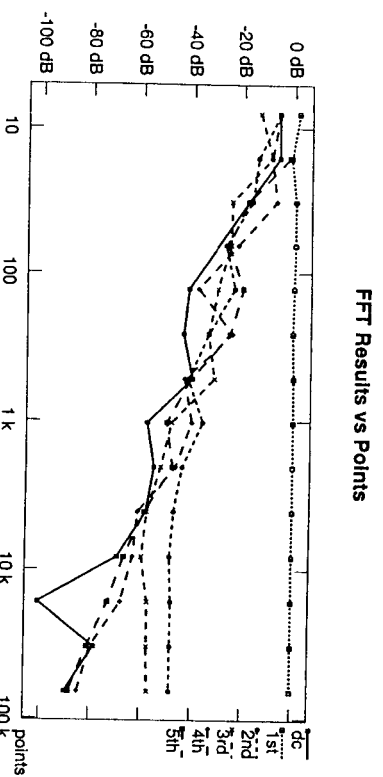


Figure 5.30: The magnitude of the Fourier coefficients computed by the FFT for the waveform in Figure 5.30 as a function of the number of points used. Even when 64k points are used, the accuracy is marginal.

5.4.4 Oscillators

Oscillators present two severe difficulties for Fourier analyzers. First, oscillators tend to take a long time to settle down after the simulation begins. And second, the period of oscillation is not known precisely in advance.

The stability of an oscillator is directly related to the Q , or quality factor, of its resonator. The higher the Q , the more stable the oscillator, and the longer it takes to settle after being perturbed in any way. Crystal and cavity oscillators, in particular, use extremely high- Q resonators to achieve very high stability and low noise. The Q of the best resonators can be as high as 1,000,000, leading to a loaded- Q in well designed oscillators of up to 100,000. A Q of 100,000 implies that the time constant of the turn-on transient is roughly 100,000 cycles of the oscillation period in length. Clearly, transient simulation of such circuit to steady-state is very painful, however steady

harmonic	Fourier Integral	64K-Point FFT
fund	0.998 V	0.998 V
2nd	-48.1 dB	-48.0 dB
3rd	-56.8 dB	-56.8 dB
4th	-93.8 dB	-84.6 dB
5th	-106 dB	-88.0 dB
6th	-138 dB	-86.2 dB
7th	-154 dB	-86.2 dB
8th	-183 dB	-87.8 dB
9th	-201 dB	-84.7 dB

Table 5.5: The magnitude of the Fourier coefficients computed by the Fourier integral and a 64K-point FFT for the waveform in Figure 5.30 on the preceding page. The Fourier integral was able to accurately resolve the -200 dB 9th harmonic whereas the FFT was unable to resolve harmonics below 85 dB.

state is required in order to apply Fourier analysis (see Figure 5.8 for an example of Fourier analysis of a sinewave that is not quite in steady-state). You can try to reduce the time required to simulate these circuits by carefully choosing the initial state of the resonator to eliminate any transients. However, in this example, it would be necessary to simulate the oscillator for 1000 cycles simply to notice a 1% difference in the signal envelope. Few have the patience or the precision to find the initial state that results immediately in steady-state when to do so requires the trial-and-error selection of initial states and where each guess requires simulating the oscillator for such a large number of cycles.

The second difficulty when Fourier analyzing the output of an oscillator is that the period of oscillation is not known precisely in advance. This requires that you run at least one transient analysis to steady-state to determine the period of oscillator, and then run another transient analysis to perform the Fourier analysis. Any change to the circuit can change the period of oscillation, requiring that the procedure be repeated.

When confronted with the need to run Fourier analysis on the output of an oscillator, try to do the best you can to get the circuit into steady-state, and then perform the Fourier analysis over hundreds (or at least tens) of periods with an external Fourier analyzer while applying a wide window function, such as the rectangular window, to determine the oscillation frequency, and a smooth window function, such as Kaiser-Bessel, to determine the distortion. The smooth window effectively suppresses the discontinuity that results from periodically extending a waveform with an approximate period.

5.4.5 Large-signal Transfer Functions

It is very common for analog designers to work with small-signal transfer functions such as gain, immittance,³ and N -port parameters. These transfer functions are always measured at a single frequency on linear (or linearized) circuits and are given as the ratio of an output to an input. Since the circuit is linear, small-signal transfer functions are easy to measure using AC analysis.

Much less common, but still very useful, are large-signal transfer functions. Large-signal transfer functions are also given as the ratio of an output to an input, but now the circuit is allowed to be nonlinear. Consider the system shown in Figure 5.31. The circuit is driven with a pure sinusoid and in steady-state responds at the input frequency and its harmonics. The large-signal transfer function is then simply the ratio of the output spectrum with respect to the fundamental of the input frequency.

$$A_k = \frac{Y_k}{X_1} \quad (5.40)$$

where Y_k and A_k are the k^{th} harmonics of the output and large-signal transfer function, and X_1 is the fundamental of the input. It is typical for the input to be nearly sinusoidal, but it is not a requirement that the signal level of the input harmonics be exactly zero.

³Immittance is an abbreviated way of referring to both impedance and admittance.

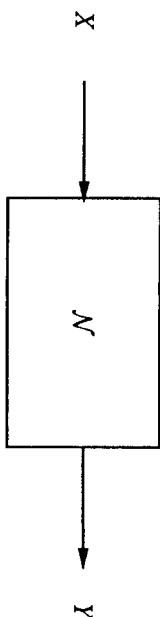


Figure 5.31: A nonlinear system for which the large-signal gain is required.

One measures a large-signal transfer function with a circuit simulator by using the Fourier analyzer to compute the spectra of both the input and the output. The ratio is computed as a post-processing step. Spectre provides a two-channel Fourier analyzer that computes spectra of both the input and the output, as well as the ratio to directly give the large-signal transfer function.

5.4.6 Clocked Analog Circuits

When computing the spectrum of clocked analog circuits such as analog/digital converters, switched-capacitor filters, or sample-and-holds it is sometimes important to sample the outputs at the end of every clock cycle and compute the spectrum of the sampled waveform rather than the spectrum of the entire waveform. If the output of the circuit drives a circuit that uses the entire waveform, such as a continuous-time filter, then the Fourier analysis should be performed on the entire output signal. However, if the output signal is sampled (for example, if a clocked ADC follows the sample-and-hold or SC filter), then it is best to compute the spectrum from the sampled waveform. The reason is that the original waveform often contains nonlinear settling phenomenon, such as slew-rate limited transitions, that are present in the original waveform, but not the sampled waveform. Once the waveform has reached the end of the clock cycle, it has settled (by design), and so the distortion from the nonlinear settling phenomenon is not present in the sampled waveform. Mea-

harmonic	Continuous	Discrete
1 st	2.34 V	4.93 V
2 nd	-75.3 dB	-84.1 dB
3 rd	-57.3 dB	-75.2 dB
4 th	-85.9 dB	-83.1 dB
5 th	-67.9 dB	-72.2 dB
6 th	-84.5 dB	-84.2 dB
7 th	-74.2 dB	-75.4 dB
8 th	-73.9 dB	-80.3 dB
9 th	-77.7 dB	-85.3 dB
THD	0.146%	0.038%

Table 5.6: The magnitude of the harmonics of the continuous- and discrete-time waveforms shown in Figure 5.32 on the next page. The difference of roughly a factor of two between the amplitude of the fundamentals of the continuous and discrete time waveforms results because the continuous time waveform has roughly a 50% duty cycle.

During the distortion of the original waveform results in artificially high levels of distortion in sampled-data applications.

Spectre provides a component that is used to model discrete-time filters by specifying its z-domain transfer function. By specifying a unity transfer function, it can be used as a sampler. It is also possible to use strobing (Section 4.4.10 on page 238) for an external FFT.

To illustrate these concepts, a MOS high-speed sample-and-hold was simulated. The sample-and-hold was clocked at 10 MHz and was driven with a 500 kHz sine wave. The output was directly applied to Spectre's Fourier analyzer. In addition, the output was sampled at the clock rate, and the output of the sampler was applied to another Fourier analyzer. The results are shown in Figure 5.32 and Table 5.6. The total harmonic distortion is about four times lower in the sampled waveform because the sampling eliminates the effects of the uninteresting distortion from slew-rate limiting.

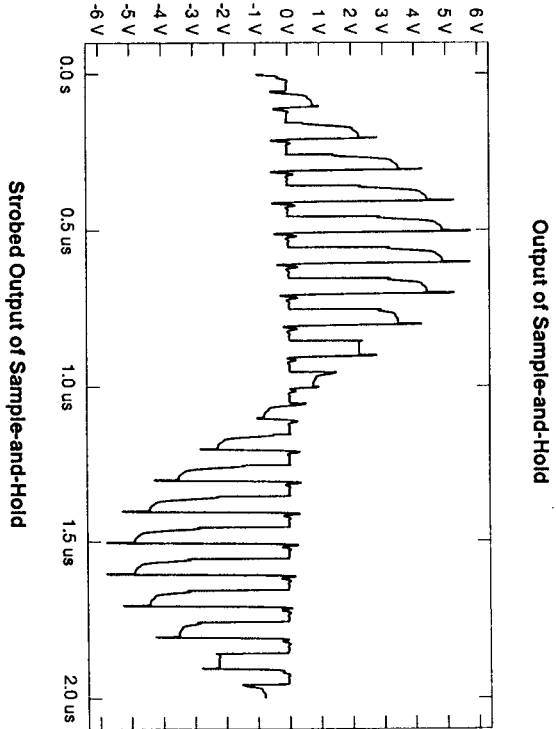


Figure 5.32: Output of sample-and-hold clocked at 10 MHz with a 500 kHz sinusoidal input. The top waveform is the output, and the bottom is the output strobed at the clock rate.

5.4.7 IMD of Narrow-Band Circuits

Finding the periodic steady-state response of a narrow-band amplifier or filter can be expensive using transient analysis because the settling time of the amplifier is usually long in comparison to the period of its center frequency. Computing quantities like intermodulation distortion, however, can be *extremely* expensive. The method used to measure distortion of a wide-band amplifier is to apply a pure sinusoid to the input, and determine by how much the output deviates from being a pure sinusoid in steady state. The harmonic distortion products in the output signal fall at frequencies that are integer multiples of the input frequency. If this same technique were applied to measure the distortion of a narrow-band amplifier, the distortion products would be attenuated because they are outside the bandwidth of the amplifier, and the calculated amount of distortion would be much too low.

Instead, distortion is measured in narrow-band amplifiers by applying two pure sinusoids with frequencies well within the bandwidth of the amplifier (call these frequencies f_1 and f_2). The harmonics of these two frequencies are outside the bandwidth of the amplifier, however there are distortion products that fall at the frequencies $2f_1 - f_2$ and $f_1 - 2f_2$. These frequencies, called the third-order intermodulation tones, should be well within the bandwidth of the amplifier and so can be used to measure accurately the intermodulation distortion produced by the amplifier.

Computing the intermodulation distortion using transient analysis requires simulation over a time interval greater than at least one period of the difference frequency, $f_1 - f_2$. If numerical integration is used, the maximum time step used must be much smaller than the minimum period of f_1 or f_2 . And as f_1 and f_2 must be close so that the third-order intermodulation tones are in the passband, the ratio of the simulation interval to the maximum usable time step is often very large.

There are two ways to measure the intermodulation distortion. One is to apply Fourier analysis directly. To avoid aliasing, use either the Fourier integral or the DFT with a large number of points. In this case, f_1 and f_2 must be commensurate. In other words, there must

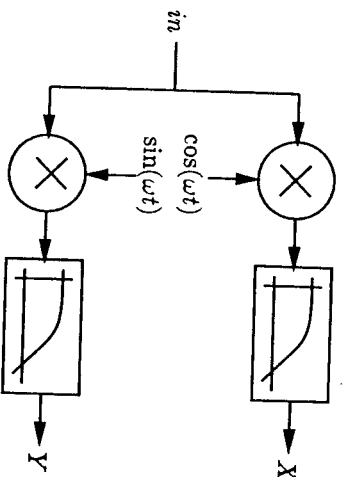


Figure 5.33: The block diagram of a vector synchronous detector.

be some frequency f such that $k_1 f = f_1$ and $k_2 f = f_2$ where k_1 and k_2 are positive integers. f is then referred to as the true fundamental and f_1 and f_2 become quasi-fundamentals. For example, if $f_1 = 14$ and $f_2 = 26$, then the frequencies are commensurate where $f = 2$, $k_1 = 7$, and $k_2 = 13$. If $f_1 = 1$ and $f_2 = \pi$, then the frequencies are not commensurate. In the case where f_1 and f_2 are commensurate, the resulting signals are periodic with period $1/f$, but k_1 and k_2 may be so large that it is prohibitively expensive to apply Fourier analysis.

The second approach is to demodulate the quasiperiodic signals before applying Fourier analysis. Depending on the approach taken, this may remove the requirement for f_1 and f_2 to be commensurate.

5.4.7.1 Demodulation using Synchronous Detection

There are various ways to demodulate, or remove the carrier or clock frequency from, a quasiperiodic waveform. For example, it is possible to build a detector from the ideal components available from the simulator. One could build a synchronous detector as shown in Figure 5.33 using the Spectre subcircuit shown in Nestlist 5.1. This

is a *vector* detector because it computes the magnitude and phase of the input signal, though it does so in rectangular coordinates.

Using a synchronous detector for demodulating quasiperiodic signals requires that you design and use a filter. The one above uses a Bessel 5-pole low-pass because of its relatively rapid settling. It was designed using standard filter tables [zverev67]. It also requires you to convert the rectangular output into magnitude and phase. Most simulation data display tools make this a simple task. However, if necessary, you could also build a little circuit to compute the magnitude for you. This circuit, shown in Netlist 5.4, computes $\sqrt{x^2 + y^2}$. This circuit is ill-conditioned when $x = y = 0$ because the slope of $\sqrt{z}$ at $z = 0$ is infinite.

An important issue with the synchronous detector is the filters give the detector a non-zero settling time and finite bandwidth. For satisfying results when using synchronous detection and Fourier analysis, it is necessary that the modulation frequency and its harmonics be well within the pass-band of the detector's filters (to assure accurate measurement of distortion), and it is necessary for the carrier frequency to be well outside of the filter's pass-band to prevent the carrier frequency from reaching the Fourier analyzer, which would contaminate the measurement through aliasing or spectral leakage. The synchronous detector's filters can be problematic when used with Fourier analysis because of their long settling time. One must simulate the circuit until the circuit and the filters in the synchronous detector have completely settled before performing the Fourier analysis.

The advantage of using a synchronous detector is that if the filter completely rejects the carrier, then there is no requirement for the carrier and the modulation frequencies to be commensurate because the Fourier analyzer never sees the carrier.

The response of the synchronous detector to a pulsed carrier wave and an amplitude modulated sine wave are shown in Figures 5.34 and 5.35. These measurements were made with Netlist 5.5.

```
//
// VECTOR SYNCHRONOUS DETECTOR
//
// Output is in rectangular form
//
include "multiplier"
include "filter"
simulator lang=spectre

subckt synchDetector ( x y in )
parameters freq bw

// Quadrature sources
Vxlo ( Xlo 0 ) vsource type=sine \
    freq=freq ampl=1 sinephase=90
Vylo ( Ylo 0 ) vsource type=sine \
    freq=freq ampl=1 sinephase=0

// Multipliers
MultX ( Xif Xlo in ) multiplier
MultY ( Yif Ylo in ) multiplier

// Filters
FiltX ( x Xif ) mfedFilter bw=bw gain=2
FiltY ( y Yif ) mfedFilter bw=bw gain=2
ends synchDetector
```

Netlist 5.1: Subcircuit that implements a vector synchronous detector. Output is in rectangular form. The component pieces are shown in Netlists 5.2 and 5.3 on the next page.

```

//
// MULTIPLIER
//
simulator lang=spectre

subckt multiplier ( prod in1 in2 )
  Mult ( prod 0 in1 0 in2 0 ) pvcvs coeffs=[0 0 0 0 1]
ends multiplier

```

Netlist 5.2: Subcircuit that implements a multiplier. It is used by the synchronous detector. The polynomial-controlled source implements $V_{\text{prod}} = V_{\text{in1}} V_{\text{in2}}$.

```

//
// FILTER
//
// Implements 5th order MFED (Bessel) active filter
//
simulator lang=spectre

```

```

subckt mfedFilter ( out in )
  parameters bw=1 r0=1 gain=1
  G1n (fin 0 in 0) vcvs gm=-2*gain
  R1n (fin 0 ) resistor r=r0
  C1 (fin 0 ) capacitor c=0.1743/(r0*6.28319*bw)
  L2 (fin mid) inductor l=0.5072*r0/(6.28319*bw)
  C3 (mid 0 ) capacitor c=0.8040/(r0*6.28319*bw)
  L4 (mid out) inductor l=1.110*r0/(6.28319*bw)
  C5 (out 0 ) capacitor c=2.2582/(r0*6.28319*bw)
  Rout (out 0 ) resistor r=r0
ends mfedFilter

```

Netlist 5.3: Subcircuit that implements a 5th order maximally flat delay lowpass filter. It is used by the synchronous detector.

```

//
// MAGNITUDE
//
// Implements z = sqrt( x^2 + y^2 )
//
simulator lang=spectre

subckt magnitude( out x y )
  Mag (z 0 x 0 y 0 z 0) pvcvs \
    coeffs=[0 0 0 0 -1 0 0 -1 0 1]
  R1ng (z 0) resistor r=1T
  // R1ng removes singularity when x = y = 0
  Buf (out 0 z 0) vcvs gain=1
  R1umny (out 0) resistor
  // infinite resistor to avoid topology warnings
ends magnitude

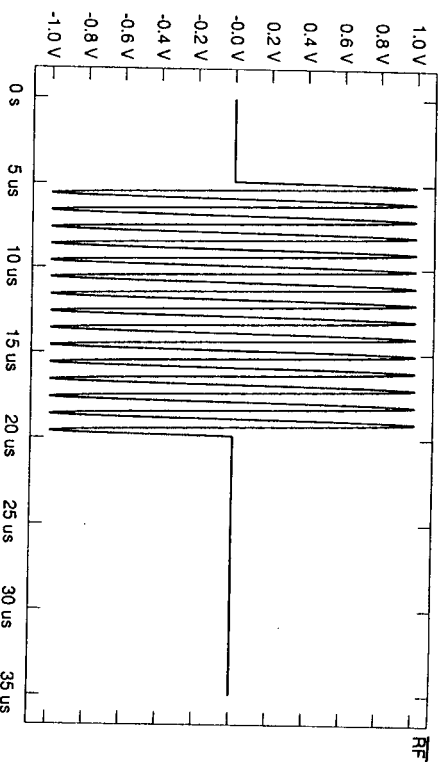
```

Netlist 5.4: Subcircuit that calculates the magnitude of a 2-dimensional vector. This circuit is ill-conditioned when $V_x = 0$ and $V_y = 0$. The polynomial-controlled source implements $I_z = -V_x^2 - V_y^2 + V_z^2$. Neglecting the small current through R1ng results in $I_z \approx 0$ and $V_z \approx \sqrt{V_x^2 + V_y^2}$.

5.4.7.2 Demodulation using Sampling

Another approach to demodulating quasiperiodic signals, which is better adapted to use with Fourier analysis, involves sampling the waveforms at the carrier or clock frequency. The sampling effectively removes the carrier frequency. For best performance, the sampler should force the simulator to place a time point at exactly the same time that the waveform is sampled. A sampler can be constructed using Spectre's z-domain transfer-function blocks as shown in Netlist 5.6. Unfortunately, many simulators do not provide a sampler or its equivalent, so implementing a sampling detector is extremely difficult.

Pulsed Carrier Wave



Pulse Response of a Synchronous Detector

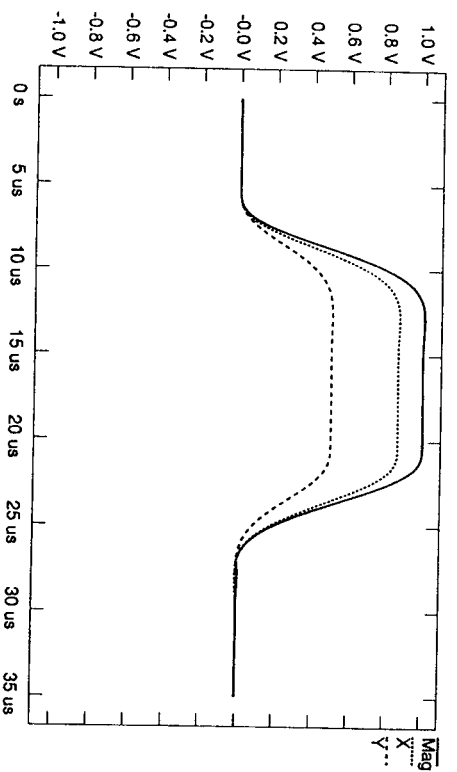
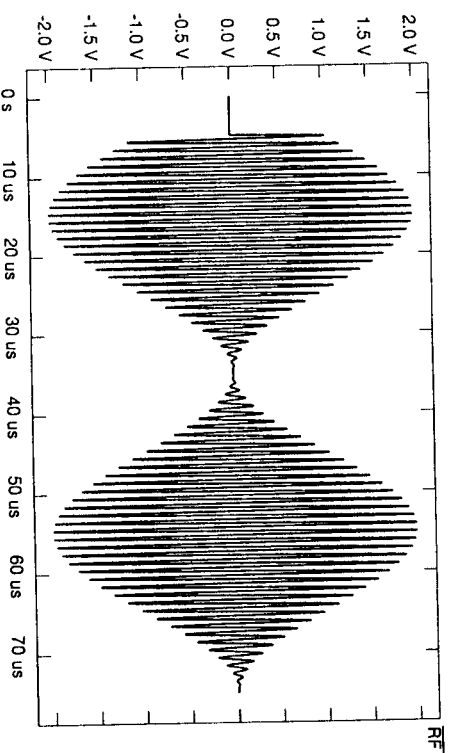


Figure 5.34: Response of a synchronous detector to a pulsed carrier wave. Notice the non-zero delay and settling time.

Amplitude Modulated Carrier Wave



Vector Demodulation with a Synchronous Detector

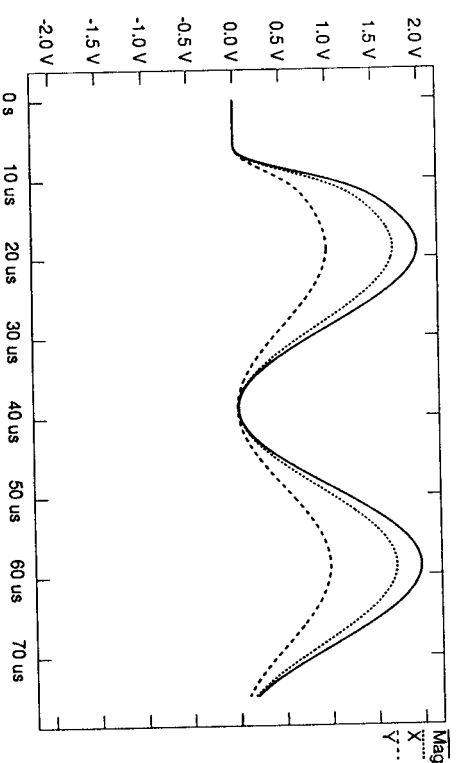


Figure 5.35: Response of a synchronous detector to an amplitude modulated carrier wave.

```
// Example that uses the synchronous detector
include "synch-detector"
include "magnitude"
simulator lang=spectre

// Generate pulsed sine wave
Vin ( mid 0 ) vsource type=sine delay=5us \
    ampl=1 freq=1MHz sinephase=60 \
    ammodindex=0 ammodfreq=25kHz
VinOff ( in mid ) vsource type=sine delay=20us \
    ampl=-1 freq=1MHz sinephase=60

// Synchronous Detector
Synch ( X Y in ) synchDetector freq=1MHz bw=100kHz
MagSynch ( Mag X Y ) magnitude

// PulseResponse
PulseResponse tran stop=35us

// Sine Response
DisableVinOff alter dev=VinOff param=type value=dc
ModulateVin alter dev=Vin param=ammodindex value=1
SineResponse tran stop=100us
```

Netlist 5.5: Example circuit that uses the vector synchronous detector.

```
//
// VECTOR SAMPLING DETECTOR
//
// Output is in rectangular form
//
simulator lang=spectre

subckt smplDetector ( x y in )
    parameters freq
    SmpLx x 0 in 0 zvcvs ts=1/freq
    Smply y 0 in 0 zvcvs ts=1/freq td=0.25/freq
ends smplDetector
```

Netlist 5.6: Subcircuit that implements a vector sampling detector. Output is in rectangular form. This subcircuit simply contains two samplers with the sample time of the second offset by 90 degrees.

The response of the sampling detector to a pulsed carrier wave and an amplitude modulated sine wave are shown in Figures 5.36 and 5.37. These measurements were made with Netlist 5.7.

Notice that the sampling detector does not suffer from settling time or bandwidth problems, making it more suitable for use with the Fourier analysis. It also eliminated the constraint that the carrier frequency be well above the modulation frequency. The sampling detector naturally discretizes time, making it ideal for use with the DFT. However, in the process of discretizing time, large portions of the waveforms are ignored. Any important phenomena that occurs only in between sample points is also ignored. Sometimes this behavior is desirable, as with a waveform that is eventually sampled in the circuit, but not always.

In order to apply Fourier analysis to the output of the sampling detector, the analysis period must be an integral multiple of the sampling interval. Since the sampling rate must equal the carrier frequency,

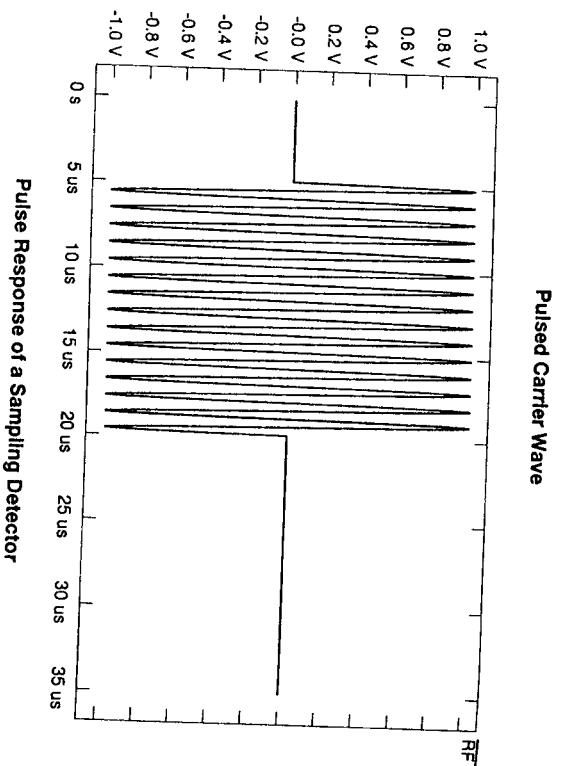


Figure 5.36: Response of a sampling detector to a pulsed carrier wave. Noticed the lack of delay and settling behavior. Strobing is used to eliminate the delay between X and Y .

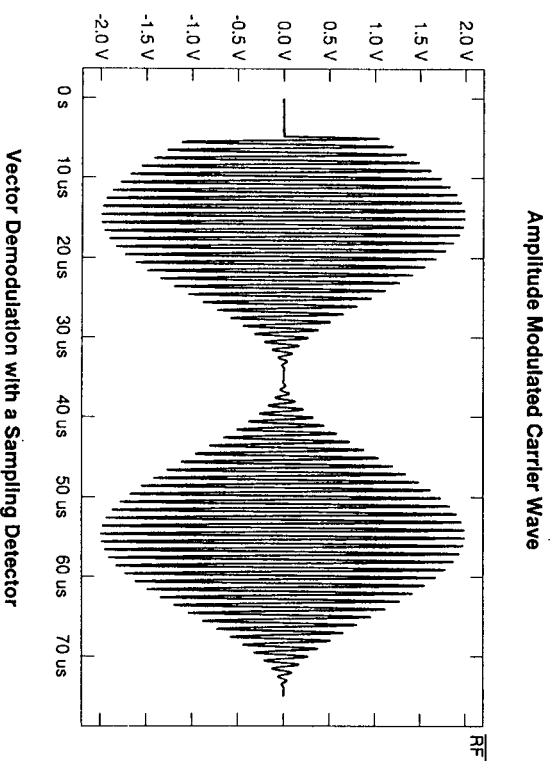


Figure 5.37: Response of a sampling detector to amplitude modulated carrier wave. Strobing is used to eliminate the stair-case appearance of the waveforms.

```
// Example that uses the sampling detector
include "smp1-detector"
include "magnitude"
simulator lang=spectre

// Generate pulsed sine wave
vin ( mid 0 ) vsource type=sine delay=5us \
    ampl=1 freq=1MHz sinephase=60 \
    ammodindex=0 ammodfreq=25kHz
vinoff ( in mid ) vsource type=sine delay=20us \
    ampl=-1 freq=1MHz sinephase=60

// Sampling Detector
smp1 ( X Y in ) smp1Detector freq=1MHz
MagSmp1 ( Mag X Y ) magnitude

// PulseResponse
PulseResponse tran stop=35us

// Sine Response
DisableVinOff alter dev=VinOff param=type value=dc
ModulateVin alter dev=Vin param=ammodindex value=1
SineResponse tran stop=100us
```

Netlist 5.7: Example circuit that uses the vector sampling detector.

the modulation and carrier frequencies must be commensurate.⁴

5.4.8 Distortion of a Mixer

This section discusses the measurement of distortion from a mixer by direct application of the Fourier analysis to illustrate several impor-

⁴It is possible to use APFT to avoid the constraint that the modulation and carrier signals be co-periodic [Kundert90]. The ill-conditioning problem alluded to earlier do not occur when using equally spaced points to transform a periodic signal as long as the points roughly span on full period.

tant issues. It is sometimes preferable to use the techniques presented in the previous section to demodulate the output of the mixer before applying Fourier analysis, but that is not discussed further.

A mixer is a nonlinear circuit commonly used in communications to translate a signal from one frequency to another. A mixer has two inputs, one usually referred to as the RF (for radio frequency), which is to be translated, and the other input, the LO (for local oscillator), which performs the translation. The output predominantly contains two signals, one at the sum of the RF and LO frequencies, and one at their difference. Usually only one of the signals is desired, and so the mixer is followed by a filter.

Mixers are very difficult to simulate for two reasons. First, the frequencies of the signals present can be very widely separated. Second, the settling time of the output filter can be very much longer than the period of the lowest frequency present in the mixer. Consider the down conversion mixer in the HP8505 network analyzer. The mixer has an input RF frequency that ranges from 500kHz to 1.3GHz and an LO frequency that is always offset from the RF by 100kHz. The desired output frequency is 100kHz, and the output is fed directly into a high- Q low-pass filter to assure that this is the only signal present at the output. Simulating this circuit is extremely difficult because the ratio of the input to output frequency can be as high as 13,000 to 1, and because the output filter has a long settling time. Transient simulation of this circuit requires time steps much less than 1 ns to capture the high frequencies, and a simulation interval of at least 100μ s to capture ten periods of the filter output — a minimum of 10^5 time points are needed.

Simulating a mixer in the time-domain is made easier by deleting the output filter. With the output filter gone, its long settling time is no longer a concern. However, eliminating the filter also results in very high frequencies being present in the output. So you must either use a Fourier analysis approach that is not subject to aliasing, such as Spectre's Fourier integral, or sample the waveform at a sufficiently high rate to avoid aliasing. One other concern when deleting the output filter is that the loading of the mixer is changed, which may change the results.

In the $\Sigma\Delta$ modulator example of Section 5.4.2, it was possible, and generally desirable, to use the FFT because the desired spectrum could be computed from a uniform sampling of the output with one sample per clock period (it was possible and desirable to ignore the details of the waveform over individual clock cycles). However, uniform sampling is undesirable when trying to compute the distortion of a mixer. Consider the example of a mixer from a hypothetical satellite receiver for video signals. The RF input frequency is 2.7GHz and the IF output frequency is 500MHz. To determine the distortion of the mixer, the input is modulated at 1MHz. The modulation frequency was chosen to satisfy the following constraints:

1. The significant intermodulation distortion products should be well within the 10MHz bandwidth of the IF filter of the mixer to be meaningful.
2. All waveforms must be periodic (a condition that must be satisfied in order to apply the Fourier analysis).
3. The product of the simulation interval (the reciprocal of the modulation frequency, plus any needed settling time) and the highest significant frequency present (some harmonic of the RF input) should be minimized in order to minimize the time required to complete the simulation.

The IF frequency is the 500th harmonic of the modulation frequency. Spectre is requested to compute the five harmonics above and below the IF. The computation is still efficient, because the Fourier coefficients are being computed for only 11 harmonics. Of course, the simulation is slow because small time steps are needed to accurately follow the high frequencies and because of the need to simulate for at least one period of the modulation frequency. However, it is still faster than if the FFT were used because the output waveform has sharp transitions, as shown in Figure 5.38, that require a very small sampling interval. The minimum step size needed during the simulation was six times smaller than the average step size, implying that to get comparable accuracy with the FFT would require six times as many time points and so is six times as slow. The results are shown in Table 5.7.

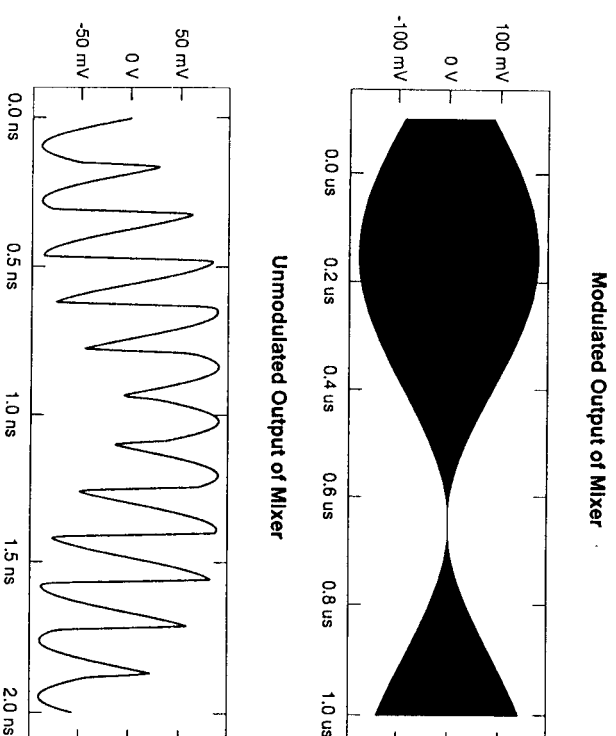


Figure 5.38: The modulated (top) and unmodulated (bottom) output of the mixer.

5.4.8.1 Errors Due to Nulls

The output of a mixer is shown in the top of Figure 5.38. Notice that the waveform undergoes a null during every period of the modulation signal. This is typical of mixers. Because the signals are so much smaller during the null than at other times, the simulator effectively loosens tolerances during the null. The reason for this is that the reference for `reltol` is equal to the value of the signal over all past time (see Section 4.3.2.1 on page 183). During the null, the signal may be smaller than `reltol` times the reference, essentially disabling all error control. The sloppy time-step control during this period can result in significant aliasing and therefore error in the Fourier analysis. When applying Fourier analysis to signals with deep nulls,

harmonic	reltol = 10^{-3}	reltol = 10^{-6}
495th	-56.5 dB	-95.4 dB
496th	-81.7 dB	-93.1 dB
497th	-60.0 dB	-68.2 dB
498th	-52.1 dB	-53.3 dB
499th	-6.06 dB	-6.05 dB
500th	56.2 mV	56.3 mV
501th	-6.05 dB	-6.05 dB
502th	-57.7 dB	-53.3 dB
503th	-63.9 dB	-68.4 dB
504th	-77.4 dB	-98.1 dB
505th	-56.8 dB	-101 dB

Table 5.7: The magnitude of the harmonics needed to determine intermodulation distortion of a mixer calculated using the Fourier integral.

it is a good idea to set the maximum time step to assure at least 10-20 time points per period of the carrier.

5.5 Summary

This is a list of the key points presented in this chapter along with the section and page numbers where they were presented.

- Harmonic distortion is a useful measure for wide-band circuits. However, the harmonics are attenuated in narrow-band circuit making harmonic distortion useless as a measure of distortion. Distortion of narrow-band circuits is given in terms of intermodulation distortion. Sect. 5.1.1.1 on page 252.
- A quasiperiodic signal can approximated arbitrarily closely by periodic signal. However, periodic signals that result from multiple quasi-fundamentals often contain frequencies that are so high compared to the required Fourier analysis interval that it

is impractical to apply Fourier analysis directly. In this case, some form of demodulation should be used. Sect. 5.2.1.2 on page 265.

- There are five important sources of error in a Fourier analysis: incorrect period, transients, aliasing, interpolation, and simulation noise. Sect. 5.3.1 on page 267.

- With both SPICE2 and SPICE3 you must assure that the number of time points computed by the simulator in the Fourier analysis interval as well as the number of samples used in the DFT are sufficient to avoid aliasing and interpolation. Sect. 5.3.2 on page 280.

- Spectre's Fourier analyzer uses the Fourier integral rather than the DFT, and so is not subject to aliasing. It also does not require equally-spaced points, but does give better answers if the maximum time step small enough to result in equally-spaced points. Sect. 5.3.3 on page 282.

- Window functions can be applied before Fourier analysis to reduce the affect of discontinuities in the waveform. Use window functions with external FFT when Fourier analyzing non-periodic signals and signals with unknown periods (such as those generated by oscillators) Sect. 5.3.4.1 on page 290.

- Wide window functions generally give better frequency resolution in general, whereas smooth window functions generally give better amplitude resolution. You can gain much more amplitude resolution with a smooth window than you can frequency resolution with a wide window. So in circuit simulators, it is generally best to use a smooth window for best amplitude resolution and a longer Fourier analysis interval to get the needed frequency resolution. Sect. 5.3.4.1 on page 298.

- Even small errors in the period result in discontinuities in the periodically extended signal. Use external Fourier analyzer with windowing to reduce the error that results from the discontinuities. Sect. 5.4.4 on page 312.

- If the output of your circuit feeds a sampled-data system, then it is best to sample the signal before applying Fourier analysis. Sect. 5.4.6 on page 313.

Appendix A

Simulator Options

A.1 Introduction

This appendix describes simulator options that affect accuracy or convergence.

A.2 SPICE Options

A.2.1 Global Options

SPICE expects the global options to be given on the .options statement.

A.2.1.1 Abstol

The absolute tolerance for currents. Defines the smallest interesting current anywhere in the circuit. Currents smaller than `abstol` are ignored when checking for convergence, and when choosing the time step. `abstol` is a supplement to `reltol` that plays a role when the simulator is checking the accuracies of very small currents. `abstol` prevents the simulator from attempting to converge femtoampere signals to attoampere levels.

As a rule, an absolute tolerance such as `abstol` should be set 10^6 – 10^8 times smaller than the largest signal of the same type present

in the circuit. This ratio could be even greater on sensitive circuits. Typically, the largest current present in analog integrated circuits is in the $1\mu\text{A}$ – $100\mu\text{A}$ range, which is why `abstol` defaults to 1 pA. It should be set higher for power electronic circuits. Setting `abstol` too loose results in degraded accuracy. Setting `abstol` too tight prevents the simulator from converging.

A.2.1.2 Gmin

A very small conductance added across nonlinear devices to prevent nodes from floating if a device is turned completely off. By default, `gmin`= 10^{-12}V . It must be positive, though Spectre allows it to be zero. The manner in which SPICE and Spectre add the `gmin` conductors to the various nonlinear devices is different and is shown in Figures 2.13 and 2.14 on page 41.

If `gmin` is too large it adversely affects accuracy. If it is too small it may adversely affect convergence. Be advised that if `gmin` is large enough to positively affect convergence, it is also large enough to negatively affect accuracy.

A.2.1.3 Limpts

Maximum number of plot points that can be plotted or printed during an AC, DC, or transient analysis. The default value is 201. This nuisance parameter was helpful when boards of undergraduates were all competing for the same line printer in the basement of Evan's Hall back in the 70's, but it has little value now. Feel free to set it to a value that is as large as you need. In SPICE2, setting this value too high wastes a precious piece of the fixed amount of memory available.

A.2.1.4 Pivrel

The relative threshold used for selecting pivots when factoring the Jacobian matrix. Big pivots are good because they reduce the likelihood of error building up while factoring. However, insisting on having the largest possible pivots can result in the sparse Jacobian

matrix filling-in during the factorization, which would result in the simulation running much more slowly and requiring more memory. `pivrel` specifies how large an entry in the Jacobian has to be in order to be a pivot candidate. It is defined as the minimum acceptable ratio between the absolute value of a pivot and the largest remaining element in the same column.

In rare cases, increasing the value of `pivrel` solves a convergence problem, but more often it just causes the simulator to run more slowly and gives no benefit. Pinning your hopes on `pivrel` is clearly a desperation move. `pivrel` must be specified between 0 and 1, with the default value being 10^{-3} . Reasonable values range between 10^{-12} to 0.5.

A.2.1.5 Pivtol

The minimum absolute value allowed for a Jacobian entry to be considered as a pivot. Default value is 10^{-13} .

Most likely this parameter is settable simply because the original developers were not sure what value to use and wanted to be able to adjust it if necessary. Fortunately, they chose wisely. It should never need adjusting.

A.2.1.6 Reltol

The universal accuracy control. Give value between 0 and 1, values closer to zero imply greater accuracy. `reltol` directly affects the Newton convergence criteria and the time-step control algorithm. It specifies the upper limit on errors relative to the size of the signals present. The default value is 0.1% and typical values range from 10^{-6} to 10^{-2} .

Reducing `reltol` decreases the error in the results computed by the simulator, however no level of accuracy is guaranteed. Nor is any particular level of accuracy implied from a given value for `reltol`. In particular, setting `reltol` to 0.1% in no way implies that the accuracy attained by the simulator is 0.1%.

A.2.1.7 Vntol

The absolute tolerance for voltage. Defines the smallest interesting voltage anywhere in the circuit. Voltages smaller than `vntol` are ignored when checking for convergence, and when choosing the time step. `vntol` is a supplement to `reltol` that plays a role when the simulator is checking the accuracies of very small voltages. `vntol` prevents the simulator from attempting to converge nanovolt signals to picovolt levels.

Absolute tolerances, such as `vntol`, should be set 10^6 – 10^8 times smaller than the largest signal of the same type present in the circuit. This ratio could be even greater on sensitive circuits. Typically, the largest voltage present in analog integrated circuits is in the order of 10 V, which is why `vntol` defaults to 1 μ V. It should be set higher for high voltage circuits. Setting `vntol` too loose results in degraded accuracy. Setting `vntol` too tight can prevent the simulator from converging, though this problem is not as severe as it is with `abstol`.

A.2.2 DC Analysis Options

SPICE expects the DC analysis options to be given on the `.options` statement.

A.2.2.1 Itl1

Maximum number of iterations for a DC operating point analysis. The default value is 100. Occasionally you can get SPICE to converge by increasing this value. The likelihood of convergence stops increasing above 1000, with larger values simply resulting in SPICE taking longer to fail.

A.2.2.2 Itl2

Maximum number of iterations per step in a DC sweep. The default value is 50. Set this to a larger value if SPICE has convergence difficulties in the middle of a DC sweep. Alternatively, you can try

A.2. SPICE Options

taking smaller steps. Again, do not bother with values over 1000, which just delay the 'no convergence' message.

A.2.2.3 Itl6

Number of steps to take in when source stepping. The default value is 0, which disables source stepping. Larger values imply higher likelihood of convergence, but longer waits.

A.2.3 Transient Analysis Options

SPICE expects the transient analysis options to be given on the `.options` statement (except as noted).

A.2.3.1 Chgtol

`chgtol` is the absolute tolerance for the state variable in the LTE criterion. Thus, `chgtol` is the absolute tolerance for charge through capacitors and flux across inductors. `chgtol` limits how closely LTE is controlled when the time step is small and when the charge through the capacitor or flux across the inductor is small. Unlike `abstol`, `chgtol` is multiplied by `reltol` in the LTE criterion. Thus, capacitor charges must be less than `reltol` $\times$ `chgtol` before they are dominated by `chgtol`.

A.2.3.2 Itl3

Sets the lower transient analysis iteration limit. The default value is 4. It is difficult to conceive of a situation that calls for adjusting the value of this parameter.

A.2.3.3 Itl4

Sets the upper per step iteration limit for transient analysis. The default value is 10. Increasing this value aids convergence on some circuits, particularly on circuits that exhibit strongly discontinuous

behavior, like some macromodels. On such circuits, increasing the value could speed up the analysis considerably. On other circuits, it could cause the analysis to run slower.

The recommended upper limit is 50 unless SPICE is failing to converge in transient, in which case 500 might be better.

A.2.3.4 Itl5

Upper limit on the total number of iterations during a transient analysis. The default value is 5000. Setting the value to 0 disables the limit. This is another parameter than made sense when there was large number of users, all simulating opamps, and all competing for a single mainframe. It is a good habit to always set this limit to 0. Who wants to wait for a long transient simulation, only to have it unexpectedly terminate prematurely due to this limit.

A.2.3.5 Lvtim

This flag specifies the time-step control algorithm. If 1, SPICE uses the iteration-count time-step control, and if 2 it uses truncation-error time-step control. Briefly put, controlling truncation error is very important. If the truncation error is not properly controlled, the time-constants computed by the simulator could be meaningless. Iteration-count time-step control ignores completely the truncation error and so is capable of computing incorrect results. Always use $Lvtim=2$ unless you do not mind if the simulator produces results in which the time-constants are way off. See Section 4.3.1.2 on page 179 for more information.

A.2.3.6 Maxord

The maximum integration method order for the backward difference method (Gear's method). Default and minimum value is 2. Maximum value is 6. Higher order methods should be more efficient when very high accuracy is needed and the signals are very smooth. However, they can also be unstable on lightly damped circuits. I rec-

ommend leaving this parameter at its default value of 2 (the higher-order methods have never been heavily used and there are rumored to be some problems in SPICE's implementation).

A.2.3.7 Method

Integration method, choose between **gear** and **trapezoidal**. The default value is **trapezoidal**. Select **gear** if trapezoidal ringing is a problem. Select **trapezoidal** is simulating highly underdamped circuits. Otherwise, you are generally free to choose either. Trapezoidal rule is usually a bit more efficient. See Section 4.2.1 on page 131 for more information.

A.2.3.8 Tmax

T_{max} is the optional fourth argument on the transient analysis statement. It places an upper bound on the time step. If T_{max} is not specified, SPICE sets it to

$$T_{max} = \frac{T_{step} - T_{start}}{50}. \quad (A.1)$$

In addition, SPICE reduces T_{max} to T_{step} (the second argument on the transient analysis statement) if there are no energy-storage elements such as capacitor or inductor present in the circuit. Finally, SPICE bounds T_{max} to be no larger than half the transmission delay of the shortest transmission line. Very short transmission lines significantly slow a SPICE transient analyses.

A.2.3.9 Tstep

T_{step} is the printing and plotting increment. It is also the suggested time step. T_{step} also has an effect on the number of points used when performing Fourier analysis. See Section 5.3.2 on page 278 for more information.

A.2.3.10 **Ttrel**

ttrel is only used in the LTE criterion where it multiplies **reltol**. Its primary function is to allow control of the LTE criterion independent of the Newton convergence criteria. It is not a good idea to tighten the LTE criterion without also tightening the Newton convergence criteria because it can cause the simulator to take very small time steps trying to follow the numerical noise generated by errors allowed by the Newton criteria. Thus, one should never reduce **ttrel** much below its default value of 7. There is benefit to loosening the LTE criterion relative to the Newton criteria. For example, if you were less concerned about accurately depicting time constants than equilibrium points, you might want to loosen the LTE criterion relative to the Newton criteria by increasing **ttrel**. Similarly, if charge conservation is important, you might want to tighten the Newton criteria without affecting the LTE criterion. This allows you to improve charge conservation without slowing the simulation down by much. In this case, tighten **reltol** and loosen **ttrel** by the same amount.

A.2.3.11 **Uic**

When **uic** is specified on the transient analysis statement, it directs SPICE to skip any attempt to perform an initial DC or IC solve for a transient analysis. With **uic**, all nodes for which initial conditions are not specified start at 0 V. One important use for this option is to allow people to use transient analysis even though the initial condition calculations do not converge. One can even continue transient analysis until steady-state is reached and save the final results for use as a node-set. For more information, see Section 4.3.5.1 on page 196.

A.3 Spectre Options

A.3.1 Global Options

Spectre expects global analysis options to be given on the options or set statements. Type 'spectre -help options' for a full list of Spectre's options.

A.3. Spectre Options

A.3.1.1 **Approx**

When set, this flag allows Spectre to make small approximations in the component model equations that allow the simulation to run faster. The errors made when using this option should be very small. If you are a nervous type, disable this to reassure yourself that the approximations are reasonable, then turn it back on.

A.3.1.2 **Diagnose**

This is a flag that when set causes additional diagnostics to be printed. It is not enabled by default because some the diagnostics may be costly to compile or may be a nuisance in normal circumstances. When enabled, Spectre warns about unusual growth in maximum signal levels, which might indicate the presence of unstable time constants. It also produces counts of the number of times each node prevents convergence. This information helps localize convergence problems to a particular portion of the circuit.

A.3.1.3 **Gmin**

A very small conductance added across nonlinear devices to prevent nodes from floating if a device is turned completely off. By default, **gmin**= 10^{-12} V. It must be positive, though Spectre allows it to be zero. The manner in which SPICE and Spectre add the **gmin** conductors to the various nonlinear devices is different and is shown in Figures 2.13 and 2.14 on page 41.

If **gmin** is too large it adversely affects accuracy. If it is too small it may adversely affect convergence. Be advised that if **gmin** is large enough to positively affect convergence, it is also large enough to negatively affect accuracy.

A.3.1.4 **Homotopy**

Specifies the homotopy (continuation method) used when computing DC operating points and initial conditions. The available choices are none, **gmin** (**gmin** stepping), **source** (**source** stepping), **ptran**

(pseudo-transient), `dptran` (damped pseudo-transient), and `all`, the default value. Specifying a different value allows you to skip methods that always fail for a particular circuit. Only change the value from `all` when experience with a particular circuit shows that there is at least one reliable method.

A.3.1.5 `labstol`

The absolute tolerance for currents. Spectre's `labstol` is the same as SPICE's `abstol`. See the description for `abstol` in Section A.2.1.1 on page 335 for more information.

A.3.1.6 `Limit`

Specifies the limiting algorithm to be used to help Newton's method converge. Possible values are `dev`, which specifies that limiting is performed at the device, `delta`, specifies a node based limiting, and `log`, which specifies a node-based logarithmic limiting. The default is `dev`. `delta` has been found to work well on MOS circuits, however both node-based approaches are very slow on circuits that exhibit high voltages. Occasionally, one can get Spectre to converge much faster by playing with the value of this parameter, but in generally it is rarely worth the time.

A.3.1.7 `Macromodels`

Over the years Spectre has been tuned to run quickly on real circuits. These optimizations have the side effect of causing Spectre to run more slowly on circuits that deviate greatly from the norms of physical circuits. In particular, Spectre performs poorly with some macromodels. Setting this flag makes Spectre more tolerant of the aberrant behavior exhibited by these macromodels. With `macromodels` set, Spectre becomes more determined to efficiently pass through the abrupt discontinuities exhibited by some macromodels and uses local error criteria to allow huge voltages and currents at a subset of the nodes.

A.3. `Spectre Options`

By default `macromodels=no` because otherwise Spectre runs more slowly on most circuits, but enabling it may cause Spectre to run much quicker on circuits that exhibit non-physical behavior.

A.3.1.8 `Opptcheck`

This flag specifies whether Spectre should check the operating point parameters against their soft limits after the each analysis. Soft limits are ranges that can be specified for any component parameter that when violated causes Spectre to issue a warning. A file containing a set of soft limits tailored for IC design is provided with Spectre, but you are free to use your own files or customize the ones provided. By default, `opptcheck=yes`. However, it is not useful unless you provide Spectre with a set of soft limits (use `+param` argument on the Spectre command line or in the `SPECTRE_DEFAULTS` environment variable).

It is highly recommended that you use this feature because it can alert you to subtle problems in your circuit. With a little effort developing your own soft limit files, Spectre will automatically warn you of the following situations and more:

1. Transistors that are the wrong type (NPNs versus PNPs, N-type versus P-type).
2. Excessive substrate currents caused by incorrectly wiring the circuit or unexpected operating conditions.
3. Excessive device currents, voltages or powers.
4. Unexpected operating conditions that result in unacceptably poor g_m or f_T .
5. Incorrectly specified parameter values, such a width or length mistakenly given in microns rather than meters.

A.3.1.9 `Pivabs`

The minimum absolute value allowed for a Jacobian entry to be considered as a pivot. Default value is 0.

A.3.1.10 Pivotdc

This flag turns on numerical pivoting throughout the DC analysis. Normally numerical pivoting is used only once at the beginning of DC analysis and the matrix is not reordered unless it is clearly needed. Occasionally convergence is improved by allowing Spectre to reorder the matrix every time it is factored during a DC analysis. It is rare for this option to help convergence, but when it does it is usually when simulating a circuit with extraordinary gain, such as a string of inverters or amplifiers. However, enabling this option can result in the DC analysis running much more slowly, especially on large circuits. By default, this option is disabled.

A.3.1.11 Pivrel

The relative threshold used for selecting pivots when factoring the Jacobian matrix. See the description of `pivrel` in Section A.2.1.4 on page 336 for more information.

A.3.1.12 Reltol

The universal accuracy control. See the description of `reltol` in Section A.2.1.6 on page 337 for more information.

A.3.1.13 Rforce

`rforce` is the value used by Spectre when imposing nodesets, forces, and initial conditions. Spectre imposes nodesets, forces, and initial conditions on nodes by attaching the series combination of a voltage source and a resistor whose value is `rforce` times a local multiplier. The multiplier is unity for free nodes, but may be different for branches and nodes internal to components. For example, you are allowed to also specify a local `rforce` multiplier on inductors. The actual value used is the product of the local and global values.

`rforce` is used adjust the forcing resistor's value to overcome small resistors in the circuit. The default value is 1, but you can make it

smaller if small resistors in your circuit are causing your nodesets, forces, and initial conditions to be inaccurate.

A.3.1.14 Vabstol

The absolute tolerance for voltage. Spectre's `vabstol` is the same as SPICE's `vtol`. See the description of `vtol` in Section A.2.1.7 on page 338 for further information.

A.3.2 DC Analysis Options

Spectre expects the DC analysis options to be given on the DC analysis statement (except as noted). Type '`spectre -help dc`' for a full list of Spectre's DC analysis parameters.

A.3.2.1 Check

Specifies that Spectre should check the operating point parameters against the soft limits for the final value of the DC analysis. See the description for `optcheck` in Section A.3.1.8 on page 345 for more information about soft limits.

A.3.2.2 Force

A 'force' is a new concept in Spectre. It provides DC analysis something analogous to transient analysis's 'initial condition'. It is used to force node voltages or branch currents to some prespecified value. Unlike with nodesets, the resulting solution is not an equilibrium point. It is occasionally useful to force node voltages or branch currents in order to give the circuit an isolated solution.

Use the `force` option to specify the source for the forced values. If `force=none`, which is the default, then no values are forced at the nodes and branches. When `force=node`, then values specified on initial condition statements are forced. When `force=dev`, the values specified as initial condition parameters on various components are used. Finally, if `force=all`, then the initial conditions provided on

both initial condition statements and as initial condition parameters are used. If you specify a 'force' file with the **readforce** parameter, force values read from the file are used in lieu of any 'ic' statements.

A.3.2.3 Homotopy

Specifies the homotopy (continuation method) used when computing DC operating points and initial conditions. See the description of homotopy parameter in Section A.3.1.4 on page 343 for further information. Specifying homotopy on the DC analysis overrides homotopy specified as a global option.

A.3.2.4 Maxiters

Maximum number of iterations for a DC operating point analysis. It also indirectly specifies the maximum number of iterations taken on each step of a homotopy. Default value is 150, and rarely needs to be changed.

A.3.2.5 Maxsteps

Maximum number of steps used in homotopy method. The default value is 10,000. There is little to be gained by increasing this value because if you get to this value there is very little hope of ever converging. However, you might want to set this value lower so that Spectre gives up quicker.

A.3.2.6 Readforce

Allows you to specify the name of a 'state' file and use its contents as node forces. The state file is an ASCII file that contains name/value pairs, one pair per line. The name is the either the name of a node or a branch. The state file can be manually generated, it can be generated by Spectre during a previous analysis, or it could be generated by Spectre on a previous run.

A.3.2.7 Readns

Allows you to specify the name of a 'state' file and use its contents as nodesets. The state file is an ASCII file that contains name/value pairs, one pair per line. The name is the either the name of a node or a branch. The state file can be manually generated, it can be generated by Spectre during a previous analysis, or it could be generated by Spectre on a previous run.

Names corresponding to signals that no longer exist are ignored except for a warning. A typical usage is to specify the same file for both **reads** and **write**. The first time the analysis is run the file is not found, which generates a warning that can be ignored. On subsequent runs, the nodesets are automatically updated on every run.

A.3.2.8 Restart

This flag causes Spectre to discard the operating point computed as the last point of the previous analysis. This is done because the last point of a temperature sweep, DC sweep, or transient analysis may be far from the operating point needed for the next analysis. With **restart=yes**, which is its default setting, Spectre discards the operating point and starts fresh if anything has changed that causes the previous operating point to be out-of-date.

It is useful to set **restart=no** if you are running a sequence of analyses and you script the analyses to assure the operating point changes little or not at all between analyses. For example, if you want to perform an AC analysis at 0 C and 50 C, and you also need a DC temperature sweep from 0 C to 50 C, then arrange to perform the AC analysis at 0 C first. It computes the operating point at 0 C. Then perform the temperature sweep with **restart=no**. It finishes by computing the operating point at 50 C. Finally, perform the AC analysis at 50 C with **restart=no**. Using **restart=no** for the second and third analyses reduces the time needed by the analyses by giving them good initial guesses for the operating point.

Setting **restart=no** does not cause Spectre to skip computing an

operating point. It only affects what is used as the starting point for the operating point computation. The only time Spectre skips computing the operating point is if it is explicitly told to (see the `preoppo` parameter for the AC analyses, or the `skipdc` parameter for transient analysis) or if it is sure that the operating point is up-to-date. In either case, `restart` does not play a role in deciding whether the operating point should be computed.

A.3.2.9 Write

Specifies the name of the file to which Spectre should write the 'state' of the circuit after the initial point of a DC analysis. This file can be later used as a nodeset file, a node force file, or an initial condition file. The file contains one name/value pair for every signal in the circuit.

A.3.2.10 Writefinal

Specifies the name of the file to which Spectre should write the 'state' of the circuit after the final point of a DC analysis. This file can be later used as a nodeset file, a node force file, or an initial condition file. The file contains one name/value pair for every signal in the circuit.

A.3.3 Transient Analysis Options

Spectre expects the transient analysis options to be given on the transient analysis statement. Type `'spectre -help tran'` for a full list of Spectre's transient analysis parameters.

A.3.3.1 Cmin

Specifies the amount of capacitance that should be connected from each node to ground. Normally this should be zero. However, if you are having trouble with convergence in the intrinsic transient analysis (not the initial condition analysis that precedes the actual

transient analysis) then setting `cmin` to a small value (say 1 fF) usually allows the transient analysis to gracefully work its way beyond discontinuities that would otherwise cause convergence failure.

A.3.3.2 Errpreset

Selects a reasonable collection of parameter settings. Possible values are `conservative`, `moderate` or `liberal`. Specifying `conservative` makes Spectre more cautious and biases it towards accuracy. Conversely, `liberal` makes Spectre more reckless (without taking undue risks) and biases it towards speed. See Section 4.3.2.2 on page 187 for more information.

A.3.3.3 Ic

Use the `ic` parameter to specify the source for the initial conditions. If `ic=none`, then no initial conditions are used. When `ic=node`, then values specified on initial condition statements are used. When `ic=dev`, the values specified as initial condition parameters on various components are used. Finally, if `ic=all`, which is the default, then the initial conditions provided on both initial condition statements and as initial condition parameters are used. If you specify an 'ic' file with the `readic` parameter, the initial conditions are read from the file are used in lieu of any 'ic' statements.

A.3.3.4 Lratio

Ratio used to compute truncation error tolerances from Newton tolerance. The default value is derived from `errpreset`. This parameter is similar to SPICE's `trtol`, except it is twice as small (the default value of `lratio=3.5` corresponds to the default value of `trtol=7`). For more information, see the description of `trtol` in Section A.2.3.10 on page 342.

A.3.3.5 Maxiters

Sets the upper per step iteration limit for transient analysis. The default value is derived from `macromodels`. Increasing this value aids convergence on some circuits, particularly on circuits that exhibit strongly discontinuous behavior, like some macromodels. On such circuits, increasing the value could speed up the analysis considerably. On other circuits, it could cause the analysis to run slower. The recommended upper limit is 50.

A.3.3.6 Maxstep

Specifies the maximum time step used during a transient analysis. On the whole, people tend to over use `maxstep`. Normally, one should tighten `reltol` if greater accuracy is needed. However, there are certain situations where tightening `maxstep` is the best way to get the best accuracy. Examples include simulating the onset of oscillation, where the oscillation is small compared to the DC level and so tightening `reltol` is largely ineffective, and when performing Fourier analysis where a well controlled and nearly evenly spaced time step is a virtue. The default value is derived from `errpreset`.

Unlike SPICE, the maximum time step in Spectre is not affected by the electrical length of the transmission lines.

A.3.3.7 Method

Specifies the integration method or combination of integration methods used during the transient analysis. The possible choices include

euler

Backward-Euler is used exclusively

trap

Backward-Euler and the trapezoidal rule are used.

traponly

Trapezoidal rule is used almost exclusively (backward-Euler is used at break-points).

A.3. Spectre Options

gear2

Backward-Euler and second-order Gear are used.

gear2only

Gear's second-order backward-difference method is used almost exclusively (backward-Euler is used at break-points).

trapgear2

Allows all three integration methods to be used. Trapezoidal rule and second-order Gear are used on alternating time steps, while backward-Euler is used at break-points.

The trapezoidal rule is usually the most efficient when you want high accuracy. This method occasionally exhibits point-to-point ringing, but you can control this by tightening the error tolerances. For this reason, if you choose very loose tolerances to get a quick answer, second-order Gear probably gives better results. Second-order Gear and backward-Euler make systems appear more stable than they really are. This effect is less pronounced with second-order Gear or when you request high accuracy.

See Section 4.2.2 on page 133 for a more detailed description of the advantages and disadvantages of each method. The default is derived from `errpreset`.

A.3.3.8 Readic

Allows you to specify the name of a 'state' file and use its contents as initial conditions. The state file is an ASCII file that contains name/value pairs, one pair per line. The name is the either the name of a node or a branch. The state file can be manually generated, it can be generated by Spectre during a previous analysis, or it could be generated by Spectre on a previous run.

A.3.3.9 Readns

Allows you to specify the name of a 'state' file and use its contents as nodesets. The state file is an ASCII file that contains name/value

pairs, one pair per line. The name is the either the name of a node or a branch. The state file can be manually generated, it can be generated by Spectre during a previous analysis, or it could be generated by Spectre on a previous run.

Names corresponding to signals that no longer exist are ignored except for a warning. A typical usage is to specify the same file for both reads and write. The first time the analysis is run the file is not found, which generates a warning that can be ignored. On subsequent runs, the nodesets are automatically be updated on every run.

A.3.3.10 Relref

This parameter determines how the relative error used in convergence criteria and the truncation error criterion is computed. The possible choices include

pointlocal

Compares the relative errors in quantities at each node to that node alone.

alllocal

Compares the relative errors at each node to the largest values found for that node alone for all past time.

sigglobal

Compares relative errors in each of the signals to the maximum for all similar signals at any previous point in time.

alldglobal

Same as sigglobal except that it also compares the residues (KCL error) for each node to the maximum of that node's past history.

Default derived is from `errpreset`. See Section 4.3.2.1 on page 183 for more information.

A.3. Spectre Options

A.3.3.11 Restart

This flag causes Spectre to discard the operating point computed as the last point of the previous analysis. For more information, see the description for `restart` in Section A.3.2.8 on page 349.

A.3.3.12 Skipdc

Flag that indicates the transient analysis should not employ the DC analysis algorithms in an attempt to compute any values in the initial state that were not specified as initial conditions (this is similar to the `uic` parameter of SPICE). This parameter is generally used when the simulator refuses to converge otherwise. See Section 4.4.1 on page 207 for more information about the various choices available with `skipdc`.

A.3.3.13 Step

Specifies the minimum step size Spectre takes to maintain the aesthetics of the results. Spectre monitors the voltage across capacitors and the current through inductors, and chooses the time step to maintain the accuracy of the solution. It does not need to do this for the node voltages and branch currents not associated with capacitors or inductors, because these quantities are accurate regardless of the time step. However, when plotting these waveforms it is important to have more points where the curvature is high in order to convey the true shape of the waveform. For this reason, Spectre chooses the time step to faithfully depict the shape of waveforms that do not directly affect the accuracy of the solution. However, occasionally these waveform have discontinuous jumps that would require infinitely small time steps to resolve. `step` specifies the minimum step size that will be taken to resolve the details of these 'algebraic' signals. The default value is derived from `errpreset`.

A.3.3.14 Write

Specifies the name of the file to which Spectre should write the 'state' of the circuit after the initial point of a transient analysis. This file can be later used as a nodeset file, a node force file, or an initial condition file. The file contains one name/value pair for every signal in the circuit.

A.3.3.15 Writefinal

Specifies the name of the file to which Spectre should write the 'state' of the circuit after the final point of a transient analysis. This file can be later used as a nodeset file, a node force file, or an initial condition file. The file contains one name/value pair for every signal in the circuit.

Appendix B

Spectre Netlist Language

B.1 Introduction

In addition to accepting SPICE netlists, Spectre also supports a simple, powerful, and extensible language for describing netlists. This appendix describes the basics of Spectre's netlist language only to the level of detail needed to allow you to understand the netlists given in this book.

B.2 The Language

When reading a netlist, SPICE determines the type of a component by the first letter in its name. For example, R1 must be a resistor and Vin must be a voltage source. This approach follows engineering convention and fits well with SPICE's original goal of being a simulator for integrated circuits. However, this convention has two unfortunate limitations. First, it limits the simulator to only supporting only 25 component types (X is reserved for subcircuits). Second, it does not allow the representation type of the components to change. For example, Q1 must refer to a built-in model for a bipolar transistor. It might also be convenient use a macromodel for the transistor. Indeed, integrated transistors often have other junctions near-by that can form parasitic transistors. The built-in model does not include parasitic transistors, but it is easy to construct a macromodel using a subcircuit that contains the original transistor

along with the parasitic transistor. However, because of the constraints on the first letter, one cannot change the type of a transistor from built-in model to macromodel without changing the transistors name. This might not seem like a serious problem until you consider a circuit with thousands of transistors. If you would like to change just the NPN transistors from the built-in model to the subcircuit macromodel that includes the parasitic transistor, you have to make perhaps thousands of hand edits on the netlist.

Spectre's native netlist language is modeled after the SPICE language, but is more uniform. It provides several important features that are unavailable in SPICE due to limitations of its language, including the ability to support an arbitrary number of primitives and the ability to switch representation type.

There are three basic types of primary statements, along with some secondary control statements. The primary statements include component instance, component model, and analysis statements. The control statements include those for specifying initial conditions, nodesets, outputs, etc, and are not discussed further here. In addition, Spectre provides statements used to define parameterized subcircuits.

Basic Language Attributes The Spectre parser has two modes. By default, Spectre is configured to expect SPICE netlists unless told otherwise. When it reads

```
simulator lang=spectre
```

it switches off SPICE compatibility. When doing so, it makes the following changes:

1. Spectre no longer accepts SPICE statements and constructs.
2. The language becomes case-sensitive, with all keywords being lower case.
3. Standard SI scale-factors are used as a convenient way of specifying either very large or very small numbers. The SI scale

$P = 10^{15}$	$T = 10^{12}$	$G = 10^9$	$M = 10^6$	$K = 10^3$
$k = 10^3$	$- = 1$	$\% = 10^{-2}$	$c = 10^{-2}$	$m = 10^{-3}$
$u = 10^{-6}$	$n = 10^{-9}$	$p = 10^{-12}$	$f = 10^{-15}$	$a = 10^{-18}$

Table B.1: SI scale factors.

$t = 10^{12}$	$g = 10^9$	$meg = 10^6$	$k = 10^3$	$m = 10^{-3}$
$mil = 25.4 \times 10^{-6}$	$u = 10^{-6}$	$n = 10^{-9}$	$p = 10^{-12}$	$f = 10^{-15}$

Table B.2: SPICE scale factors.

factors are different than those used by SPICE. For example, with SI, $m = 10^{-3}$ and $M = 10^6$, whereas with SPICE both m and $M = 10^{-3}$, while $MEG = 10^6$. The SI scale factors are detailed in Table B.1 while the SPICE scale-factors are shown in Table B.2.

To switch back to accepting SPICE netlists, use

```
simulator lang=spice
```

When Spectre includes one file from another using the include statement, it automatically switches to SPICE mode at the beginning of the new file, and returns to the previously active mode when it finishes reading it. For this reason, all Spectre netlists must begin with a `lang=spectre` statement.

Model Statements Often, certain characteristics are common to a large number of instances of components of the same type. For example, the saturation current of a diode is a function of the process used to construct the diode and also of the area of the diode. Rather than describing the process on each diode instantiation, that

description is done once in a model statement and many diode instances refer to it. The area, which can be different for each component, is included on each instance statement. Though it is possible to have several model statements for a particular type of component, each instance can only reference at most one model. Not all types of components support model statements.

Model statements have the form

```
model dnp diode is=3.1e-10 n=1.12 cjo=3.1e-8 vj=.914
```

In this example, *dnp* is the model name, and *diode* is the primitive name. The primitive name is either the name of a built-in primitive or a user designed behavioral model.

Instance Statements The instance statement consists of an instance name, the nodes to which the terminals of the instance are connected, the name of the master, and the parameters. It takes the form,

```
M1 (4 pin 1 1) nmos w=402.4u l=7.6u
```

where *M1* is the instance name, '(4 pin 1 1)' is the node list (parentheses are optional), *nmos* is the master name, in this case a model name, and 'w=402.4u l=7.6u' are the parameters. The master field contains either the name of a built-in component type (for example, capacitor), the name of a user-defined behavioral model, a model name, or a subcircuit definition name. The nodes must appear in the order defined by the master. Unlike SPICE, the first character in the instance names is not fixed to any particular value by the type field.

The list of Spectre's available built-in components is shown in Table B.3 on the facing page and Table B.4 on page 362. Further information on these components and their parameters is found by using Spectre's -help command-line option.

	bjt	Bipolar junction transistor
bsim1	BSIM1	MOSFET
bsim2	BSIM2	MOSFET
bsim3	BSIM3	MOSFET
diode	Diode	model
gaas	GaAs	MESFET
jfet	Junction	FET
mos0	MOSFET	level-0
mos1	MOSFET	level-1
mos2	MOSFET	level-2
mos3	MOSFET	level-3
mos8	GE-Intersil	MOSFET
phy-res	Physical	resistor

Table B.3: Spectre's semiconductors. In addition, Spectre supports a number of proprietary foundry models.

Subcircuit Definitions Subcircuit definitions are circuit macros that can be expanded anywhere in the circuit any number of times. They take the form

```
subckt diffamp (pin nin pout nout)
  parameters ad=1 ac=0 rd=0 rc=0
  Acmp (i1 0 pin 0)      vcvs gain=ac/2
  Acmn (cm i2 nin 0)     vcvs gain=ac/2
  Admp (poutx cm pin nin) vcvs gain=ad/2
  Admn (noutx cm pin nin) vcvs gain=-ad/2
  Rdp (pout poutx)       resistor r=rd/2
  Rdn (nout noutx)       resistor r=rd/2
  Rc (i1 i2)              resistor r=rc-rd/4
ends diffamp
```

subckt, parameters, and ends are keywords. This subcircuit has four terminals (*pin*, *nin*, *pout*, and *nout*), and four parameters (*ad*, *ac*, *rd*, and *rc*). The parameters are used in constant expressions that specify the parameter values for the components in the subcircuit.

capacitor	Two terminal capacitor
cccs	Linear current-controlled current source
ccvs	Linear current-controlled voltage source
core	Magnetic core with hysteresis
delay	Ideal delay line
fourier	Fourier analyzer
inductor	Two Terminal inductor
iprobe	Current probe
isource	Independent current source
msline	Microstrip transmission line
mutual.inductor	Mutual inductor coupling
nport	N-port (specified using file of S-parameters)
peccs	Polynomial current-controlled current source
pccvs	Polynomial current-controlled voltage source
port	Independent resistive source
pvcvs	Polynomial voltage-controlled current source
pvcvs	Polynomial current-controlled voltage source
relay	Four terminal relay
resistor	Two terminal resistor
scccs	s-domain current-controlled current source
sccvs	s-domain current-controlled voltage source
svccs	s-domain voltage-controlled current source
svcvcs	s-domain voltage-controlled voltage source
switch	Ideal switch
tlline	Transmission line (with skin effect)
transformer	Linear two-winding ideal transformer
vccs	Linear voltage-controlled current source
vcvs	Linear voltage-controlled voltage source
vsouce	Independent voltage source
winding	Winding for magnetic core
zcacs	z-domain current-controlled current source
zcacs	z-domain voltage-controlled voltage source
zvcvs	z-domain current-controlled current source
zvcs	z-domain voltage-controlled voltage source

Table B.4: Spectre's components.

When an instance in your input file refers to a subcircuit definition, the instances specified within the subcircuit are inserted into the circuit. Local model definitions are allowed within a subcircuit definition. Also instances of other subcircuits as well as local subcircuit definitions are allowed within a subcircuit definition. Names of subcircuits and models defined within a subcircuit are strictly local to that subcircuit.

Instances that instantiate a subcircuit definition are referred to as subcircuit calls. The node names (or numbers) specified in the subcircuit call are substituted, in order, for the node names given in the subcircuit definition. All instances that refer to a subcircuit definition must have the same number of nodes as specified in the subcircuit definition and must be in the same order. Node names inside the subcircuit definition are strictly local unless declared otherwise in the input file with a global statement. Model names for models defined inside a subcircuit definition are strictly local and are not accessible for specifying component instances outside the current subcircuit definition.

Parameter specification in subcircuit definitions is optional. Any parameters that are specified are referred to by name followed by an equals sign and then a default value. If, when making a subcircuit call in your input file, you do not specify a particular parameter, then this default value is used in the macro expansion. Subcircuit parameters are used in expressions within the subcircuit as demonstrated in the example above ($r=r_c-r_d/4$). The expressions employ subcircuit parameters and constants, and are constructed using the four standard arithmetic operators (+, -, *, /) and parentheses for grouping.

Analysis Statements Analysis statements have the same form as component instance statements, the main difference is that they specify analyses rather than components, and the order in which they are given determines the order in which the analyses are run. Analysis statements take the form,

```
dmStepResponse tran stop=2us erpreset=conservative
```

ac	AC small-signal analysis
alter	Alter a circuit or component parameter
check	Check component parameter values
dc	DC operating-point analysis
info	Circuit and component information
noise	Noise analysis
options	Set circuit options (immediate)
set	Set circuit options (deferred)
shell	Run a UNIX shell command
sp	S-parameter analysis
tdr	Time-domain reflectometer analysis
tran	Transient analysis
xf	Small-signal transfer-function analysis

Table B.5: Spectre's analyses.

It is also possible to provide a list of nodes with an analysis, as in

```
dmNoise (dout 0) noise start=1 stop=1GHz iprobe=Vid
```

In contrast to SPICE, Spectre provides a rich set of parameters that are used to specify the behavior of each analysis. Any number of any analysis commands can be given in any order. Information on individual analyses are available by using the `-help` command line option to Spectre.

Currently, Spectre supports the analyses shown in Table B.5.

SPICE Extensions In addition to providing support for SPICE netlists, Spectre also provides access to many of the benefits of the Spectre netlist language from SPICE netlists. For example, parameterized subcircuits and multiple analyses are available. Furthermore, any Spectre specific parameters can be appended to the SPICE form of a component instance or analysis statement. You are also free to use either SPICE or Spectre forms for instance and analysis state-

ments in SPICE netlists. By switching to Spectre forms, you get representation switching, named parameters, constant expressions for parameter values, and support for new component types without constraining the first letter of the component instance name.

Bibliography

- [antognetti93] Paolo Antognetti and Giuseppe Massobrio (editors). *Semiconductor Device Modeling with SPICE*. McGraw-Hill, 1993.
- [boser88] Bernhard E. Boser and Bruce A. Wooley. "The design of sigma-delta modulation analog-to-digital converters." *IEEE Journal of Solid-State Circuits*. vol. 26, no. 6, December 1988.
- [chua87] Leon O. Chua, Charles A. Desoer and Ernest S. Kuh. *Linear and Nonlinear Circuits*. McGraw-Hill, 1987.
- [connelly92] J. Alvin Connelly and Pyung Choi *Macromodeling with SPICE*. Prentice-Hall, 1992.
- [duff86] I. S. Duff, A. M. Erisman and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 1986.
- [gilbert82] Barrie Gilbert. "A monolithic microsystem for analog synthesis of trigonometric functions and their inverses." *IEEE Journal of Solid-State Circuits*, vol. SC-17, No. 6, December 1982.
- [gray84] Paul R. Gray and Robert G. Meyer. *Analysis and Design of Analog Integrated Circuits*. John Wiley and Sons, 1984.
- [harris78] Fredric J. Harris. "On the use of windows for harmonic analysis with the discrete Fourier trans-

form." *Proceedings of the IEEE*, vol. 66, no. 1, January 1978.

[jeng] Min-Chie Jeng. To be published by Kluwer Academic Publishers, Boston.

[kinget94] Peter Kinget, Jan Crois, Mark Ingels and Enzo Peluso. *Circuits and Devices Magazine*. "Are circuit simulators becoming too stable?", vol. 10, no. 3, pp. 50, May 1994.

[kundert90] Kenneth S. Kundert, Jacob K. White and Alberto Sangiovanni-Vincentelli. *Steady-State Methods for Simulating Analog And Microwave Circuits*. Kluwer Academic Publishers, Boston 1990.

[maas88] Stephen A. Maas. *Nonlinear Microwave Circuits*. Artech, 1988.

[mccalla87] W. J. McCalla. *Fundamentals of Computer-Aided Circuit Simulation*. Kluwer Academic Publishers, Boston 1987.

[meyer71] J. E. Meyer. "MOS models and circuit simulation." *RCA Review*, vol. 32, 1971.

[meyer89] Robert G. Meyer and William D. Mack. "A wide-band class AB monolithic power amplifier." *IEEE Journal of Solid-State Circuits*, vol. 24, no. 1, February 1989.

[middlebrook75] R. D. Middlebrook. "Measurement of loop gain in feedback systems." *International Journal of Electronics*, vol. 38, no. 4, April 1975.

[motchenbacher73] C. D. Motchenbacher and F. C. Fitchen. *Low Noise Electronic Design*. John Wiley and Sons, 1973.

[pederson84] Donald O. Pederson. "A historical review of circuit simulation." *IEEE Transactions on Circuits and Systems*, vol. CAS-31, no. 1, January 1984.

[rohrer71] Ronald Rohrer, Laurence Nagel, Robert Meyer and Lynn Weber. "Computationally efficient electronic-circuit noise calculations." *IEEE Journal of Solid-State Circuits*, vol. SC-6, no. 4, August 1971.

[sangiovanni81] Alberto L. Sangiovanni-Vincentelli. "Circuit Simulation." In *Computer Design Aids for VLSI Circuits*, P. Antognetti, D. O. Pederson and H. De Man (editors). Martinus Nijhoff Publishers, 1986, pp. 19-112.

[spice2] *SPICE Version 2G*, Available through the Software Distribution Office, Cory Hall, University of California, Berkeley, CA 94720.

[spice3] *SPICE Version 3F*, Available through the Software Distribution Office, Cory Hall, University of California, Berkeley, CA 94720.

[vlach83] Jiri Vlach and Kishore Singhal. *Computer Methods for Circuit Analysis and Design*. Van Nostrand Reinhold, 1983.

[vladimirescu94] Andrd Vladimirescu. *The SPICE Book*. Wiley, 1994.

[ward78] Donald. E. Ward and Robert. W. Dutton. "A charge-oriented model for MOS transistor capacitances." *IEEE Journal of Solid-State Circuits*, vol. SC-13, no. 5, pp. 703-708, October 1978.

[white86] J. K. White and A. Sangiovanni-Vincentelli. *Relaxation Techniques for the Simulation of VLSI Circuits*. Kluwer Academic Publishers, Boston 1986.

[yang82] Ping Yang and Pallab K. Chatterjee. "SPICE modeling for small geometry MOSFET circuits." *IEEE Transactions on the Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-1, no. 4, pp. 169-182, October 1982.

[Zverev67] Anatol I. Zverev. *Handbook of Filter Synthesis*. Wiley, 1967.

Index

A

abstol, 23, 24, 25, 36, 37,
178, 200, 205, 207,
225, 335

AC analyses, 51–128

applications, 67–119

noise analysis, 57

practice, 54–67

SP analysis, 56

theory, 53–54

XF analysis, 55

AC analysis, 54

differential amplifiers, 113

feedback, 67

noise, 57

non-quiescent operating

point, 113

versus bias, 106, 108

ADC

oversampled, 160, 303

admittance

large signal, 312

aliasing, 263, 266, 271

SPICE, 280

albrkpts, 190

allglobal, *see* relref

alllocal, *see* relref

almost-periodic Fourier

transform, 264

amplifier

differential, 113

distributed, 203

feedback, 67

breaking loop, 73, 76

configurations, 69

ideal circuits, 68, 73

non-ideal circuits, 84

open-loop gain, 79

parameters, 73

series series, 71, 93

series shunt, 69, 97

shunt series, 72, 102

shunt shunt, 70, 87

low-distortion, 299

narrow-band, 316

slew-rate limiting, 113

transconductance, 106

traveling wave, 203

analog-digital converter

oversampled, 160, 303

APFT, 264

approx, 276, 343

artificial damping, 147, 219

autonomous, 189n

circuits, 268, 289

distortion, 310

B

- backward Euler, 38, 132, 133, 166, 196, 208
- damping, 147
- delay, 156
- SPICE, 208
- $s \rightarrow z$ domain map, 140
- backward-difference formula, *see* Gear's method
- balun, 113
- BDF, *see* Gear's method
- bifurcation
 - homotopy, 31
- BJT noise, 60
- break point, 133, 155, 156, 189, 189, 218, 222, 288-290
- transmission line, 191
- bypass
 - device, 194, 276, 281, 288
 - Fourier analysis, 276, 281, 288

C

- capacitor
 - current, 150, 152
 - junction, 108
 - negative, 231
- cavity oscillator, 310
- charge
 - as state variable, 182, 287
 - conservation, 167-176, 178, 182, 200, 225
 - convergence criteria, 167
- charge-storage circuits, 160, 167, 224

check, 347

- check statement, 47
- chgtol, 179, 182, 339
- clock distribution networks, 234
- clocked analog circuits, 238
- distortion, 313
- closed-loop gain, 69
- cmn, 202, 350
- common-mode gain, 117
- feedback parameter
- measurement, 75
- component
 - models, 359
- compression point, 253
- conservative, *see* **errpreset**
- continuation method, *see* homotopy
- convergence
 - bypass, 195
 - criteria, 22, 19-23, 41, 184, 186
 - charge conservation, 167
 - Fourier analysis, 276
 - truncation error, 178, 182
 - false, 21
 - floating nodes, 25
 - force, 28
 - gmin, 26
 - guaranteed, 18
 - homotopy, 30
 - initial conditions, 198
 - large circuits, 45
 - large floating capacitors, 205
 - loops of shorts, 25

D

- macromodels, 227
- nodeset, 28
- remedies
 - DC analysis, 35
 - transient analysis, 206
- small floating resistors, 24
- topology checker, 26
- transient analysis, 201
- cmn, 202
- troubled netlist, 24
- using transient analysis, 38, 207
- crystal oscillator, 310
- current
 - errpreset**, 237
 - step, 237
 - capacitor, 152
 - Fourier analysis, 278
 - Gear overshoot, 155
 - loop gain, 77
 - measurements, 235
 - noise source, 64
- delay
 - backward Euler, 156
 - $\Delta\omega$ modulator, 160
 - noise and distortion, 303
- demodulation
 - AM, 241
 - FM, 242
 - sampling, 321
 - strobing, 241
 - synchronous detection, 317
- detection, *see* demodulation
- device bypass, 194, 276, 281, 288
- DFT, *see* discrete Fourier transform
- diagnose, 35, 343
- differential amplifiers, 113
- differential-mode gain, 117
- diode
 - model, 12
 - noise, 60
- discrete Fourier transform, 265, 266
- aliasing, 266, 271
- external, 289
- FFT, 267, 272, 289
- windows, 290
- amplitude resolution, 298
- frequency resolution, 298
- distortion
 - analysis, 251
 - clocked analog circuits, 313
 - compression point, 253
 - harmonic, 252, 316
- damped pseudo-transient, 30
- damping
 - artificial, 147
 - distribution networks, 234
 - unstable circuits, 217, 219
- DC analysis, 15-50
- applications, 42-49
- convergence, 23-35
- using transient analysis, 207
- practice, 35-42
- theory, 16-23

- intercept point, 254
- intermodulation, 252, 316, 329
 - demodulation, 317, 321
 - mixer, 328, 329
- narrow-band circuits, 252, 316
 - oscillator, 310
- pulse-width modulator, 308
 - $\Sigma\Delta$ modulator, 303
 - wide-band circuits, 251
- distributed amplifier, 203
- distributed components, 111, 191
- distribution networks, 234
- dptran, 30
- dskip, 276

E

- end, 361
- ends, 361
- equilibrium point, 15, 16, 166, 209, 347
 - multiple, 17
 - unstable, 17, 209, 219
- errpreset, 187, 200, 227, 351
 - current measurements, 237
- extremes, *see* info statement

F

- $1/f$ noise, 59
- false convergence, 21
- fast Fourier transform, *see* discrete Fourier

transform

- feedback, 67–106
 - breaking loop, 73, 76
 - configurations, 69
 - factor, 69
- ideal circuits, 68, 73
 - series series, 71
 - series shunt, 69
 - shunt series, 72
 - shunt shunt, 70
- non-ideal circuits, 84
 - series series, 93
 - series shunt, 97
 - shunt series, 102
 - shunt shunt, 87
- open-loop gain, 79
 - parameters, 73
- FFT, *see* discrete Fourier transform

filter

- band-pass
 - distortion, 316
 - low loss, 147
- FIR filters, 264
 - first character, 360
 - flicker noise, 59
- folds
 - homotopy, 31
- force, 28, 29, 347
 - versus nodesets, 29
- force, 347
- forward Euler, 132
 - $s \rightarrow z$ domain map, 144
 - unstable, 135
- Fourier analysis, 251–334
 - amplitude resolution, 298
 - applications, 299–332

- clocked analog circuits, 313
 - current, 278
 - error mechanisms, 267
 - aliasing, 266, 271, 280
 - bypass, 194, 276, 281, 288
 - inaccurate period, 267, 311
 - interpolation, 272, 279, 281, 286, 287
 - nulls, 331
 - simulator noise, 276
 - SPICE, 279
 - transients, 268, 310
 - external, 289
- frequency resolution, 298
- intermodulation
 - distortion, 252
- demodulation, 317, 321
- large-signal transfer functions, 312
- low-distortion amplifier, 299
 - mixer, 328
- narrow-band circuit, 316
- oscillators, 268, 289, 310, 312
 - practice, 267–299
- pulse-width modulator, 308
 - $\Sigma\Delta$ modulator, 303
 - Spectre, 282
 - SPICE, 278
 - switched-filter, 313
 - THD, 252, 299
 - theory, 257–267
 - windows, 290

G

- G -parameters, 99, 102
- gain
 - large signal, 312
 - open, closed, and loop, 67, 69
- Gear's method, 132
 - damping, 147, 246
 - overshoot, 155
 - $s \rightarrow z$ domain map, 141
 - unstable, 150
- global, 363
- global truncation error, 160
- g_{min} , 24, 26, 37, 39, 226, 336, 343
 - automatic installation, 28
 - g_{min} stepping, 32, 33
 - g_{min} stepping, 30

H

- H -parameters, 97, 105
- harmonic distortion, 252, 316
- clocked analog circuits, 313
- compression point, 253

- intercept point, 254
 - low distortion amplifier, 299
 - mixer, 328
 - oscillator, 310
 - hierarchical circuit
 - description, 361
 - homotopy, 343, 348
 - homotopy method, 29
- I**
- iabstol, 23, 24, 25, 36, 37, 207, 344
 - ic, 351
 - IMD, *see* intermodulation distortion
 - immittance, 67n, 312n
 - impedance
 - large signal, 312
 - inductor
 - negative, 231
 - voltage, 150, 152
 - info statement, 45, 209
 - initial condition, 11, 195, 347
 - Spectre, 199
 - SPICE, 196
 - starting oscillators, 210
 - uic, 196
 - versus nodesets, 199
 - input, *see* info statement
 - input impedance
 - differential- and common-mode, 117
 - input-referred noise, 62
 - integration methods, 131-157
 - artificial damping, 147
 - characteristics, 133

- delay, 156
 - overshoot, 155
 - ringing, 150, 155
 - stiff circuits, 134
 - $s \rightarrow z$ domain map, 139
- integrator**
- truncation error, 161
- intercept point, 254**
- intermodulation distortion, 252, 316
 - compression point, 253
 - demodulation, 317, 321
 - intercept point, 254
 - mixer, 328
- narrow-band circuit, 316**
- interpolation error**
- Fourier analysis, 272, 279, 281, 286, 287
- iteration-count time-step control, 179**
- it11, 36, 338
 - it12, 36, 338
 - it13, 339
 - it14, 206, 339
 - it15, 340
 - it16, 36, 339

J

- Jacobian
 - singular, 25, 205
- JFET noise, 60
- Johnson noise, 59
- junction
 - capacitance, 108
 - model, 12
 - noise, 60

K

- Kirchoff's laws, 9

L

- lang, 358
- language
 - first character, 360
 - model statements, 359
 - subcircuit call, 361
 - subcircuit definition, 361
- large circuits, 45, 234
- large-signal transfer functions, 312

- liberal, *see* erreset

- limit, 344

- limpts, 336

- local truncation error, 160, 166, 180

- threshold criterion, 177, 180

- loop gain, 69

- current, 77

- voltage, 77

- low-loss resonators, 147

- LTE, *see* local truncation error

- iteratio, 181, 187, 207, 225, 351

- lvltim, 177, 179, 340

M

- macromodels, 227
- macromodels, 344
- matrix
 - singular, 25, 205
- maxiters, 206, 348, 352

- maxord, 340

- maxstep, 178, 187, 200, 201, 276, 280-282, 289, 298, 332, 352

- maxsteps, 348

- method, 187, 188, 341, 352

- Middlebrook, 77

- mixer, 329

- noise, 66

- model, 360

- model statements, 359

- moderate, *see* erreset

- modified nodal analysis, 11

- modulator

- pulse-width, 308

- $\Sigma\Delta$, 160, 303

- MOSFET noise, 61

- mu, 208

N

- negative capacitors, inductors, resistors, 231

- Newton's method, 18

- nodal analysis, 11

- node force, 29

- nodeset, 28

- nodesets

- versus force, 29

- versus initial conditions, 199

- noise

- BJT, 60

- differential- and common-mode, 119

- diode, 60

- $1/f$, 59

- flicker, 59

- input referred, 62, 119
- JFET, 60
- Johnson, 59
- junction, 60
- mixer, 66
- MOSFET, 61
- Nyquist, 59
- oscillator, 66
- phase, 67
- pink, 59
- red, 59
- resistor, 59
- shot, 59
- $\Sigma\Delta$ modulator, 303
- simulator
 - bypass, 194, 276, 281, 288
 - Fourier analysis error mechanism, 276
 - source, 64
 - spot, 59
 - switched-filter, 66
 - thermal, 57
 - total, 61
 - white, 59
- noise analysis, 57
- limitations, 66
- noise density, 59
- noise factor and figure, 62
- nonconvergence, *see* convergence
- nonperiodicity, 268
- Nyquist frequency, 272
- noise, 59

O

- opamps, 79
- open-loop gain, 79
- operating point, 15, 51, 55
- oppoint, *see* info statement
- optcheck, 345
- oscillation, *see* ringing
- oscillator, 160, 209-218
 - cavity, crystal, 310
 - DC solution, 17, 35, 38, 39, 166, 208
 - distortion, 310
 - emitter-coupled pair, 211
 - Fourier analysis, 268, 289, 312
 - high-Q, 147
 - in large circuit, 235
 - noise, 66
 - relaxation, 182, 189, 202, 218
 - ring, 39, 210
 - starting, 200, 210
 - truncation error, 161
 - unexpected, 235
- output, *see* info statement
- output impedance, 55
- differential- and common-mode, 117
- oversampled ADC, 160, 303
- overshoot
 - current, 155
 - Gear's method, 155

P

- parameters
 - subcircuit, 363
 - parameters, 361

- periodic, 258
- phase noise, 67
- phasors, 53
- pink noise, 59
- pivabs, 345
- pivotdc, 346
- pivrel, 336, 346
- pivotol, 337
- pointlocal, *see* relref
- power distribution networks, 234
- power measurements, 238
- power supply rejection, 55
- differential- and common-mode, 117
- pseudo-transient analysis, 30
- pulse-width modulator
 - distortion, 308

Q

- Q or quality factor
 - high-Q resonators, 147
- quasi-fundamental frequency, 265, 317
- quasiperiodic, 263
- demodulation, 317

R

- readforce, 348
- readic, 353
- readns, 349, 353
- red noise, 59
- relaxation oscillator, *see* oscillator, relaxation
- relref, 187, 354

- reltol, 22, 41, 166, 177, 181, 187, 200, 207, 225, 337, 346
- affect on Fourier analysis, 276

resistor

- negative, 231
- noise, 59
- noiseless, 64
- resonators, 147
- restart, 37, 349, 355
- rforce, 27, 346
- right-hand plane zero, 79, 233
- ringing
 - artificial damping distribution networks, 234
- trapezoidal, 150, 155
- rmin, 25

S

- S-parameters, *see* scattering parameters
- sampling
 - aliasing, 266, 271
 - demodulation, 321
 - strobing, 238
- scattering parameters
 - analysis, 56
 - large signal, 312
- Schmitt trigger, 202
- series-series feedback, 71, 93
- series-shunt feedback, 69, 97
- shot noise, 59
- shunt-series feedback, 72, 102
- shunt-shunt feedback, 70, 87
- sigglobal, *see* relref

- $\Sigma\Delta$ modulator, 160
 - noise and distortion, 303
- simulator statement, 358
- singular matrix or Jacobian, 25, 205
 - g_{min} , 40
- sinusoidal circuits, 226
- `skipdc`, 208, 355
- slew-rate limiting, 113
- source stepping, 30, 32, 33, 339
- SP analysis, 56
- spectral leakage, 263, 268
- spectral noise density, 59
- Spectre
 - Fourier analysis, 282
 - initial condition, 199
 - language
 - first character, 360
 - model statements, 359
 - subcircuit call, 361
 - subcircuit definition, 361
 - options, 342
 - time-step control, 180
- SPICE
 - Fourier analysis, 278
 - initial condition, 196
 - language
 - first character, 360
 - options, 335
 - time-step control, 177
 - iteration count, 179
 - spot noise, 59
 - stability, 51, 219
 - Gear methods, 134
 - preservation versus integration method,
- 147
 - time-step constraint, 135
 - trapezoidal rule, 141
 - state file, 348-350, 353, 356
 - statements
 - `global`, 363
 - models, 359
 - parameters, 361
 - subckt, 361
 - step, 355
 - current measurements, 237
 - stiff circuits, 134
 - strobing, 238
 - demodulation, 241
 - subcircuit
 - definition nesting, 363
 - instance nesting, 363
 - subcircuit parameters, 363
 - subcircuits
 - call, 361
 - definition, 361
 - parameter expressions, 363
 - parameterized, 363
 - subckt, 361
 - switched-filter, 118, 160, 167, 224, 238
 - distortion, 313
 - noise, 66
 - symmetry
 - continuation methods, 32
 - synchronous detection, 317
- T**
 - TDR analysis, 56
 - THD, *see* harmonic distortion

- thermal noise, 57
- time-domain reflectometry, 56
- T_{max} , 341
- T_{max} , *see* `maxstep`
- total harmonic distortion, *see* harmonic distortion
- total noise, 61
- transfer function, 53, 55
 - large-signal, 312
- transient analysis, 129-249
 - applications, 207-246
 - computing DC operating point, 207
 - convergence, 201
 - oscillator, 209
 - practice, 177-207
 - sensitivity of circuit to error, 160
 - theory, 130-157
 - transmission line
 - break point, 191
 - T_{max} , 341
 - trapezoidal rule, 132
 - ringing, 150, 152, 155
 - stiff circuits, 134
 - $s \rightarrow z$ domain map, 141
 - traveling-wave amplifier, 203
 - `trtol`, 166, 178, 181, 207, 225, 342
 - truncation error, 160
 - sensitivity of circuit to error, 160
 - T_{step} , 341
 - Fourier analysis, 280
 - 2 ports, 84
- U**
 - `uic`, 196, 342
 - undersampling, 226
 - unstable
 - circuits, 219
 - equilibrium point, 219
 - Gear methods, 134
- V**
 - `vabstol`, 22, 37, 181, 182, 347
 - vector detection, 317, 321
 - `vntol`, 22, 37, 200, 338
 - voltage
 - as state variable, 182, 287
 - inductor, 152, *see* current, capacitor
 - loop gain, 77
 - noise source, 64
- W**
 - white noise, 59
 - window functions, 290
 - amplitude resolution, 298
 - frequency resolution, 298
 - `write`, 350, 356
 - `writetfinal`, 350, 356
- X**
 - XF analysis, 55
- Y**
 - Y-parameters, 87, 96

Z

- Z*-parameters, 90, 94
- z*-transform
 - backward Euler, 140
 - forward Euler, 144
 - Gear's method, 141
 - trapezoidal rule, 141
- zero
 - right-hand plane, 233
- zero, right-hand plane, 79