SANDIA REPORT
SAND2023-05926
Printed July 2023

Sandia National Laboratories

Xyce™ Parallel Electronic Simulator Users' Guide, Version 7.7

Eric R. Keiter, Thomas V. Russo, Richard L. Schiek,
Heidi K. Thornquist, Ting Mei, Jason C. Verley
Karthik V. Aadithya, Joshua D. Schickling

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

This manual describes the use of the Xyce Parallel Electronic Simulator. Xyce has been designed as a SPICE-compatible, high-performance analog circuit simulator, and has been written to support the simulation needs of the Sandia National Laboratories electrical designers. This development has focused on improving capability over the current state-of-the-art in the following areas:

- Capability to solve extremely large circuit problems by supporting large-scale parallel computing platforms (up to thousands of processors). This includes support for most popular parallel and serial computers.
- A differential-algebraic-equation (DAE) formulation, which better isolates the device model package from solver algorithms. This allows one to develop new types of analysis without requiring the implementation of analysis-specific device models.
- Device models that are specifically tailored to meet Sandia's needs, including some radiation-aware devices (for Sandia users only).
- Object-oriented code design and implementation using modern coding practices.

Xyce is a parallel code in the most general sense of the phrase — a message passing parallel implementation — which allows it to run efficiently a wide range of computing platforms. These include serial, shared-memory and distributed-memory parallel platforms. Attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved as the number of processors grows.

ACKNOWLEDGMENTS

We would like to acknowledge all the code and test suite developers who have contributed to the Xyce project over the years: Aaron Gibson, Alan Lundin, Antonio Gonzales, Ashley Meek, Bart van Bloemen Waanders, Brad Bond, Brian Fett, Christina Warrender, David Baur, David Day, David Shirley, Deborah Fixel, Derek Barnes, Eric Rankin, Erik Zeek, Gary Hennigan, Herman "Buddy" Watts, Jim Emery, Jonathan Woodbridge, Jonathen Kwok, Keith Santarelli, Laura Boucheron, Lawrence Musson, Lon Waters, Mary Meinelt, Michael Skoufis, Mingyu "Genie" Hsieh, Nicholas Johnson, Peter Sholander, Philip Campbell, Rachel Campbell, Randall Lober, Rebecca Arnold, Regina Schells, Richard Drake, Robert Hoekstra, Roger Pawlowski, Russell Hooper, Samuel Browne, Scott Hutchinson, Simon Zou, Smitha Sam, Steven Verzi, Tamara Kolda, Timur Takhtaganov, and Todd Coffey.

Also, thanks to Hue Lai for the original typesetting of this document in LaTeX.

Trademarks

Contact Information

Address

Electrical Models & Simulation Dept.
Sandia National Laboratories
P.O. Box 5800, MS 1177
Albuquerque, NM 87185-1177

Outside Sandia
World Wide Web `http://xyce.sandia.gov`
Email `xyce@sandia.gov`
Inside Sandia
World Wide Web `https://info-ng.sandia.gov/xyce/`
Email `xyce-sandia@sandia.gov`

CONTENTS

This page intentionally left blank.

LIST OF FIGURES

This page intentionally left blank.

LIST OF TABLES

1. INTRODUCTION

Welcome to Xyce™

The Xyce™ Parallel Electronic Simulator is a SPICE-compatible [1] [2] circuit simulator that has been written to support the unique simulation needs of electrical designers at Sandia National Laboratories. It is specifically targeted to run on large-scale parallel computing platforms, but is also available on a variety of architectures including single processor workstations. It aims to support a variety of devices and models specific to Sandia needs, as well as standard capabilities available from current commercial simulators.

1.1. Xyce Overview

The Xyce Parallel Electronic Simulator project was started in 1999 to support the simulation needs of electrical designers at Sandia National Laboratories and has evolved into a mature platform for large-scale circuit simulation.

Xyce includes several unique features. An important driver has been the need to simulate very large-scale circuits (100,000 devices or more) on the transistor level. To this end, scalable algorithms for simulating large circuits in parallel have been developed. In addition Xyce includes novel approaches to numerical kernels including model-order reduction, continuation algorithms, time-integration, nonlinear and linear solvers. Also, unlike most SPICE-based codes, Xyce uses a differential-algebraic-equation (DAE) formulation, which better isolates the device model package from solver algorithms.

1.2. Xyce Capabilities

1.2.1. Support for Large-Scale Parallel Computing

Xyce is a truly parallel simulation code, designed and written from the ground up to support large-scale parallel computing architectures with up to thousands of processors. This provides Xyce the capability to solve large circuit problems with quick enough runtimes to make these simulations practical. Xyce uses a message passing parallel implementation, allowing it to run efficiently on a variety of parallel computing platforms. These include serial, shared-memory and distributed-memory parallel. Careful attention has been paid to the specific nature of circuit-simulation problems to ensure optimal parallel efficiency, even as the number of processors increases.

1.2.2. Differential-Algebraic Equation (DAE) formulation

Xyce has been designed to use a DAE formulation. Among other advantages, this has the benefit of allowing the device models to be nearly independent of the type analysis to be performed, and allows a lot of encapsulation between the models and the solver layers of the source code. In a SPICE-based code, new device functions are created for each type of analysis, such as transient and AC analysis. With Xyce's DAE implementation, this is not necessary. The same device load functions can be used for all analysis types, resulting in faster development time for new types of analysis.

1.2.3. Device Model Support

The Xyce development team continually adds new device models to Xyce to meet the needs of Sandia users. This includes the full set of models that can be found in most SPICE-based codes. For current device availability, consult The Xyce Reference Guide [3].

1.3. Reference Guide

The Xyce User's Guide companion document, the Xyce Reference Guide [3], contains detailed information including a netlist reference for Xyce-supported input-file commands and elements; a command line reference, which describes the available command line arguments; and quick-references for users of other circuit codes, such as Orcad's PSpice [4].

1.4. How to Use this Guide

This guide is designed to enable one to quickly find the information needed to use Xyce. It assumes familiarity with basic Unix-type commands, and how Unix manages applications and files to perform routine tasks (e.g., starting applications, opening files, and saving work).

1.4.0.1. Typographical conventions

Table 1-1 defines the typographical conventions used in this guide.

Table 1-1. Xyce typographical conventions.

Notation	Example	Description
`Typewriter text`	`mpirun -np 4`	Commands entered from the keyboard on the command line or text entered in a netlist.
Bold Roman Font	Set nominal temperature using the **TNOM** option.	SPICE-type parameters used in models, etc.
Gray Shaded Text	DEBUGLEVEL	Feature that is designed primarily for use by Xyce developers.
`[text in brackets]`	`Xyce [options] <netlist>`	Optional parameters.
`<text in angle brackets>`	`Xyce [options] <netlist>`	Parameters to be inserted by the user.
`<object with asterisk>*`	`K1 <ind. 1> [<ind. n>*]`	Parameter that may be multiply specified.
`<TEXT1|TEXT2>`	`.PRINT TRAN + DELIMITER=<TAB|COMMA>`	Parameters that may only take specified values.

This page intentionally left blank.

2. INSTALLING AND RUNNING XYCE

Chapter Overview

This chapter describes the basic mechanics of installing and running Xyce. It includes the following sections:

- Section 2.1, *Xyce Installation*
- Section 2.2, *Running Xyce*

2.1. Xyce Installation

Xyce is distributed in two ways: source code and binary installers. Instructions for both methods of installation are available on the Xyce web site `http://xyce.sandia.gov`.

2.1.1. Postinstallation steps: PATH setting

In order to run Xyce, either type the path to its binary every time, or add the installed location of Xyce to the PATH variable. Do so by editing the shell start-up file (`.bashrc` if your shell is bash, `.profile` if the shell is sh, or `.cshrc` if the shell is the c-shell). Exact syntax depends on the shell used, but for bash and sh the syntax is:

```
export PATH=$PATH:/usr/local/Xyce-Release-6.9.0/bin
```

The exact path will depend on the installed version of Xyce. Look in `/usr/local` with the following command:

```
ls -l /usr/local
```

to identify the actual install directory used by the installed version of Xyce. Once entered into the start-up file, the path will be set this way at the next log in. The same command can be issued directly in the command line and it will take effect immediately.

Binary installations on Windows create a "Command Prompt" shortcut for Xyce that sets this path for you. When the command prompt is opened with this shortcut, Xyce may simply be invoked by typing its name.

2.2. Running Xyce

While it is possible to connect Xyce to graphical interfaces, such as gEDA [5] or Qucs-s [6], Xyce is not provided with any graphical user interface. It is primarily used as a command-line-only program across all supported platforms, including traditionally "GUI-centered" platforms such as Mac OS X and Microsoft Windows.

This section describes how Xyce is run from the command line, for serial and MPI parallel simulations.

2.2.1. Command Line Simulation

The syntax for running Xyce from the command line differs depending it is the serial or parallel version.

Running Xyce Assuming the `Xyce` executable in in the user's path, then the commands for running Xyce are:

- Running serial Xyce:

```
Xyce [options] <netlist filename>
```

- Running Xyce in parallel:

```
mpirun -np <# procs> Xyce [options] <netlist filename>
```

Note that `mpiexec` is sometimes provided as an alternative to `mpirun`. With Open MPI, the two commands are identical, but that is not the case for all MPI implementations.

General comments The `[options]` are the command line arguments for Xyce. For example, to log output to a file named `sample.log` type:

```
Xyce -l sample.log <netlist filename>
```

The next example runs parallel Xyce on four processors and places the results into a set of files with the "basename" of `results` . So, for example, the output from any `.PRINT TRAN` and/or `.MEASURE TRAN` lines in the netlist would be placed into the files `results.prn` and `results.mt0`, respectively (assuming a source code install):

```
mpirun -np 4 Xyce -o results <netlist filename>
```

While Xyce is running, simulation progress is output to the command line window along with any error messages.

Xyce assumes that `<netlist filename>` is either in the current working directory, or includes the path (full or relative) to the netlist file. Enclose the filename in quotation marks (`"`) if the path contains spaces. Help is accessible with the `-h` option.

Consult the documentation installed with MPI on the user's platform for more details concerning MPI options. The `-np <# procs>` denotes the number of processors to use for the simulation.
NOTE: It is critical that the number of processors used must be smaller than the number of devices and voltage nodes in the netlist.

2.2.1.1. Guidance for Running Xyce in Parallel

The basic mechanics of running Xyce in parallel have been discussed above. For general guidance regarding solver options, partitioning options, and other parallel issues, refer to chapter 11. Distributed memory circuit simulation still contains a number of research issues, so obtaining an optimal simulation in parallel is a bit of an art.

NOTE: If you are running on Linux, please see chapter 11, especially section 11.2, for critical guidance on OpenMPI options that can seriously impact the performance of parallel jobs on a multiuser system.

2.2.2. Command Line Options

Xyce supports a handful of command line options that must be given *before* the netlist filename. Table 2-1 lists Xyce core options.

Table 2-1. List of Xyce command line arguments.

Argument	Description	Usage	Default
-h	Help option. Prints usage and exits.	`-h`	-
-v	Prints the version banner and exits.	`-v`	-
-license	Prints the license text and exits.	`-license`	-
-capabilities	Prints a list of compiled-in options and exits.	`-capabilities`	-
-delim	Set the output file field delimiter.	`-delim <TAB|COMMA|string>`	-
-o	Place the output result(s) into file(s) with the specified basename and the appropriate extension (e.g., `<basename>.prn` or `<basename>.mt0`).	`-o <basename>`	output file name(s) based on netlist file name.
-l	Place the log output into specified file.	`-l <file>`	Log output sent to standard out.
-r	Output a binary rawfile.	`-r <file>`	No rawfile written.
-a	Use with `-r` to output a readable (ASCII) rawfile. Without the concurrent use of `-r`, `-a` will cause all .PRINT lines that use FORMAT=RAW to produce ASCII rawfiles.	`-r <file> -a` `-a`	Default rawfile is binary.
-nox	Use the NOX nonlinear solver.	`-nox <ON|OFF>`	on
-linsolv	Set the linear solver.	`-linsolv <KLU|KSPARSE|SUPERLU|AZTECOO|BELOS>`	klu(serial) and aztecoo(parallel)
-param	Print a terse summary of model parameters, device parameters and default values.	`-param`	-
-remeasure	Recompute .measure and/or .fft results with existing data.	`-remeasure`	-
-syntax	Check netlist syntax and exit.	`-syntax`	-
-norun	Check netlist syntax and topology and exit.	`-norun`	-
-quiet	Suppress some of the simulation-progress messages sent to stdout for transient simulations.	`-quiet`	-
-namesfile	Output a list of all solution variables generated by the netlist into `<filename>`	`-namesfile <filename>`	-
-noise_names_file	Output a list of all noise sources for all devices generated by the netlist into `<filename>`	`-noise_names_file <filename>`	-

Table 2-1. List of Xyce command line arguments.

Argument	Description	Usage	Default
-randseed	Set random number seed for expression library's random number functions and also `.SAMPLING` analysis.	`-randseed <number>`	If not provided, Xyce will select a seed using the system "time" function.
-maxord	Maximum time integration order.	`-maxord <1..5>`	-
-max-warnings	Maximum number of warning messages.	`-max-warnings <#>`	100
-jacobian_test	Jacobian matrix diagnostic.	`-jacobian_test`	-
-redefined_params	Set precendence for multiply defined parameters. Can be set to `ignore` (use last), `usefirst`, `warn` or `error`.	`-redefined_params <option>`	`ignore`, which means use the last duplicate parameter in the netlist without warning or error.
-hspice-ext	Hspice parser extensions. This will override several parser issues that are difficult to translate with scripts.	`-hspice-ext <all|separator|units|math>`	The second argument will determine which features are enabled. For details see the Xyce Reference Guide [3].

While these options are intended for general use, others may exist for new features that are disabled by default, and older, deprecated features. The Xyce Reference Guide [3] provides a comprehensive list, including trial and deprecated options.

This page intentionally left blank.

3. SIMULATION EXAMPLES WITH XYCE

Chapter Overview

This chapter provides several simple examples of Xyce usage. An example circuit is provided for each available analysis type.

- Section 3.1, *Example Circuit Construction*
- Section 3.2, *DC Sweep Analysis*
- Section 3.3, *Transient Analysis*

3.1. Example Circuit Construction

This section describes how to use Xyce to create the simple diode clipper circuit shown in figure 3-1.

Xyce only supports circuit creation via netlist editing. Xyce supports most of the standard netlist entries common to Berkeley SPICE 3F5 and Orcad PSpice. For users familiar with PSpice netlists, the Xyce Reference Guide [3] lists the differences between PSpice and Xyce netlists.

3.1.0.1. Example: diode clipper circuit

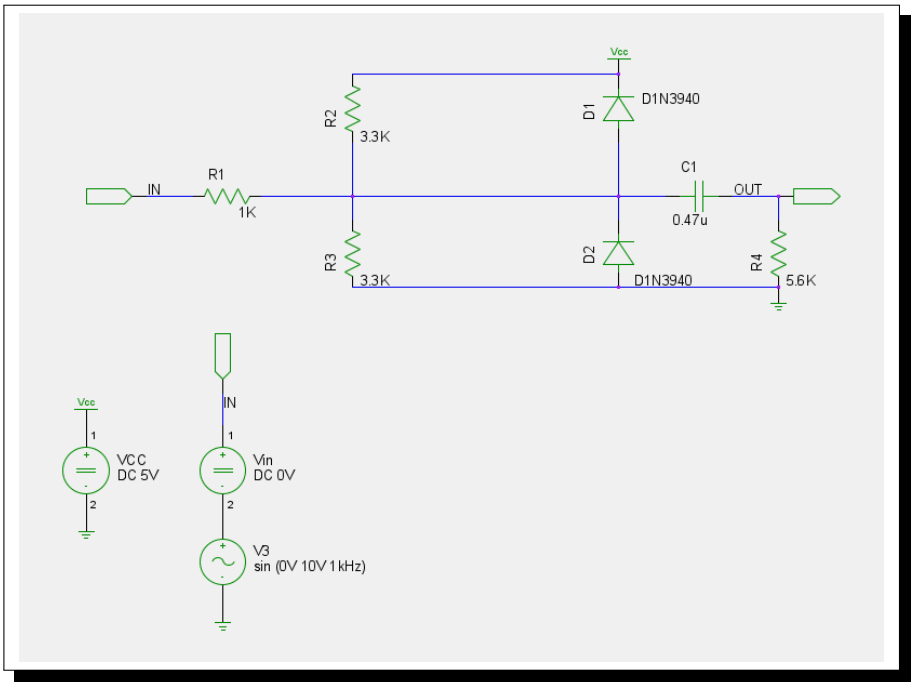

Figure 3-1. Schematic of diode clipper circuit with DC and transient voltage sources.

Using a plain text editor (e.g., vi, Emacs, Notepad), but not a word processor (e.g., OpenOffice or Microsoft Word), create a file containing the netlist of figure 3-2. For this example, the file is named `clipper.cir`

The netlist in figure 3-2 illustrates some of the syntax of a netlist input file. Netlists always begin with a title line (*e.g.* "`Diode Clipper Circuit`"), and may contain comments (lines beginning with the "`*`" character), devices, and model definitions. Netlists must always end with the "`.END`" statement.

The diode clipper circuit contains two-terminal devices (diodes, resistors, and capacitors), each of which specifies two connecting nodes and either a model (for the diode) or a value (resistance or capacitance). The netlist of figure 3-2 describes the circuit shown in the schematic of figure 3-1

This netlist file is not yet complete and will not run properly using Xyce (see section 2.2 for instructions on running Xyce) as it lacks an analysis statement. This chapter later decribes how to add the appropriate analysis statement and run the diode clipper circuit.

```
Diode Clipper Circuit
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 3-2. Diode clipper circuit netlist

3.2. DC Sweep Analysis

This section includes an example of DC sweep analysis using Xyce. The DC response of the clipper circuit is obtained by sweeping the DC voltage source (`Vin`) from -10 to 15 volts in one-volt steps. Chapter 7.2 provides more details about DC analysis, as does the Xyce Reference Guide [3].

3.2.0.1. Example: DC sweep analysis

To set up and run a DC sweep analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file (`clipper.cir`) using a standard text editor (*e.g.* vi, Emacs, Notepad, *etc.*).

2. Enter the analysis control statement (as shown in the netlist in figure 3-4):

```
.DC VIN -10 15 1
```

3. Enter the output control statement:

```
.PRINT DC V(3) V(2) V(4)
```

4. Save the netlist file and run Xyce on the circuit. For example, to run serial Xyce:

```
Xyce clipper.cir
```

5. Open the results file (`clipper.cir.prn`) and examine (or plot) the output voltages that were calculated for nodes 3 (Vin), 2 and 4 (Out). Figure 3-3 shows the output plotted as a function of the swept variable Vin.

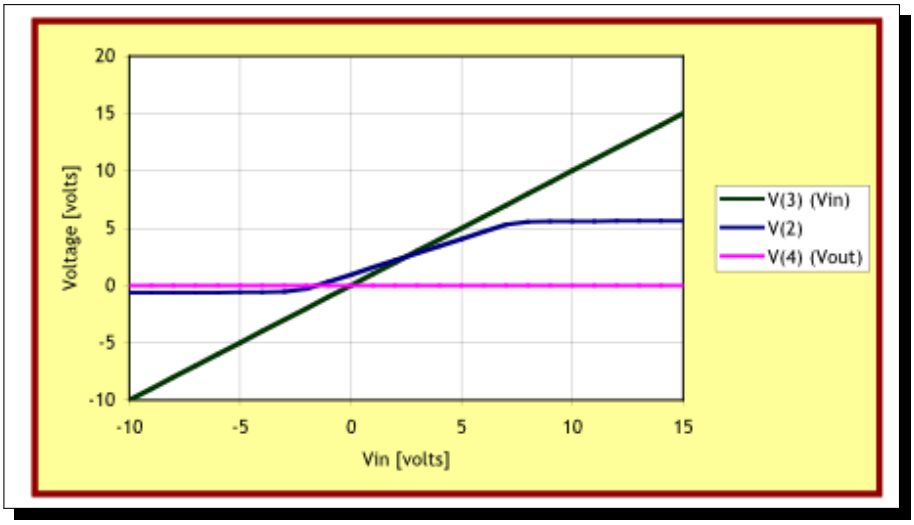

Figure 3-3. DC sweep voltages at Vin, node 2, and Vout

```
Diode Clipper Circuit with DC sweep analysis statement
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 3-4. Diode clipper circuit netlist for DC sweep analysis

3.3. Transient Analysis

This section contains an example of transient analysis in Xyce. In this example, the DC analysis of the diode clipper circuit of the previous section has been modified so that the input voltage source (`Vin`) is a time-dependent sinusoidal input source. The frequency of `Vin` is 1 kHz, and has an amplitude of 10 volts. For more details about transient analysis see chapter 7.3, or the Xyce Reference Guide [3].

3.3.0.1. Example: transient analysis

To set up and run a transient analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file file (`clipper.cir`) using a standard text editor (*e.g.* VI, Emacs, Notepad, *etc.*).

2. Remove DC analysis and output statements if added in the previous example (figure 3-4).

3. Enter the analysis control (as shown in the netlist in figure 3-5):

```
.TRAN 2ns 2ms
```

4. Enter the output control statement:

```
.PRINT TRAN V(3) V(2) V(4)
```

5. Modify the input voltage source (`Vin`) to generate the sinusoidal input signal:

```
VIN 3 0 SIN(0V 10V 1kHz)
```

6. At this point, the netlist should look similar to the netlist in figure 3-5. Save the netlist file and run Xyce on the circuit. For example, to run serial Xyce:

```
Xyce clipper.cir
```

7. Open the results file and examine (or plot) the output voltages for nodes 3 (`Vin`), 2, and 4 (`Out`). The plot in figure 3-6 shows the output plotted as a function of time.

Figure 3-5 shows the modified netlist and figure 3-6 shows the corresponding results.

```
Diode clipper circuit with transient analysis statement
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 3-5. Diode clipper circuit netlist for transient analysis

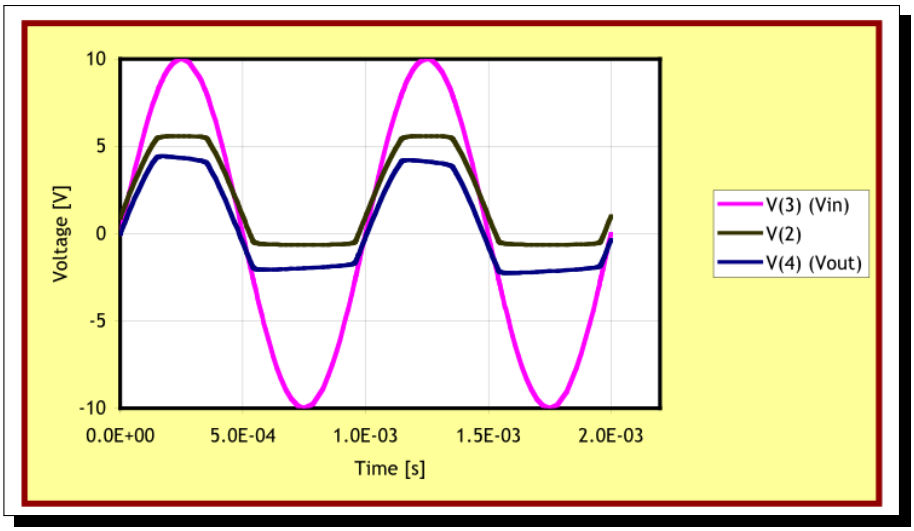

Figure 3-6. Sinusoidal input signal and clipped outputs

Chapter Overview

This chapter contains introductory material on netlist syntax and usage. Sections include:

- Section 4.1 *General Overview*
- Section 4.2 *Devices Available for Simulation*
- Section 4.3 *Parameters and Expressions*
- Section 4.4 *Device Multiplier* `M`

4.1. General Overview

4.1.1. Introduction

Using a netlist to describe a circuit for Xyce is the primary method for running a circuit simulation. Netlist support within Xyce largely conforms to that used by Berkeley SPICE 3F5 with several new options for controlling functionality unique to Xyce.

In a netlist, the circuit is described by a set of *element lines* defining circuit elements and their associated parameters, the circuit topology (i.e., the connection of the circuit elements), and a variety of control options for the simulation. The first line in the netlist file must be a title and the last line must be "`.END`". Between these two constraints, the order of the statements is irrelevant.

4.1.2. Nodes

Nodes and elements form the foundation for the circuit topology. Each node represents a point in the circuit that is connected to the leads of multiple elements (devices). Each lead of every element is connected to a node, and each node is connected to multiple element leads.

A node is simply a named point in the circuit. The naming of normal nodes is mainly known within the level of circuit hierarchy where they appear. Nodes can be passed into subcircuits through an argument list, and in this manner subcircuits are given access to nodes from the upper-level circuit. Historicaly, this is how nodes are passed thru the circuit hierarchy in most circuit simulators, and this is the convention used by most circuit netlists.

However, Xyce also has the capability for fully resolved, internal subcircuit nodes to be accessed in the top level circuit. Examples of this usage are as follows:

```
* example usage of fully resolved subcircuit nodes
Vin 1 0 1.0
Rin 1 2 1.0
X1 2 3 test
Rout 3 0 1.0

.subckt test A B
Rt1 A testNode 1.0
Rt2 testNode B 1.0
.ends

* this works:
Btest1 4 0 V = V(X1:testNode)
Rtest1 4 0 1.0

* this also works:
Itest2 0 5 1.0
Rtest2 X1:testNode 5 1.0
Rtest3 X1:testNode 0 1.0
```

The fully resolved name is specific to a single subcircuit instance. Also, note that the field separator in this example is the colon (`:`), but Xyce can optionally use other characters as the separator, including a period (`.`) . For information about specifying a different separator, see section 2.2.2, about Xyce command line arguments, or the Xyce Reference Guide [3] for even more details.

4.1.2.1. Global Nodes

For cases where a particular node is used widely throughout various subcircuits it can be more convenient to use a global node, which is referenced by the same name throughout the circuit. This is often the case for power rails such as `VDD` or `VSS`.

Global nodes start with the prefix `$G`. Examples of global node names would be: `$`G_VDD or `$G1`. Nodes or global nodes require no declaration, as they are declared implicitly by appearing in *element lines*.

For compatibility with HSPICE, the `.GLOBAL` command can be used to define global nodes that do not start with the prefix "$G". Consult the Xyce Reference Guide [3] for more details.

4.1.3. Elements

An *element line* defines each circuit element instance. While each element type determines the specific format, the general format is given by:

```
<type><name> <node information> <element information...>
```

The `<type>` must be a letter (A through Z) with the `<name>` immediately following. For example, `RARESISTOR` specifies a device of type "R" (for "Resistor") with a name `ARESISTOR`. Nodes are separated by spaces, and additional element information required by the device is given after the node list as described in the Netlist Reference section of the Xyce Reference Guide [3]. Xyce ignores character case when reading a netlist such that `RARESISTOR` is equivalent to `raresistor`. The only exception to this case insensitivity occurs when including external files in a netlist where the filename specified in the netlist must have the same case as the actual filename.

A number field may be an integer or a floating-point value. Either one may be followed by one of the following scaling factors shown in Table 4-1:

Table 4-1. Scaling factors.

Symbol	Equivalent Value
T	10^{12}
G	10^{9}
Meg	10^{6}
X	10^{6}
K	10^{3}
mil	25.4^{-6}
m	10^{-3}
u (μ)	10^{-6}
n	10^{-9}
p	10^{-12}
f	10^{-15}

Node information is given in terms of node names, which are character strings. One key requirement is that the ground node is named '0'. (Note: Consult the Xyce Reference Guide [3] for more details on allowed characters in both node names and device names.) There is one restriction on the circuit topology: there can be no loop of voltage sources and/or inductors. In addition to this requirement, the following additional topology constraints are highly recommended:

- Every node has a DC path to ground.
- Every node has at least two connections (with the exception of unterminated transmission lines and MOSFET substrate nodes).

While Xyce can theoretically handle netlists that violate the above two constraints, such topologies are typically the result of human error in creating a netlist file, and will often lead to convergence failures. Chapter 14 provides more information on this topic.

The following line provides an example of an element line that defines a resistor between nodes 1 and 3 with a resistance value of 10kΩ.

Example: `RARESISTOR 1 3 10K`

4.1.3.1. Title, Comments and End

The first line of the netlist is the title line of the netlist. This line is treated as a comment even if it does not begin with an asterisk. It is a common mistake to forget the meaning of this first line and begin the circuit elements on the first line; doing so will probably result in a parsing error.

Example: `Test RLC Circuit`

The "`.END`" line must be the last line in the netlist.

Example: `.END`

Comments are supported in netlists and are indicated by placing an asterisk at the beginning of the comment line. They may occur anywhere in the netlist *but* they must be at the beginning of a line. Xyce also supports *in-line* comments. An in-line comment is designated by a semicolon and may occur on any line. Xyce ignores everything after a semicolon. Xyce considers lines beginning with leading whitespace as comments unless the first character after the whitespace is a + symbol, in which case it treats the line as a continuation.

Example: `* This is a netlist comment.`

Example: **WRONG:`.DC .... * This type of in-line comment is *not supported*.`**

Example: `.DC .... ; This type of in-line comment is supported.`

4.1.3.2. Continuation Lines

Continuation lines begin with a + symbol, and their contents are appended to those of the previous line. If the previous line or lines were comments, the continuation line is appended to the first noncomment line preceding it. Continuation lines can have leading whitespace before the + symbol.

4.1.3.3. Netlist Commands

Command elements are used to describe the analysis being defined by the netlist. Examples include analysis types, initial conditions, device models, and output control. The Xyce Reference Guide [3] contains a reference for these commands.

Example: `.PRINT TRAN V(Vout)`

4.1.3.4. Analog Devices

Xyce-supported analog devices include most of the standard circuit components normally found in circuit simulators, such as SPICE 3F5, PSpice, etc., plus several Sandia-specific devices.

Example: `D_CR303 N_0065 0 D159700`

The Xyce Reference Guide [3] provides more information concerning its supported devices.

4.2. Devices Available for Simulation

The analog devices available in Xyce include all of the standard circuit components needed for most analog circuits. User-defined models may be implemented using the `.MODEL` (model definition) statement, and macromodels can be created as subcircuits using the `.SUBCKT` (subcircuit) statement.

In addition to the traditional analog devices, which are modeled in Xyce by sets of coupled differential algebraic equations, Xyce also supports digital behavioral models. The digital devices are behavioral devices in the sense that they rely on truth tables to determine their outputs. Once one or more of a digital device's inputs go past user-specified thresholds, its outputs will change according to its truth table after a user-specified delay time. The impedance characteristics of the inputs and outputs of the digital devices are modeled with RC time constants.

Xyce also include TCAD devices, which solve a coupled set of partial differential equations (PDEs), discretized on a mesh. The use of these devices are described in detail in Chapter 15.

The device element statements in the netlist always start with the name of the individual device instance. The first letter of the name determines the device type. The format of the subsequent information depends on the device type and its parameters. Table 4-2 provides a quick reference to the analog devices and their netlist formats as supported by Xyce. Except where noted, the devices are based upon those found in [7]. The Xyce Reference Guide [3] provides a complete description of the syntax for all the supported devices.

Table 4-2. Analog Device Quick Reference.

Device Type	Letter	Typical Netlist Format
Nonlinear Dependent Source (B Source)	B	B<name> <+ node> <- node> + <I or V>={<expression>}
Capacitor	C	C<name> <+ node> <- node> [model name] <value> + [IC=<initial value>]
Diode	D	D<name> <anode node> <cathode node> + <model name> [area value]
Voltage Controlled Voltage Source	E	E<name> <+ node> <- node> <+ controlling node> + <- controlling node> <gain>
Current Controlled Current Source	F	F<name> <+ node> <- node> + <controlling V device name> <gain>
Voltage Controlled Current Source	G	G<name> <+ node> <- node> <+ controlling node> + <- controlling node> <transconductance>
Current Controlled Voltage Source	H	H<name> <+ node> <- node> + <controlling V device name> <gain>
Independent Current Source	I	I<name> <+ node> <- node> [[DC] <value>] + [AC [magnitude value [phase value]]] + [transient specification]
Mutual Inductor	K	K<name> <inductor 1> [<ind. n>*] + <linear coupling or model>
Inductor	L	L<name> <+ node> <- node> [model name] <value> + [IC=<initial value>]

Table 4-2. Analog Device Quick Reference.

Device Type	Letter	Typical Netlist Format
JFET	J	J<name> <drain node> <gate node> <source node> + <model name> [area value]
MOSFET	M	M<name> <drain node> <gate node> <source node> + <bulk/substrate node> [SOI node(s)] + <model name> [common model parameter]*
Lossy Transmission Line (LTRA)	O	O<name> <A port (+) node> <A port (-) node> + <B port (+) node> <B port (-) node> + <model name>
Bipolar Junction Transistor (BJT)	Q	Q<name> <collector node> <base node> + <emitter node> [substrate node] + <model name> [area value]
Resistor	R	R<name> <+ node> <- node> [model name] <value> + [L=<length>] [W=<width>]
Voltage Controlled Switch	S	S<name> <+ switch node> <- switch node> + <+ controlling node> <- controlling node> + <model name>
Generic Switch	S	S<name> <+ switch node> <- switch node> + <model name> CONTROL=\{expression\}
Transmission Line	T	T<name> <A port + node> <A port - node> + <B port + node> <B port - node> + <ideal specification>
Digital Devices	U	U<name> <type> <digital power node> + <digital ground node> [node]* <model name>
Independent Voltage Source	V	V<name> <+ node> <- node> [[DC] <value>] + [AC [magnitude value [phase value]]] + [transient specification]
Port Device	P	P<name> <+ node> <- node> [[DC] <value>] + port=port number [Z0 = value] + [AC [magnitude value [phase value]]] + [transient specification]
Subcircuit	X	X<name> [node]* <subcircuit name> + [PARAMS:[<name>=<value>]*]
Current Controlled Switch	W	W<name> <+ switch node> <- switch node> + <controlling V device name> <model name>
Digital Devices, Y Type (deprecated)	Y<type>	Y<type> <name> [node]* <model name>
PDE Devices	YPDE	YPDE <name> [node]* <model name>
Accelerated masses	YACC	YACC <name> <acceleration> <velocity> <position> + [x0=<initial position>] [v0=<initial velocity>]
Linear Device	YLIN	YLIN <name> <+ node> <- node> <model name>
Memristor Device	YMEMRISTOR	YMEMRISTOR <name> <+ node> <- node> <model name>
MESFET	Z	Z<name> <drain node> <gate node> <source node> + <model name> [area value]

4.3. Parameters and Expressions

In addition to explicit values, the user may use parameters and expressions to symbolize numeric values in the circuit design.

4.3.1. Parameters

A parameter is a symbolic name representing a numeric value. Parameters must start with a letter or underscore. The characters after the first can be letter, underscore, or digits. Once a parameter is defined (by having its name declared and having a value assigned to it) at a particular level in the circuit hierarchy, it can be used to represent circuit values at that level or any level directly beneath it in the circuit hierarchy. One way to use parameters is to apply the same value to multiple part instances.

4.3.2. How to Declare and Use Parameters

For using a parameter in a circuit, one must:

- Define the parameter using a `.PARAM` statement within a netlist
- Replace an explicit value with the parameter in the circuit

Xyce reserves the following keywords that may not be used as parameter names:

- `Time`
- `Freq`
- `Hertz`
- `Vt`
- `Temp`
- `Temper`
- `GMIN`

`Time`, `TEMP`, `Vt`, `FREQ` and `GMIN` are reserved and defined as “special variables”, and may be used in B source expressions. Their values are set by the simulator. `HERTZ` is a synonym for `FREQ` and `TEMPER` is a synonym for `TEMP`.

4.3.2.1. Example: Declaring a parameter

1. Locate the level in the circuit hierarchy at which the `.PARAM` statement declaring a parameter will be placed. To declare a parameter capable of being used anywhere in the netlist, place the `.PARAM` statement at the top-most level of the circuit.
2. Name the parameter and give it a value. The value can be numeric or given by an expression:

```
.SUBCKT subckt1 n1 n2 n3
.PARAM res = 100
*
* other netlist statements here
*
.ENDS
```

3. NOTE: The parameter *res* can be used anywhere within the subcircuit `subckt1`, including subcircuits defined within it, but cannot be used outside of `subckt1`.

4.3.2.2. Example: Using a parameter in the circuit

1. Locate the numeric value (a device instance parameter value, model parameter value, etc.) that is to be replaced by a parameter.
2. Replace the numeric value with the parameter name contained within braces ({}) as in:

```
R1 1 2 {res}
```

NOTE: Ensure the value being replaced remains accessible within the current hierarchy level.

4.3.3. Global Parameters

A normal parameter defined using `.PARAM` at the main circuit level will have global scope. However, such parameters may be redefined within a subcircuit, which would change the value in the subcircuit and below.

A global parameter, defined using `.global_param`, differs from a normal parameter in that it can only be defined at the main circuit level. Both types of parameters can be modified by netlist commands such as `.STEP` and `.SAMPLING`, as long as they are globally scoped.

Examples of some global parameter usages are:

```
.param dTdt=100
.global_param T={27+dTdt*time}
R1  1  2  RMOD TEMP={T}

or

.global_param T=27
R1  1  2  RMOD TEMP={T}
C1  1  2  CMOD TEMP={T}
.step T 20 50 10
```

In these examples, T is used to represent an environmental variable that changes.

4.3.4. Expressions

In Xyce, an expression is a mathematical relationship that may be used any place one would use a number (numeric or boolean). To use an expression in a circuit netlist:

1. Locate the value to be replaced (component, model parameter, etc.).
2. Substitute the value with an expression using the `{}` syntax:

   {*expression*}

   where *expression* can contain any of the following:

   - Arithmetic and logical operators.
   - Arithmetic, trigonometric, or SPICE-type functions.
   - User-defined functions.
   - User-defined parameters within scope.
   - Literal operands.

   The braces (`{}`) instruct Xyce to evaluate the expression and use the resulting value. Alternatively, for netlist compatibility with other simulators expressions may be enclosed in single quotation marks instead (`'`). In some circumstances, (also for compatibility) expressions don't need to be surrounded by either braces or single quotes at all. These circumstances include `.param` and `.global_param` expressions, as well as device instance parameters that use an equals sign. Curly braces are the original expression delimeter used by Xyce and it is recommended that the braces be used in netlists written specifically for Xyce.

   Additional time-dependent constructs are available in expressions used in analog behavioral modeling sources (see chapter 6). Complete documentation of supported functions and operators may be found in the Xyce Reference Guide [3].

4.3.4.1. Example: Using an expression

Scaling the DC voltage of a 12*V* independent voltage source, designated `VF`, by some factor can be accomplished by the following netlist statements (in this example the factor is 1.5):

```
.PARAM FACTORV=1.5
VF 3 4 {FACTORV*12}
```

Xyce will evaluate the expression to $12 * 1.5$ or 18 volts.

4.4. Device Multiplier `M`

Many device models support a special parameter, `M`, which is used to approximate multiple identical devices placed in parallel. It can also be thought of as a current multiplier, and as such is not required to be an integer.

The following devices support the multiplier `M`:

Table 4-3. Devices supporting multipliers

Device Type	Letter
Capacitor	`C`
Diode	`D`
Inductor	`L`
MOSFET	`M`
Bipolar Junction Transistor (BJT)	`Q`
Resistor	`R`
Subcircuit	`X`

The handling of the multiplier parameter in subcircuits, represented by the letter `X`, is a special case. This is described in section 5.5.

This page intentionally left blank.

Chapter Overview

This chapter provides model examples and summarizes ways to create and modify models. Sections include:

- Section 5.1, *Model Definition*
- Section 5.2, *Subcircuit Creation*
- Section 5.3, *Model Organization*
- Section 5.4, *Model Interpolation*
- Section 5.5, *Subcircuit Multiplier* `M`

5.1. Model Definitions

A model describes the electrical performance of a *part*, such as a specific vendor's version of a 2N2222 transistor. To simulate a part requires specification of *simulation properties*. These properties define the model of the part.

Depending on the given device type and the requirements of the circuit design, a model is specified using a model parameter set, a subcircuit netlist, or both.

In general, *model parameter sets* define the parameters used in ideal models of specific device types, while *subcircuit netlists* allow the user to combine ideal device models to simulate more complex effects. For example, one could simulate a bipolar transistor using the Xyce BJT device by specifying model parameters extracted to fit the simulation behavior to the behavior of the part used. One could also develop a subcircuit macro-model of a capacitor that adds effects such as lead inductance and resistance to the basic capacitor device.

Both methods of defining a model use a netlist format, with precise syntax rules. In this section we give an overview of how to define model parameter sets in Xyce. A subsequent subsection will provide a similar overview of how to define subcircuit models. For full details, consult the Xyce Reference Guide [3].

Defining models using model parameters Although Xyce has no built-in part models, models can be defined for a device by changing some or all of the *model parameters* from their defaults via the `.MODEL` statement. For example:

```
M5 3 2 1 0 MLOAD1
.MODEL MLOAD1 NMOS (LEVEL=3 VTO=0.5 CJ=0.025pF)
```

This example defines a MOSFET device `M5` that is an instance of a part described by the model parameter set `MLOAD1`. The `MLOAD1` parameter set is defined in the `.MODEL` statement.

Most device types in Xyce support some form of model parameters. Consult the Xyce Reference Guide [3] for the model parameters supported by each device type.

Defining models using subcircuit netlists In Xyce, models may also be defined using the `.SUBCKT`/`.ENDS` subcircuit syntax. This syntax allows the creation of *Netlists*, which define the configuration and function of the part, and the use of *Variable input parameters*, which can be used to create device-specific implementations of the model. The `.SUBCKT` syntax, and an example of how to use `.SUBCKT` to implement a model, is given in Section 5.2.

5.2. Subcircuit Creation

A subcircuit can be created within Xyce using the `.SUBCKT` keyword. The `.ENDS` keyword is used to mark the end of the subcircut. All the lines between the two keywords are considered to be part of the subcurcuit. Figure 5-1 provides an example of how a subcircuit is defined and used.

```
****other devices
X5 5  6  7  8 l3dsc1 PARAMS: ScaleFac=2.0
X6 9 10 11 12 l3dsc1
****more netlist commands

*** SUBCIRCUIT: l3dsc1
*** Parasitic Model: microstrip
*** Only one segment
.SUBCKT l3dsc1 1 3 2 4 PARAMS: ScaleFac=1.0
C01 1 0 4.540e-12
RG01 1 0 7.816e+03
L1 1 5 3.718e-08
R1 5 2 4.300e-01
C1 2 0 4.540e-12
RG1 2 0 7.816e+03
C02 3 0 4.540e-12
RG02 3 0 7.816e+03
L2 3 6 3.668e-08
R2 6 4 4.184e-01
C2 4 0 4.540e-12
RG2 4 0 7.816e+03
CM012 1 3 5.288e-13
KM12 L1 L2 2.229e-01
CM12 2 4 {5.288e-13*ScaleFac}
.ENDS
```

Figure 5-1. Example subcircuit model.

In this example, a subcircuit model named `l3dsc1`, which implements one part of a microstrip transmission line, is defined between the `.SUBCKT`/`.ENDS` lines; and two different instances of the subcircuit are used in the `X` lines. This somewhat artificial example shows how input parameters are used, where the last capacitor in the subcircuit is scaled by the input parameter `ScaleFac`. If input parameters are not specified on the `X` line (as in the case of device `X6`), then the default values specified on the `.SUBCKT` line are used. Non-default values are specified on the `X` line using the `PARAMS:` keyword. Consult the Xyce Reference Guide [3] for precise syntax.

In addition to devices, a subcircuit may contain definitions, such as models via the `.MODEL` statement, parameters via the `.PARAM` statement, and functions via the `.FUNC` statement. Xyce also supports the definition of one or more subcircuits within another subcircuit. Subcircuits can be nested to an arbitrary extent, where one subcircuit can contain another subcircuit, which can contain yet another subcircuit, and so on.

The creation of nested subcircuits requires an understanding of "scope," such that each subcircuit defines the scope for the definitions it contains. That is, *the definitions contained within a subcircuit can be used within that subcircuit and within any subcircuit it contains, but not at any higher level.* Definitions occurring in the main circuit have global scope and can be used anywhere in the circuit. A name, such as a model, parameter, function, or subcircuit name, occurring in a definition at one level of a circuit hierarchy can be redefined at any lower level contained directly by that subcircuit. In this case, the new definition applies at the given level and those below.

5.2.1. Examples of Scoping for Parameters, Models and Functions

The idea of "scope" may be best illustrated by examples. This section gives example for parameters and models. However, this discussion also applies to functions (defined with `.FUNC` statements).

In the netlist provided in Figure 5-2, the model named `MOD1` can be used in subcircuits `SUB1` and `SUB2`, but not in the subcircuit `SUB3`. The parameter `P1` has a value of 10 in subcircuit `SUB1` and a value of 20 in subcircuit `SUB2`. In subcircuit `SUB3`, `P1` has no meaning. In addition, `MOD1` and `P1` would have no meaning at the main circuit level.

```
.SUBCKT SUB1 1 2 3 4
.MODEL MOD1 NMOS(LEVEL=2)
.PARAM P1=10
*
* subcircuit devices omitted for brevity
*
.SUBCKT SUB2 1 3 2 4
.PARAM P1=20
*
* subcircuit devices omitted for brevity
*
.ENDS
.ENDS

.SUBCKT SUB3 1 2 3 4
*
* subcircuit devices omitted for brevity
*
.ENDS
```

Figure 5-2. Example subcircuit hierarchy, with scoping.

In the netlist provided in Figure 5-3, the parameters `P2` and `P3` are defined in the main circuit. So, `P2` is accessible in subcircuit `SUB4`, and has a value of 5 there. The parameter `P3` was redefined in the context of subcircuit `SUB4`. So, `P3` has a value of 10 in the main circuit, and a value of 15 in the context of subcircuit `SUB4` and any subcircuit subsequently defined within subcircuit `SUB4`.

```
* parameters defined in main circuit
.PARAM P2=5
.PARAM P3=10

.SUBCKT SUB4 1 2 3 4
.PARAM P3=15
*
* subcircuit devices omitted for brevity
*
.ENDS
```

Figure 5-3. Example subcircuit, with parameter definition override.

In the netlist provided in Figure 5-4, the definition of subcircuit `SUB5` defines the argument `A1`. The subcircuit instance `X1` would use the default value of 5 for `A1`. So the resistance value of device `X1:R1` would be 5. The subcircuit instance `X2` uses the specified `A1` value of 10. So the resistance value of device `X2:R1` would be 10. Another key point about "scope" is that node, device, and model names are scoped to the subcircuit in which they are defined. So, It is allowable to use names in a subcircuit that has been previously used in either the main circuit netlist or in other subcircuit definitions. When the subcircuits are flattened (expanded to become part of the main netlist), all of their names are given prefixes via their subcircuit instance names. For example, device `R1` in subcircuit `X1` becomes the unique device name X1:R1 after expansion.

```
* subcircuit instance lines
X1 a b  SUB5
X2 e f SUB5 PARAMS: A1=10
R1 1 2 2

* Note use of {} around A1 and FSIN parameters in
* the body of the subcircuit definition.
.SUBCKT SUB5 1 2 PARAMS: A1=5 FSIN=1
VSIN 1 g SIN(0 1V {FSIN} 0)
R1 g 2 {A1}
.ENDS
```

Figure 5-4. Example subcircuit, with PARAMS arguments.

5.3. Model Organization

While it is always possible to make a self-contained netlist in which all models for all parts are included along with the circuit definition, Xyce provides a simple mechanism to conveniently organize frequently used models into separate model libraries. Models are simply collected into model library files, and then accessed by netlists as needed by inserting an `.INCLUDE` directive. This section describes that process in detail.

5.3.1. Model Libraries

Device model and subcircuit definitions may be organized into model libraries as text files (similar to netlist files) with one or more model definitions. Many users choose to name model library files ending with `.lib`, but they may be named using any convention.

In general, most users create model libraries files that include similar model types. In these files, the *header comments* describe the models therein.

5.3.2. Model Library Configuration using `.INCLUDE`

Xyce uses model libraries by inserting an `.INCLUDE` statement into a netlist. Once a file is included, its contents become available to the netlist just as if the entire contents had been inserted directly into the netlist.

As an example, one might create the following model library file called `bjtmodels.lib`, containing `.MODEL` statements for common types of bipolar junction transistors:

```
*bjtmodels.lib
* Bipolar transistor models
.MODEL Q2N2222 NPN (Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=5 Ne=1.307
+  Ise=14.34f Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1
+  Cjc=7.306p Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75
+  Tr=46.91n Tf=411.1p Itf=.6 Vtf=1.7 Xtf=3 Rb=10)

.MODEL 2N3700 NPN (IS=17.2E-15 BF=100)

.MODEL 2N2907A PNP (IS=1.E-12 BF=100)
```

The models `Q2N2222`, `2N3700` and `2N2907A` could then be used in a netlist by including the `bjtmodels.lib` file.

```
.INCLUDE "bjtmodels.lib"
Q1 1 2 3 Q2N2222
Q2 5 6 7 2N3700
Q3 8 9 10 2N2907A
*other netlist entries
.END
```

Because the contents of an included file are simply inserted into the netlist at the point where the .INCLUDE statement appears, the scoping rules for .INCLUDE statements are the same as for other types of definitions as outlined in the preceding subsections.

NOTE: The path to the library file is assumed to be relative to the execution directory, but absolute pathnames are permissible. The entire file name, including its "extension" must be specified. There is no assumed default extension.

5.3.3. Model Library Configuration using .LIB

An alternative technique for organizing model libraries employs the .LIB command. With .LIB, a library file can contain multiple versions of a model and specific versions may be selected at the top level using a keyword on the .LIB line.

There are two different uses for the .LIB command. In the main netlist, .LIB functions in a similar manner to .INCLUDE: it reads in a file. Inside that file, .LIB and .ENDL are used to specify blocks of model code that may be included independently of other parts of the same file.

As an example, if you had two different 2N2222 transistor models extracted at different **TNOM** values, you could define them in a model library inside .LIB/.ENDL pairs:

```
* transistors.lib file
.lib roomtemp
.MODEL Q2N2222 NPN (TNOM=27 Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=5 Ne=1.307
+  Ise=14.34f Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1
+  Cjc=7.306p Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75
+  Tr=46.91n Tf=411.1p Itf=.6 Vtf=1.7 Xtf=3 Rb=10)
.endl

.lib hightemp
.MODEL Q2N2222 NPN (TNOM=55 [...parameters omitted for brevity...])
.endl
```

Note that both models are given identical names, but are enclosed within .LIB/.ENDL pairs with different names. When this file is used in a netlist, a specific model can be used by specifying it on the .LIB line in the main netlist.

```
*This netlist uses only the high temperature model from the library
.lib transistors.lib hightemp
Q1 collector base emitter Q2N2222
[...]
```

The exact format and usage of the .LIB command is documented in the Xyce Reference Guide [3].

5.4. Model Interpolation

Traditionally, SPICE simulators handle thermal effects by coding the temperature dependence of model parameters into each device. These expressions modify the nominal device parameters given in the `.MODEL` card when the ambient temperature is not equal to **TNOM**, the temperature at which the nominal device parameters were extracted.

These temperature correction equations may be reasonable at temperatures close to **TNOM**, but Sandia users of Xyce have found them inadequate when simulations must be performed over a wide range of temperatures. To address this inadequacy, Xyce implements a model interpolation option that allows the user to specify multiple `.MODEL` cards, each extracted from real device measurements at a different **TNOM**. From these model cards, Xyce will interpolate parameters based on the ambient temperature using either piecewise linear or quadratic interpolation.

Interpolation of models is accessed through the model parameter **TEMPMODEL** in the models that support this capability. In the netlist, a base model is specified, and is followed by multiple models at other temperatures.

Interpolation of model cards in this fashion is implemented in the BJT level 1, JFET, MESFET, and MOSFETS levels 1-6, 10, and 18.

The use of model interpolation is best shown by example:

```
Jtest 1a 2a 3 SA2108 TEMP= 40
*
.MODEL SA2108 PJF ( TEMPMODEL=QUADRATIC TNOM = 27
+ LEVEL=2 BETA= 0.003130 VTO = -1.9966 PB = 1.046
+ LAMBDA = 0.00401 DELTA = 0.578; THETA = 0;
+ IS = 1.393E-10          RS = 1e-3)
*
.MODEL SA2108 PJF ( TEMPMODEL=QUADRATIC TNOM = -55
+ LEVEL=2 BETA = 0.00365 VTO = -1.9360 PB = 0.304
+ LAMBDA = 0.00286 DELTA = 0.2540 THETA = 0.0
+ IS = 1.393E-10 RD = 0.0 RS = 1e-3)
*
.MODEL SA2108 PJF ( TEMPMODEL=QUADRATIC TNOM = 90
+ LEVEL=2 BETA = 0.002770 VTO = -2.0350 PB = 1.507
+ LAMBDA = 0.00528 DELTA = 0.630 THETA = 0.0
+ IS = 1.393E-10          RS = 5.66)
```

Note that the model names are all identical for the three `.MODEL` lines, and that they all specify `TEMPMODEL=QUADRATIC`, but with different **TNOM**. For parameters that appear in all three `.MODEL` lines, the value of the parameter will be interpolated using the `TEMP=` value in the device line in the first line, which is 40°C in this example. For parameters that are not interpolated, such as **RD**, it is not necessary to include these in the second and third `.MODEL` lines.

The only valid arguments for **TEMPMODEL** are **QUADRATIC** and **PWL** (piecewise linear). The quadratic method includes a limiting feature that prevents the parameter value from exceeding the range of values specified in the `.MODEL` lines. For example, the **RS** value in the example would take on negative values for most of the interval between -55 and 27, as the value at 90 is very high. This truncation is

necessary as parameters can easily take on values (such as the negative resistance of **RS** in this example) that will cause a Xyce failure.

With the BJT parameters **IS** and **ISE**, interpolation is done not on the parameter itself, but on the the log of the parameter. This provides excellent interpolation of these two parameters, that vary over many orders of magnitude, for quadratic and piecewise linear temperature dependence.

The interpolation scheme used for model interpolation bases the interpolation on the difference between the ambient temperature and the **TNOM** value of the first model card in the netlist, which can sometimes lead to poorly conditioned interpolation. Thus it is often best that the first model card in the netlist be the one that has the "middle" **TNOM**, as in the example above. This assures that no matter where in the range of temperature values the ambient temperature lies, it is a minimal distance from the base point of the interpolation.

5.5. Subcircuit Multiplier M

Similar to many device models, subcircuits support a scalar multiplier, `M` on the instance line. The usage syntax is similar to that of subcircuit parameters, except that it is implicitly applied to all devices (see table 4-3) in the subcircuit instance which support multipliers. If the devices in a subcircuit already have multiplier parameters specified, the value is multiplied by the subcircuit instance value. This is applied recursively for nested subcircuits.

Note that it is not necessary to specify it as a parameter on the `.subckt` line. Also, the multiplier is *not* considered to be a normal user-defined parameter that can be used in expressions.

A very simple example of a subcircuit instance multiplier usage is given in figure 5-5. In this example, a multiplier, `M=5`, is given on the subcircuit line, `Xtest`. This multiplier is automatically applied to the resistor `R1` inside the `Xtest` instance, so the effective multiplier on `Xtest:R1` is `M=5`.

```
Xtest 2 3 testA M=5
.subckt testB A B
R1 A B 0.5
.ends
```

Figure 5-5. Example subcircuit using a multiplier.

An example of nested subcircuits using instance multipliers is given in figure 5-6. In this example, multipliers are specified at each level of the hierarchy, including on the resistor, `R1`. As each multiplier is applied multiplicatively, the effective multiplier on `Xtest1:Xtest:R1` is `M=4*3*5=60`.

```
Xtest1 2 3 testA M=5

.subckt testA A B
Xtest A B testB M=3
.ends

.subckt testB A B
R1 A B 0.5  M=4
.ends
```

Figure 5-6. Example nested subcircuits using a multiplier.

A slightly more complicated example of nested subcircuits, which also makes use of expressions, is given in figure 5-7. In this example the effective multiplier on `Xtest1:Xtest:R1` will be `M=4*0.5*2*15=60`.

```
Xtest1 2 3 testA  M=15

.subckt testA A B
.param fred=2
Xtest A B testB M='0.5*fred'
.ends

.subckt testB A B
R1 A B 0.5 M=4
.ends
```

Figure 5-7. Example nested subcircuits using a multiplier, where a multiplier is set with an expression.

As noted above, the `M` subcircuit instance multiplier is not considered to be a parameter available to expressions. So, attempting to use it in an expression will not work, resulting in a fatal error. However, it is allowed to declare a parameter named `M`, and use it in combination with multlipliers. For example of such usage, see figure 5-8.

```
Xtest1 2 3 testA  M=15

.subckt testA A B
Xtest A B testB M=2
.ends

.subckt testB A B M=3
R1 A B 0.5 M='4*M'
.ends
```

Figure 5-8. Example of `M` being used as a subcircuit multiplier and a user-defined parameter.

In this case, `M` is being declared as a user-defined parameter of value `M=3` inside the subcircuit `testB`, and this parameter is used inside the right-hand-side expression of `M='4*M'`. The expression `4*M` will evaluate to `12`. The other `M` keywords in this netlist are multipliers on the subcircuit instance lines, and they are applied in the usual way. As a result, the final multiplier on `Xtest1:Xtest:R1` is `M=12*2*15=360`.

This page intentionally left blank.

6. ANALOG BEHAVIORAL MODELING

Chapter Overview

This chapter describes analog behavioral modeling in Xyce. Sections include:

- Section 6.1,*Overview of Analog Behavioral Modeling*
- Section 6.2, *Specifying ABM Devices*
- Section 6.3, *Guidance for ABM Use*

6.1. Overview of Analog Behavioral Modeling (ABM)

The analog behavioral modeling capability of Xyce provides for flexible descriptions of electronic components or subsystems in terms of a transfer function or lookup table. In other words, a mathematical relationship is used to model a circuit segment thereby removing the need for component-by-component design information for those components or subsystems.

The `B` device, or nonlinear dependent source, is the primary device used for analog behavioral modeling in Xyce. A `B` device can serve as either a voltage or current source, and by using expressions dependent on voltages and currents elsewhere in the circuit the user can produce a wide range of behaviors.

6.2. Specifying ABM Devices

ABM devices (`B` devices) are specified in a netlist the same way as other devices. Customizing the operational behavior of the device is achieved by defining an ABM expression describing how inputs are transformed into outputs.

For example, the following pair of lines would provide exactly the same behavior as a 10K resistor between nodes 1 and 2, and is written to be a current source with current specified using Ohm's law and the constant resistance value of 10K Ω.

```
.PARAM Res1=10K
Blinearres 1 2 I={(V(2)-V(1))/Res1}
```

A nonlinear resistor could be specified similarly:

```
.PARAM R1=0.15
.PARAM R2=6
.PARAM E2 = {2*E1}
.PARAM delr = {R1-R0}
.PARAM k1 = {1/E1**2}
.PARAM r2 = {R0+sqrt(2)*delr}

.FUNC Rreg1(a,b,c,d) {a +(b-a)*c/d}
.Func Rreg2(a,b,c,d,f) {a+sqrt(2-b*(2*c-d)**2)*f}

Bnlr 4 2 V = {I(Vmon) * IF(
+ V(101) < E1, Rreg1(R0,R1,V(101),E1),
+ IF(
+ V(101) < E2, Rreg2(R0,k1,E1,V(101),delr), R2
+ )
+ )}
```

In this example, `Bnlr` provides a voltage between nodes 4 and 2, determined using Ohm's law with a resistance that is a function of the voltage on node 101 and a number of parameters. These two examples demonstrate how the `B` source can be used either as a voltage source (by specifying `V={expression}`) or as a current source (with `I={expression}`).

NOTE: Unlike expressions used in parameters or function declarations, expressions in the nonlinear dependent source may contain voltages and currents from other parts of the circuit, or even explicit time-dependent functions. Xyce evaluates these expressions when the current or voltage through the ABM source is needed. In contrast, expressions used in parameters or function declarations are evaluated only once, prior to the start of the circuit simulation.

6.2.1. Additional constructs for use in ABM expressions

ABM expressions follow the same rules as other expressions in a netlist, with the additional ability to specify signals (node voltages and voltage source currents) and explicitly time-dependent functions in the expression. In ABM expressions, refer to signals by name. Xyce recognizes the following constructs in ABM expressions:

- `V(<node name>)`
- `V(<node name>,<node name>)` (the voltage difference between the first and second nodes)
- `I(<voltage source name>)`
- Reserved simulator variables, such as `TIME`, `TEMP`, `VT`, `FREQ` and `GMIN`
- The constants, `PI` and `EXP`, which equal π and e, respectively.
- Lookup tables, which may be a function of time and other inputs
- User-defined functions via `.func`
- User-defined `.param` and `.global_param` parameters.
- Random operators such as `AGAUSS`.

In a hierarchical circuit (a circuit with possibly nested levels of subcircuits), voltage source names in an ABM expression must be the name of a voltage source in the same subcircuit as the ABM device, or in a subcircuit instantiated by that subcircuit. Similarly, node names in an ABM expression must be the node names of one or more devices in the same subcircuit as the ABM device, or in a subcircuit instantiated by that subcircuit.

6.2.2. Examples of Analog Behavioral Modeling

A variety of examples of legal usage of analog behavioral modeling is probably the most effective means of demonstrating what is allowed. The following netlist fragment shows the range of simple items allowed in ABM expressions:

```
* Current through B1 given as expression of voltage drop between
* nodes 2 and 3 plus current through voltage source Vr4mon
B1   1   0   I={V(2,3) + I(Vr4mon)}
R4   2   0   10K
Vr4mon   2a 2   0V
* Voltage across device Em given as time-dependent expression
Em   3   2a   VALUE={PAR3+1000*time}
* Voltage across device B2 set to  current through device Em
```

```
B2  2a  0  V={I(Em)}
M3  Drain  6  0  NMOD
VdrainM3 DrainPrime Drain 0v
* Voltage across B3 is function of voltage on node two and
* current through device VdrainM3
B3  6  4  V={I(VdrainM3)+V(2)}
* Voltage across device B4 is function of an internal node
* named "5" of subcircuit instance X1
X1 1 3 mysubcircuit
B4 4 5 V={V(X1:5)}
* Current through device B5 taken from current through
* internal device V4 of subcircuit instance X1
B5 4 5 I={I(X1:V4)}
```

The range of items that can be used in the current and voltage parameters of a B (or E, F, G, or H) source is far greater than what is allowed for expressions in other contexts. In particular, the use of solution values (V(*), V(*,*), I(V*)) are prohibited in all other expressions because they lead to unstable behavior if used elsewhere beside ABM. Time-dependent expressions are allowed for some device parameters, but this feature should be used with caution, as the behavior of a non-ABM device cannot be guaranteed to be correct when its device parameters are not constant throughout a simulation run. Frequency-dependent expressions are not supported for the current and voltage parameters of a B (or E, F, G, or H) source.

6.2.2.1. Behavioral modeling with tables

Lookup tables and splines provide a means of specifying a interpolated data in an expression. A table expression is specified with the keyword `TABLE` followed by an expression that is evaluated as the independent variable of the function, followed by a list of pairs of independent variable/dependent variable values. For example:

Example: B1 1 0 V={TABLE {time} = (0, 0) (1, 2) (2, 4) (3, 6)}

An equivalent example uses the table function, which has a simpler syntax, but may be hard to read for long tables:

Example: B1 1 0 V={TABLE(time, 0, 0, 1, 2, 2, 4, 3, 6)}

The previous two examples will produce a voltage source (B1) whose voltage is a simple linear function of time. At $t = 0$ the voltage is 0 volts and at time $t = 1s$ the voltage is 2 volts. Similarly, the voltage will be 4 volts at $t = 2s$ and 6 volts at $t = 3s$. Linear interpolation is used at times in between those tabulated values.

It is also possible to create ABM sources from files of time-value pairs by providing the name of the file containing the pairs between quotation marks:

Example: Bfile 1 0 V=tablefile(“myfile”)

Example: Bfile 1 0 V={tablefile(“myfile”)}

Example: Bfile 1 0 V={table(“myfile”)}

The file provided must have one time-voltage pair per line, separated by spaces. Comma-separated files are not supported, and will not be parsed correctly. If the file “myfile” contains the following data:

```
0 0
1 2
2 4
3 6
```

then the “Bfile” example above will be identical to either of the “B1” examples given above. The quoted-file sytax is in fact converted internally to precisely the same TABLE format as the first B1 example, with an independent variable of TIME and the given time-value pairs inserted.[1]

The independent variable of the table source does not have to be a simple expression. In the example shown below, the independent variable is a function of voltages and currents throughout the circuit.

Example: Bcomplicated 1 0 V={TABLE {V(5)-V(3)/4+I(V6)*Res1} = (0, 0) (1, 2) (2, 4) (3, 6)}

There is also a version of the table capability which can run significantly faster for really large tables. That capability is the `fasttable` capability, which is available in the Xyce expression library. It has the same format as `table`, and behaves very similarly. However, whereas `table` will apply time integration breakpoints at every entry in the table, the `fasttable` capability will only apply breakpoints at the first and last time point in the table.

6.2.2.2. Behavioral modeling with splines

An alternative to `table` is to use one of the more advanced interpolator functions that are available in the Xyce expression library. All of these methods use higher-order interpolation methods and (in theory) will produce a smoother result. As they tend to be very smooth, the interpolators do not produce any time integration breakpoints, except for the first and last time point. In this regard (breakpoints) they behave the same as `fasttable`. The spline options include `spline`, `cubic`, `akima` [8], and `wodicka` [9]. Another high-order interpolator is `BLI`, which is for Barycentric Lagrange Interpolation [10]. All of these interpolators use the same syntax as the `table` function in the expression library. For example:

Example: Bfile 1 0 V={spline(“myfile”)}

Example: Bfile 1 0 V={bli(“myfile”)}

Higher orders of interpolation is primarily useful for data that represents a smooth function. If the data has intentionally sharp discontinuities (such as a square wave input), then higher orders of interpolation (and the removal of breakpoints) are a bad choice. Discontinuties due to noisy measurement data may be another matter. Assuming noise isn’t a desirable aspect, using splines will naturally smooth away noise.

[1] The use of a B source in this manner is similar to using the `FILE` option to the piecewise linear (`PWL`) voltage source as documented in the Xyce Reference Guide [3], but unlike the PWL source, the file-based table function does *NOT* support reading comma-separated files.

6.2.2.3. Automatically sparsifying tabular data

Interpolated ABM models, based on tabular data can optionally have the size of their dataset reduced using the sparsification option. To enable this feature, the `TABLE` or `SPLINE` operator requires an additional argument specifying the number of points. To use a log scale for the gradient based table refinement, a third argument can be set to "true". Sparsification can only be used for tables specified from a file.

Example: Bfile 1 0 V={table("myfile",100)}

Example: Bfile 1 0 V={spline("myfile",100)}

Example: Bfile 1 0 V={table("myfile",100, true)}; use log scale for gradients

In the above examples, the number of points used internally for the `Bfile` device is automatically reduced to 100. This reduction is a gradient-based reduction, so the resulting dataset will cluster larger numbers of points in regions of high gradients and small numbers of points where the dataset is nearly flat.

The reduction requires an internal spline interprolation. The internal spline interpolation is used, no matter which interpolation option is later used by the Bsrc. So, for example, even though `{table("myfile",100)}` specifies a piecewise-linear interpolation, the internal sparsification will use a spline.

The motivation for using this feature is to improve simulation performance. Piecewise linear sources in Xyce apply time integration breakpoints at every point in the dataset. For large datasets with finely spaced time points, Xyce may take a lot of very small time steps. This can result in very slow simulation times. Sparsifying the data will mitigate this issue.

Table sparsification is only useful for data pulses that resemble smooth functions. If the data has a lot of sharp discontinuities (such as a square wave), this feature is not recommended. Also, this feature will tend to remove noise from a dataset.

6.2.3. Alternate behavioral modeling sources

In addition to the primary nonlinear dependent source, the `B` source, Xyce also supports the PSpice extensions to the standard Spice voltage- and current-controlled sources, the `E`, `F`, `G`, and `H` sources. Xyce provides these sources for PSpice compatibility, and converts them internally into equivalent `B` sources. The Xyce Reference Guide [3] netlist reference chapter provides the syntax of these compatibility devices.

6.3. Guidance for ABM Use

6.3.1. ABM devices add equations to the system of equations used by the solver

As Xyce solves a complex nonlinear set of equations at each time step, it is important to remember this system of equations is solved iteratively to obtain a converged solution. Specifying an ABM device in a Xyce netlist adds one or more equations to the nonlinear problem that Xyce must solve.

When the nonlinear solver has converged, the expression given in the ABM device will be satisfied to within a solver tolerance. However, during the course of the iterative solve, the unconverged values of nodal voltages and currents, which are often inputs and outputs of ABM devices, are not guaranteed to be solutions to the system of equations.

During this preconverged phase, solution variables are not guaranteed to have physically reasonable values. They could, for example, temporarily have the wrong sign. Only at the end of a successful nonlinear iterative solve are the solution variables consistent, legal values. This convergence behavior motivates the caveats on ABM usage given in the next subsection.

6.3.2. Expressions used in ABM devices must be valid for any possible input

While ABM devices look temptingly like calculators, it is potentially dangerous to use them as such. The previous subsection stated that during the nonlinear solution of each timestep equations, nodal voltages and currents are usually not solutions to the full set of equations, and often violate Kirchhoff's laws. Only at the end of the nonlinear solution are all the constraints on voltages and currents satisfied. This has some important consequences to the user of ABM devices.

All expressions involving nodal voltages and currents used in ABM devices should be valid for any possible value they might see — even those that appear to be physically meaningless and those that a knowledgeable user might never expect to see in the real circuit. This is particularly important when using square roots or exponentiating to a fractional power. For example, consider the following netlist fragment:

```
*...other parts of more complex circuit deleted...
* potentially bad usage of ABM device
Vexample 1 0 5V
d1 1 0 diode_model
B1 2 0 V={sqrt(v(1))}
r1 2 0 10k
*...other parts of more complex circuit deleted...
```

This example demonstrates a potentially dangerous usage. It is assumed, because node 1 is connected to a 5V DC source, that the argument of the square root function is always positive. However, it could be the case that during the nonlinear solution of the full circuit that an unconverged value of node 1 might be negative. Tracking down mistakes such as this can be difficult, as on most platforms B1 will result in a "Not a Number" value for the nodal voltage of node 2 but the program will not crash. This frequently results in inexplicable "timestep too small" errors.

Although such things can be avoided by protecting the arguments of functions with a limited domain, care must be taken when doing this. One obvious way to protect the example circuit fragment would be to take the absolute value of V(1) before calling the square root (sqrt) function:

```
*...other parts of more complex circuit deleted...
* safer usage of ABM device
Vexample 1 0 5V
d1 1 0 diode_model
B1 2 0 V={sqrt(abs(v(1)))}
r1 2 0 10k
```

```
*...other parts of more complex circuit deleted...
```

There are many other ways to protect the square root function from negative arguments, such as by using the maximum of zero and V(1). Some alternatives might be more appropriate than others in different contexts.

Note, though, that it would be a mistake to attempt to generate the absolute value as shown here:

```
*...other parts of more complex circuit deleted...
* really bad misuse of ABM device
Vexample 1 0 5V
d1 1 0 diode_model
B2 3 0 V={abs(v(1))} ; watch out!
B1 2 0 V={sqrt(v(3))}; just as bad as first example!
r1 2 0 10k
*...other parts of more complex circuit deleted...
```

There are two things wrong with this example — first, node 3 is floating and this alone could lead to convergence problems. Second, by adding the second ABM device one has merely created an equation whose solution is that node 3 contains the absolute value of the voltage on node 1. However, until convergence is reached there is no guarantee node 3 will be precisely the absolute value of V(3), nor is it guaranteed that node 3 will have a positive voltage. To re-iterate, nodes have values that are solutions to the set of equations created by the netlist only at convergence.

6.3.3. Infinite slope transitions can cause convergence problems

It is possible for a user to specify expressions that could have infinite-slope transitions with B-, E-, F-, G- and H-sources. This can lead to "timestep too small" errors when Xyce reaches the transition point. For example, inclusion of the following B-source expression in a circuit can cause the simulation to fail when `V(IN)=3.5`:

```
Bcrtl OUTA 0 V={ IF( (V(IN) > 3.5), 5, 0 ) }
```

Infinite-slope transitions in expressions dependent only on the `time` variable are a special case, because Xyce can detect that they are going to happen, and set a "breakpoint" to capture them. Infinite-slope transitions depending on other solution variables, however, cannot be predicted in advance, and will cause the time integrator to scale back the timestep repeatedly in an attempt to capture the feature until the timestep is too small to continue. One solution to this problem is to modify the expression to allow a continuous transition. For example, the above expression could be modified to:

```
Bcrtl OUTA 0 V={IF( (V(IN) > 3.4), IF( (V(IN) > 3.5), 5, 50*(V(IN)-3.4) ), 0 )}
```

However, this can become complicated with multiple inputs. The other solution is to specify device options or instance parameters to allow smooth transitions. The parameter `smoothbsrc` enables the smooth transitions. This is done by adding a RC network to the output of B sources. For example,

```
Bcrtl OUTA 0 V={ IF( (V(IN) > 3.5), 5, 0 ) } smoothbsrc=1
```

```
.options device  smoothbsrc=1
```

The smoothness of the transition can be controlled by specifying the rc constant of the RC network. For example,

```
Bcrtl OUTA 0 V={ IF( (V(IN) > 3.5), 5, 0 ) } smoothbsrc=1
+ rcconst = 1e-10

.options device  smoothbsrc=1 rcconst = 1e-10
```

Note that this smoothed ABM only applies to voltage sources, such as voltage B sources and E sources. The voltage behavioral source supports two instance parameters `smoothbsrc` and `rcconst`. Parameters may be provided as space separated `<parameter>=<value>` specifications as needed. The default value for `smoothbsrc` is 0 and the default for `rcconst` is 1e-9.

If the device options are specified, they apply to all voltage ABM devices in the circuit. If the device instance parameters are specified, they apply to that device. When both the device options and instance parameters are specified, the device instance parameters take precedence over the device options.

6.3.4. ABM devices should not be used purely for output postprocessing

Users sometimes use ABM devices to provide output postprocessing. For example, if a user was interested in the absolute value, or the log of an output voltage, then that user might create an ABM circuit element to calculate the desired output value.

Using ABM sources in this manner is bad practice though. By creating a circuit element whose only purpose is postprocessing, Xyce is forced to include it and the corresponding nonlinear solve in the circuit, which can cause unnecessary solver problems. If postprocessing is the goal, it is much better to use expressions directly on the `.PRINT` line.

An example of a “bad use” of ABM sources can be found in the following code fragment:

```
* Bad example
B1 test1 0 V = {(abs(I(VMON)))*1.0e-10}
VIN 1 0 DC 5V
R1 1 2 2K
D1 3 0 DMOD
VMON 2 3 0
.MODEL DMOD D (IS=100FA)
.DC VIN 5 5 1
.PRINT DC I(VMON) V(3) V(test1)
```

Although the source `B1` provides a postprocessing output, it doesn’t play a functional role in the circuit and Xyce would still be forced to include `B1` in the problem it is attempting to solve.

A better solution to the previous problem is given here, where the post-processing is done on a `.PRINT` line:

```
* Good example
VIN 1 0 DC 5V
R1 1 2 2K
D1 3 0 DMOD
VMON 2 3 0
.MODEL DMOD D (IS=100FA)
.DC VIN 5 5 1
.PRINT DC I(VMON) V(3)  {(abs(I(VMON)))*1.0e-10}
```

Section 9.1 and the Xyce Reference Guide [3]provide a more detailed explanation of how to use expressions in the `.PRINT` line.

6.3.5. ABM devices in frequency domain analysis

ABM devices can be used in both types of frequency domain analysis, AC and HB. However, there is one important limitation. They can only be used in this type of analysis if the source is not an independent source. In other words, they cannot be time-dependent. ABM sources that only depend on circuit solution variables (such as voltage node values) will work in both AC and HB.

7. ANALYSIS TYPES

Chapter Overview

This chapter describes the different analysis types available in Xyce. It includes the following sections:

- Section 7.1, *Introduction*
- Section 7.2, *DC Analysis*
- Section 7.3, *Transient Analysis*
- Section 7.4, *STEP Parametric Analysis*
- Section 7.5, *Sampling Analysis*
- Section 7.6, *Polynomial Chaos Expansion Analysis*
- Section 7.7, *Harmonic Balance Analysis*
- Section 7.8, *AC Analysis*
- Section 7.9, *Noise Analysis*
- Section 7.10, *Sensitivity Analysis*
- Section 7.11, *S-parameter Analysis*

7.1. Introduction

Xyce supports several simulation analysis options, including DC bias point (`.DC`, section 7.2), transient (`.TRAN`, section 7.3), AC (`.AC`, section 7.8), Noise (`.NOISE`, section 7.9), harmonic balance (`.HB`, section 7.7), and sensitivity (`.SENS`, section 7.10) analysis.

Using `.STEP`(section 7.4), Xyce can also apply an outer parametric loop to any type of analysis. This allows one (for example) to sweep a model parameter and perform a transient simulation for each parameter value.

Using `.SAMPLING` or `.EMBEDDEDSAMPLING` (section 7.5), Xyce can apply random sampling loop to any type of analysis. This will compute statistical moments for various circuit outputs.

Alternatively, one can use several types of Polynomial Chaos Expansion (PCE) methods in Xyce to propagate uncertainty thru a calculation (section 7.6). Similar to the sampling methods, these methods can also be used to compute statistical moments for various circuit outputs.

There are some analysis types typically found in SPICE-style simulators that are still a work in progress for Xyce. Operating point analysis (`.OP`, section 7.2.3) is partially supported in Xyce.

7.2. Steady-State (.DC) Analysis

The DC sweep analysis capability in Xyce computes the DC bias point of a circuit for a range of values of input sources. DC sweep is supported for a source or device parameter, through a range of specified values. As the sweep proceeds, Xyce computes the bias point for each value in the specified range of the sweep.

If the variable to be swept is a voltage or current source, a DC source must be used and its value set in the netlist (see Xyce Reference Guide [3]). In simulating the DC response of an analog circuit, Xyce eliminates time dependence from the circuit by treating capacitor elements as open circuits and inductor elements as short circuits, while using only the DC values of voltage and current sources.

7.2.1. .DC Statement

To specify a `.DC` analysis, include a `.DC` line in the netlist. Some examples of typical `.DC` lines are:

Example:

```
.DC V1 7m 5m -1m
.DC I1 5u 10u 1u
.DC M1:L 7u 5u -1u
.DC OCT V0 0.125 64 2
.DC DEC R1 100 10000 3
.DC TEMP LIST 10.0 15.0 18.0 27.0 33.0
.DC data=table
.param init=7m, final=5m, step=-1m
.DC V1 {init} {final} {step}
```

The examples include several types of sweep — linear, octave, decade, list and data. They also demonstrate sweeping over voltage and current sources as well as device parameters. The Xyce Reference Guide [3] provides a complete description of each.

7.2.2. Setting Up and Running a DC Sweep

Following the example given in section 3.2, figure 7-1 shows the diode clipper circuit netlist with a DC sweep analysis specified. Here, the voltage source `Vin` is swept from -10 to 15 in 1-volt increments, resulting in 26 DC operating point calculations.

NOTE: Xyce ignores the default setting for `Vin` during these calculations. All other source values use the specified values (in this case, `VCC = 5V`).

Running Xyce on this netlist produces an output results file named `clipper.cir.prn`. Obtaining this file requires specifying the `.PRINT DC` line. Plotting this data produces the graph shown in figure 7-2.

```
Diode Clipper Circuit
** Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940 D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
.MODEL D1N3940 D(
+ IS=4E-10 RS=.105 N=1.48 TT=8E-7
+ CJO=1.95E-11 VJ=.4 M=.38 EG=1.36
+ XTI=-8 KF=0 AF=1 FC=.9
+ BV=600 IBV=1E-4)
.END
```

Figure 7-1. Diode clipper circuit netlist for DC sweep analysis.

7.2.3. OP Analysis

Xyce also supports `.OP` analysis statements. In Xyce, consider `.OP` as a shorthand for a single-step DC sweep, in which all the default operating point values are used. One may also consider `.OP` analysis to be

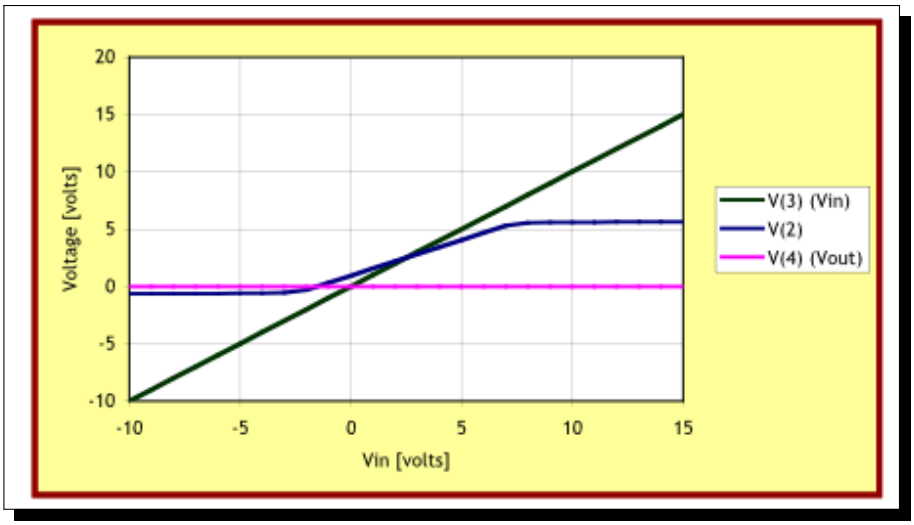

Figure 7-2. DC sweep voltages at Vin, node 2 and Vout.

the operating point calculation that would occur as the initial step to a transient calculation, without the subsequent time steps.

This capability was mainly added to enable the code to handle legacy netlists using this analysis statement type. In most versions of SPICE, using `.OP` results in extra output not available from a DC sweep. Xyce will also output some of this extra information about devices, but the capability is not fully implemented.

7.2.4. Output

During analysis a number of output files may be generated. The selection of which files are created depends on a variety of factors, most obvious of which is the `.PRINT` command. Table 7-1 lists the format options and files created. The column labeled “Additional Columns” lists the additional data that is written, though not specified on the `.PRINT` line.

Table 7-1. Output generated for DC analysis

Command	Files	Additional Columns
`.PRINT DC`	*circuit-file*.prn	INDEX TIME
`.PRINT DC FORMAT=NOINDEX`	*circuit-file*.prn	TIME
`.PRINT DC FORMAT=CSV`	*circuit-file*.csv	TIME
`.PRINT DC FORMAT=RAW`	*circuit-file*.raw	TIME
`.PRINT DC FORMAT=TECPLOT`	*circuit-file*.dat	TIME
`.PRINT DC FORMAT=PROBE`	*circuit-file*.csd	TIME
Xyce `-r raw-file-name`	*raw-file-name*	All circuit variables printed
Xyce `-r raw-file-name -a`	*raw-file-name*	All circuit variables printed
`.OP`	*log-file*	Operating point information

7.3. Transient Analysis

The transient response analysis simulates the response of the circuit from `TIME=0` to a specified time. Throughout a transient analysis, any or all of the independent sources may have time-dependent values.

In Xyce (and most other circuit simulators), the transient analysis begins by performing its own bias point calculation at the beginning of the run, using the same method as used for DC sweep. This is required to set the initial conditions for the transient solution as the initial values of the sources may differ from their DC values.

7.3.1. .TRAN Statement

To run a transient simulation, the circuit netlist file must contain a `.TRAN` command.

Example:

```
.TRAN 100us 300ms
.TRAN 100p 12.05u 9.95u
```

Example:

```
.param tstep=100us, final=300ms
.TRAN {tstep} {final}
```

The Xyce Reference Guide [3] provides a detailed explanation of the `.TRAN` statement. The netlist must also contain one of the following:

- Independent, transient source (see table 7-2),
- Initial condition on a reactive element, or
- Time-dependent analog behavioral modeling source (see chapter 6)

7.3.2. Defining a Time-Dependent (transient) Source

7.3.2.1. Overview of Source Elements

Source elements, either voltage or current, are entered in the netlist file as described in the Xyce Reference Guide [3]. Table 7-2 lists the time-dependent sources available in Xyce for either voltage or current. For voltage sources, the name is preceded by `V` while current sources are preceded by `I`.

To use time-dependent or transient sources, place the source element line in the netlist and characterize the transient behavior using the appropriate parameters. Each transient source element has a separate set of parameters dependent on its transient behavior. In this way, the user can create analog sources that produce sine wave, square pulse, exponential pulse, single-frequency FM, and piecewise linear (PWL) waveforms.

Table 7-2. Summary of Xyce-supported time-dependent sources

Source Element Name	Description
EXP	Exponential Waveform
PAT	Pattern Waveform
PULSE	Pulse Waveform
PWL	Piecewise Linear Waveform
SFFM	Frequency-modulated Waveform
SIN	Sinusoidal Waveform

7.3.2.2. Defining Transient Sources

To define a transient source, select one of the supported sources: independent voltage or current, choose a transient source type from table 7-2, and provide the transient parameters (refer to the Xyce Reference Guide [3] to fully define the source).

The following example of an independent sinusoidal voltage source in a circuit netlist creates a voltage source between nodes 1 and 5 that oscillates sinusoidally between -5V and +5V with a frequency of 50 KHz. The arguments specify an offset of -5V, a 10V amplitude, and a 50KHz frequency, in that order.

Example: `Vexample 1 5 SIN(-5V 10V 50K)`

7.3.3. Transient Time Steps

During the simulation, Xyce uses a calculated time step that is continuously adjusted for accuracy and efficiency (see [11] and [12]). Calculation time step increases during periods of circuit idleness, and decreases during dynamic portions of the waveform. Users may control the maximum internal step size by specifying the step's ceiling value in the `.TRAN` command (see the Xyce Reference Guide [3]).

The internal calculation time steps used might not be consistent with the user-requested output time steps. By default, Xyce outputs solution results at every time step it calculates. If the user selects output timesteps via the `.OPTIONS OUTPUT` statement (see chapter 9), then Xyce will output results at the interval requested, interpolating solution variables to desired output times if necessary.

7.3.4. Time Integration Methods

For a transient analysis, several time integration methods can be selected to solve the circuit model's differential algebraic equations. The following algorithms are available:

- Variable order Trapezoidal (combines Trapezoidal and Backward Euler)
- Gear method, orders 1-2.

You can set the `method`, `maxord` and `minord` parameters to select the time integration methods via a .OPTIONS line. The following table shows the possible settings for those three parameters. (Note: Consult the Xyce Reference Guide [3]for the exact syntax of the .OPTIONS line for each time integration method.) The default time integration method in Xyce is Trap, which is the same as SPICE, PSpice and HSPICE.

Table 7-3. Summary of Xyce-supported time integration methods

Integration Methods	Option Settings	Comments
Backward-Euler	method=trap maxord=1	Backward-Euler only
Trap	method=trap	combines Trapezoidal and Backward Euler (default)
Trap only	method=trap minord=2	Trapezoidal only
Gear	method=gear	combines Backward Euler and 2nd order Gear
Gear2 only	method=gear minord=2	2nd order Gear only

The Trapezoidal method is often the preferred method because it is accurate and fast. However, this method can exhibit artificial point-to-point ringing, which can be controlled by using tighter tolerances. If a circuit fails to converge with the Trapezoidal method then you can re-run the transient analysis using the Gear method.

The Gear method may help convergence for some circuits. The 2nd order Gear method is typically more accurate than the Backward-Euler method (1st order Gear). However, both of these methods are overly stable methods, and they can damp the actual circuit behavior when simulating high-Q resonators such as oscillators. The Backward-Euler method has more damping effect than the 2nd order Gear method. This effect can be alleviated by using tighter tolerances in the simulations. However, it is suggested to use the pure Trapezoidal method for oscillators.

7.3.5. Error Controls

There are two basic time-step error control methods in Xyce — Local Truncation Error (LTE) based and non-LTE based.

7.3.5.1. Local Truncation Error (LTE) Strategy

All time integration methods use the LTE-based strategy by default. The accuracy of the simulation can be controlled by specifying appropriate relative and absolute error tolerances (RELTOL and ABSTOL).

Example:

```
.OPTIONS TIMEINT RELTOL=1e-4 ABSTOL=1e-8
```

The total tolerance of LTE is

Tol_{LTE} = `abstol + reltol*ref`

The parameter `ref` is the reference value that the relative error is compared to. It can be controlled by setting `newlte` option.

Example:

```
.OPTIONS TIMEINT NEWLTE=1
```

The choices for `newlte` option are:

- 0. The reference value is the current value on each node. This is the most conservative and least used.
- 1. The reference value is the maximum of all the signals at the current time. This is the default value.
- 2. The reference value is the maximum of all the signals over all past time. This is the loosest criterion. It normally produces the best performance and should be used if the overall size of the signals is roughly the same on all nodes.
- 3. The reference value is the maximum value on each signal over all past time. This should be used if the scale of signals varies widely in a system.

The Trapezoid integrator algorithm introduces no numerical dissipation. So, a strong ringing (artificially introduced by the numerical algorithm) will occur when sources or models introduce discontinuities. This can result in a large local truncation error estimate, ultimately leading to a "time-step too small" error. In this case, using the Gear method or a non-LTE strategy may help.

7.3.5.2. Non-LTE Strategy

The non-LTE strategy used in Xyce is based on success of the nonlinear solve, and is enabled by setting ERROPTION=1. Since the step-size selection is based only upon nonlinear iteration statistics rather than accuracy, it is highly suggested that DELMAX be specified, in a circuit-specific manner, for all three time integrators. The purpose of DELMAX is to limit the largest time step taken.

Example:

```
.OPTIONS TIMEINT ERROPTION=1 DELMAX=1.0e-4
```

For the Trapezoid and Gear integrators, the options are slightly more refined. If the number of nonlinear iterations is below NLMIN, then the step size is doubled. If the number of nonlinear iterations is above NLMAX then the step size is cut by one eighth. In between, the step size is not changed. An example using Trap (METHOD=7) is given below.

Example:

```
.OPTIONS TIMEINT METHOD=7 ERROPTION=1 NLMIN=3 NLMAX=8 DELMAX=1.0e-4
```

If the number of Newton iterations is bigger than NLmax and TIMESTEPSREVERSAL is not set, then Xyce will cut the next step. If the number of Newton iterations is bigger than NLmax and TIMESTEPSREVERSAL is set, then Xyce will reject the current step and also cut the current step.

Example:

```
.OPTIONS TIMEINT METHOD=7 ERROPTION=1 DELMAX=1.0e-4
TIMESTEPSREVERSAL=1
```

7.3.6. Breakpoints

It is often necessary or desirable for the time integrator to be forced to land on a specific time point and restart integration from that point. The most common scenario for which this is necessary is when an device (such as a PULSE or a PWL source) produces a discontinuity. If a discontinuity is present, and no breakpoint is set, then the LTE analysis will force the time integrator to take really small time steps, and this can possibly impact the robustness of the calculation.

Fortunately, Xyce handles most device-based discontinuities automatically, so it is not necessary for the user to worry about them. However, it can be desirable for the user to manually set breakpoints from the netlist. In a Xyce netlist, these are specified using the BREAKPOINTS parameter with a comma-separated list in the following .OPTIONS TIMEINT statement.

Example:

```
.OPTIONS TIMEINT BREAKPOINTS=1ms,2ms,3ms
```

7.3.7. Checkpointing and Restarting

Xyce was designed to simulate large, complex circuits over long simulation runs. Because complex simulations can take many hours (or even days) to complete, it can sometimes be helpful to use "checkpointing." When checkpointing is used, Xyce periodically saves its complete simulation state. The saved state can be used to restart Xyce from one of these "checkpoints." In the event of a computer crash, power outage, or should the simulation need to be interrupted for some other reason, checkpointing allows the user to restart a long simulation in the middle of a run without having to start over.

Xyce uses the .OPTIONS RESTART netlist command to control all checkpoint output and restarting.

7.3.7.1. Checkpointing Command Format

- .OPTIONS RESTART [PACK=<0|1>] JOB=<job name> INITIAL_INTERVAL=<interval> [[<t0> <i0> [<t1> <i1>...]]]

  PACK=<0|1> indicates whether restart data files will contain byte-packed (binary) data(PACK=1, the default) or unpacked (ASCII)(PACK=0). JOB=<job name> identifies the prefix for restart files. The actual restart files will be the job name appended with the current simulation time (e.g., name1e-05 for JOB=name and simulation time 1e-05 seconds). Furthermore, the INITIAL_INTERVAL=<interval> identifies the initial interval time used for restart output; this parameter must be given. The <tx ix> intervals identify times (tx) at which the output interval (ix) will change. This functionality is identical to that described for the .OPTIONS OUTPUT command (section 9.1).

- Example — Generate checkpoints every 0.1 μs:

  ```
  .OPTIONS RESTART JOB=checkpt INITIAL_INTERVAL=0.1us
  ```

- Example — Generate unpacked checkpoints every 0.1 μs:

  ```
  .OPTIONS RESTART PACK=0 JOB=checkpt INITIAL_INTERVAL=0.1us
  ```

- Example — Initial interval of 0.1 μs, at 1 μs in the simulation, change to interval of 0.5 μs, and at 10 μs change to an interval of 0.1 μs:

```
.OPTIONS RESTART JOB=checkpt INITIAL_INTERVAL=0.1us 1us 0.5us
+ 10us 0.1us
```

7.3.7.2. Restarting Command Format

-
```
.OPTIONS RESTART <FILE=<filename> | JOB=<job name> START_TIME=<time>>
+ [INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1> ...]]]
```

To restart from an existing restart file, specify the file by using either the `FILE=<filename>` parameter to explicitly request a file or `JOB=<job name> START_TIME=<time>` to specify a file prefix and a specific time. The time must exactly match an output file time for the simulator to correctly load the file.

To continue checkpointing the simulation in a restarted run, append `INITIAL_INTERVAL=<interval>` and the following intervals to the command in the same format as previously described. Without these additional parameters, the simulation will restart as requested, but will not generate further checkpoint files.

- Example — Restart from checkpoint file at 0.133 μs:

```
.OPTIONS RESTART JOB=checkpt START_TIME=0.133us
```

- Example — Restart from checkpoint file at 0.133 μs :

```
.OPTIONS RESTART FILE=checkpt0.000000133
```

- Example — Restart from 0.133 μs and continue checkpointing at 0.1 μs intervals:

```
.OPTIONS RESTART FILE=checkpt0.000000133 JOB=checkpt_again
+ INITIAL_INTERVAL=0.1us
```

7.3.8. Output

During analysis a number of output files may be generated. The selection of which files are created depends on a variety of factors, most obvious of which is the `.PRINT` command. Table 7-4 lists the format options and files created. The column labeled “Additional Columns” lists the additional data that is written, though not specified on the `.PRINT` line.

Table 7-4. Output generated for Transient analysis

Command	Files	Additional Columns
`.PRINT TRAN`	*circuit-file*.prn	INDEX TIME
`.PRINT TRAN FORMAT=NOINDEX`	*circuit-file*.prn	TIME
`.PRINT TRAN FORMAT=CSV`	*circuit-file*.csv	TIME
`.PRINT TRAN FORMAT=RAW`	*circuit-file*.raw	TIME
`.PRINT TRAN FORMAT=TECPLOT`	*circuit-file*.dat	TIME
`.PRINT TRAN FORMAT=PROBE`	*circuit-file*.csd	TIME
Xyce `-r raw-file-name`	*raw-file-name*	All circuit variables printed
Xyce `-r raw-file-name -a`	*raw-file-name*	All circuit variables printed
`.OP`	*log-file*	Operating point information

7.4. STEP Parametric Analysis

The .STEP command performs a parametric sweep for all the analyses of the circuit. When the .STEP command is invoked, typical analyses, such as .DC, .AC, and .TRAN are performed for each value of the stepped parameter.

This capability is very similar, but not identical, to the STEP capability in PSpice [4]. Xyce can use .STEP to sweep over any device instance or device model parameter, as well as the circuit temperature. It can also be used to sweep over any user-defined parameter (either .GLOBAL_PARAM or .PARAM) , as long as that parameter is defined in the global scope.

7.4.1. .STEP Statement

A .STEP analysis may be specified by simply adding a .STEP line to a netlist. Unlike .DC, .STEP by itself is not an adequate analysis specification, as it merely specifies an outer loop around the normal analysis. A standard analysis line, either specifying .TRAN, .AC and .DC analysis, is still required.

Some examples of typical .STEP lines are:

Example:

```
.STEP M1:L 7u 5u -1u
.STEP OCT V0 0.125 64 2
.STEP DEC R1 100 10000 3
.STEP TEMP LIST 10.0 15.0 18.0 27.0 33.0
.STEP data=table
```

.STEP has a format similar to that of the .DC format specification. In the first example, M1:L is the name of the parameter (in this instance, the length parameter of the MOSFET M1), 7u is the initial value of the parameter, 5u is the final value of the parameter, and -1u is the step size. Like .DC, .STEP in Xyce can also handle sweeps by decade, octave, a specified lists of values, or multivariate parameter values from a table. Consult the Xyce Reference Guide [3] for complete explanations of each sweep type.

7.4.2. Sweeping over a Device Instance Parameter

The first example uses M1:L as the parameter, but it could have used any model or instance parameter existing in the circuit. Internally, Xyce handles the parameters for all device models and device instances in the same way. Users can uniquely identify any parameter by specifying the device instance name, followed by a colon (:), followed by the specific parameter name. For example, all the MOSFET models have an instance parameter for the channel length, L. For a MOSFET instance specified in a netlist, named M1, the full name for the M1 channel length parameter is M1:L.

Figure 7-3 provides a simple application of .STEP to a device instance. This is the same diode clipper circuit as was used in the transient analysis chapter, except that a single line (in red) has been added. This .STEP line will cause Xyce to sweep the resistance of the resistor, R4, from 3.0 KOhms to 15.0 KOhms, in 2.0 KOhms increments, meaning seven transient simulations will be performed, each one with a different value for R4.

As the circuit is executed multiple times, the resulting output file is a little more sophisticated. The .PRINT statement is still used in much the same way as before. However, the .prn output file contains the concatenated output of each .STEP increment. The end of this section provides more details of how .STEP changes output files.

```
Transient Diode Clipper Circuit with Step Analysis
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
 * Step statement
.STEP R4:R 3.0K 15.0K 2.0K
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
.MODEL D1N3940 D(
+ IS=4E-10 RS=.105 N=1.48 TT=8E-7
+ CJO=1.95E-11 VJ=.4 M=.38 EG=1.36
+ XTI=-8 KF=0 AF=1 FC=.9
+ BV=600 IBV=1E-4)
.END
```

Figure 7-3. Diode clipper circuit netlist for step transient analysis

7.4.3. Sweeping over a Device Model Parameter

Sweeping a model parameter can be done in an identical manner to an instance parameter. Figure 7-4 contains the same circuit as in figure 7-3, but with additional .STEP line referring to a model parameter, D1N3940:IS.

NOTE: .STEP line syntax differs from .DC line syntax in that multiple parameters require separate .STEP lines. Each parameter needs a separate line.

```
Transient Diode Clipper Circuit with Step Analysis
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
 * Step statements
.STEP R4:R 3.0K 15.0K 2.0K
.STEP D1N3940:IS 2.0e-10 6.0e-10 2.0e-10
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
.MODEL D1N3940 D(
+ IS=4E-10 RS=.105 N=1.48 TT=8E-7
+ CJO=1.95E-11 VJ=.4 M=.38 EG=1.36
+ XTI=-8 KF=0 AF=1 FC=.9
+ BV=600 IBV=1E-4)
.END
```

Figure 7-4. Diode clipper circuit netlist for 2-step transient analysis

7.4.4. Sweeping over Temperature

It is also possible to sweep over temperature. To do so, simply specify `temp` as the parameter name. It will work in the same manner as `.STEP` when applied to model and instance parameters.

7.4.5. Special cases: Sweeping Independent Sources, Resistors, Capacitors and Inductors

For some devices, there is generally only one parameter that one would want to sweep. For example, a linear resistor's only parameter of interest is resistance, R. Similarly, for a DC voltage or current source, one is usually only interested in the magnitude of the source, DCV0. Finally, linear capacitors generally only have capacitance, C, as a parameter of interest, while inductors generally only have inductance, L, as a parameter of interest.

For these simple devices, it is not necessary to specify both the parameter and device on the `.STEP` line: only the device name is strictly required, as these devices have default parameters that are assumed if no parameter name is given explicitly.

Examples of usage are given below. The first two lines are equivalent — in the first line, the resistance parameter of `R4` is named explicitly, and in the second line the resistance parameter is implicit. A similar example is then shown for the DC voltage of the voltage source `VCC`. In the remaining lines, parameter names are all implicit, and the default parameters of the associated devices are used.

Example:

```
.STEP R4:R      3.0K   15.0K   2.0K
.STEP R4        3.0K   15.0K   2.0K
.STEP VCC:DCV0  4.0     6.0    1.0
.STEP VCC       4.0     6.0    1.0
.STEP ICC       4.0     6.0    1.0
.STEP C1        0.45u   0.50u  0.1u
.STEP L1        0.5m    1.0m   0.5m
```

Independent sources require further explanation. Their default parameter, DCV0, only applies to `.DC` analyses. They do not have default parameters for their transient forms, such as `SIN` or `PULSE`.

7.4.6. Output files

Users can think of `.STEP` simulations as several distinct executions of the same circuit netlist. The output data, as specified by a `.PRINT` line, however, goes to a single (`*.prn`) file. For convenience, Xyce also creates a second auxilliary file with the `*.res` suffix.

Figure 7-3 shows an example file named `clip.cir`, which when run will produce files `clip.cir.res` and `clip.cir.prn`. The file `clip.cir.res` contains one line for each step, showing what parameter value was used on that step. `clip.cir.prn` is the familiar output format, but the `INDEX` field recycles to zero each time a new step begins. As the default behavior distinguishes each step's output only by recycling the `INDEX` field to zero, it can be beneficial to add the sweep parameters to the `.PRINT` line. For the default file format (`format=std`), Xyce will not automatically include these sweep parameters, so for plotting it is usually best to specify them by hand.

If using the default `.prn` file format (`format=std`), the resulting `.STEP` simulation output file will be a simple concatenation of each step's underlying analysis output. If using `format=probe`, the data for each execution of the circuit will be in distinct sections of the file, and it should be easy to plot the results using PROBE. If using `format=tecplot`, the results of each `.STEP` simulation will be in a distinct Tecplot zone. Finally, format=raw will place the results for each `.STEP` simulation in a distinct "plot" region.

7.5. Random Sampling Analysis

Xyce supports two different styles of random sampling analysis. One is specified with the .SAMPLING command, and the other is specified with the .EMBEDDEDSAMPLING commmand. These are relatively new capabilities and are still under development.

The .SAMPLING command causes Xyce to execute the primary analysis (.DC, .AC, .TRAN, etc.) inside a loop over randomly distributed parameters. The user specifies a list of random parameters, their distributions, distribution parameters (such as mean and standard deviation), and the total number of sample points. The primary analysis is then executed sequentially for each sample point. The .SAMPLING capability uses a lot of the same code base as .STEP, so most of the guidance for .STEP, including legal parameters, output file formats, supported analyses types, etc., also applies to .SAMPLING.

The .EMBEDDEDSAMPLING command also causes Xyce to create a loop over randomly distributed parameters, but unlike .SAMPLING, this loop is implemented inside the solver loop rather than outside of it. As such, it creates a block linear system, in which each matrix block has the structure of the original matrix, and the number of blocks is equal to the number of sample points. As such, all the parameters are propagated simultaneously, rather than sequentially. Unlike .SAMPLING, which can be applied to all types of Xyce analysis, .EMBEDDEDSAMPLING can only be applied to .DC and .TRAN.

Both types of sampling methods can be applied to a variety of inputs and outputs. For inputs, Xyce can use .SAMPLING or .EMBEDDEDSAMPLING to modify any device instance or device model parameter, as well as the circuit temperature. It can also be used to sweep over any user-defined parameter (either .GLOBAL_PARAM or .PARAM), as long as that parameter is defined in the global scope.

For outputs, both .SAMPLING and .EMBEDDEDSAMPLING can apply statistical analysis (computing moments such as mean and standard deviation) to most outputs that appear on the .PRINT line. However, the two analysis types have some differences. .SAMPLING will only apply statistical analysis at the end of the calculation, while .EMBEDDEDSAMPLING can apply statistical analysis at each time step. Relatedly, .SAMPLING can also be applied to any .MEASURE, but .EMBEDDEDSAMPLING lacks this capability.

7.5.1. .SAMPLING and .EMBEDDEDSAMPLING Statements

Sampling analysis may be specified by simply adding a .SAMPLING or a .EMBEDDEDSAMPLING line to a netlist. The format for the two commands is nearly identical.

Sampling analysis such as .SAMPLING or .EMBEDDEDSAMPLING is not considered a “primary” analysis type. This means that such a command by itself is not an adequate analysis specification, as it merely specifies an additional parametric loop to augment the primary analysis. A standard analysis line, specifying .TRAN, .AC, .HB, and .DC analysis, is still required.

Examples of typical .SAMPLING lines:

Example:

```
.SAMPLING
+ param=M1:L, M1:W
+ type=uniform,uniform
+ lower_bounds=5u,5u
+ upper_bounds=7u,7u
```

```
.SAMPLING
+ param=R1,c1
+ type=normal,normal
+ means=3K,1uF
+ std_deviations=1K,0.2uF
```

In the first example, `M1:L` and `M1:W` are the names of the parameters (in this instance, the length and width parameters of the MOSFET `M1`), `5u` is the minimum value of both parameters and `7u` is the maximum value of both parameters. For both parameters, the distribution (specified by `type`) is uniform. In the second example, the two parameters are `R1` and `c1`. In this case, it isn't necessary to specify the specific parameter of `R1` or `c1` as resistors and capacitors have simple default parameters (see section 7.5.3.5). The distributions of both parameters are normal, and the means and standard deviations are specified for both in comma-separated lists.

For any list of parameters, all the required fields must be provided in comma-separated lists. Xyce will throw an error if the length of any of the lists is different than that of the parameter list.

7.5.2. .options SAMPLES and EMBEDDEDSAMPLES Statements

To specify the number of samples, use the netlist command `.options SAMPLES numsamples=<value>`. For sampling analysis to work correctly, this parameter is required. Other sampling details, including the sampling type, the covariance matrix, the statistical outputs and the random seed can be specified using the same `.options SAMPLES` netlist command. Some examples of this command are:

Example:

```
.options samples numsamples=1000

.options samples numsamples=30
+ covmatrix=1e6,1.0e-3,1.0e-3,4e-14
+ OUTPUTS=R1:R,C1:C
+ MEASURES=maxSine
+ SAMPLE_TYPE=LHS
+ SEED=743190581
```

The random seed can be set either in the `.options SAMPLES` statement, or it can be set from the command line using the `-randseed` option. If it is not set either way, then the random seed is generated internally.

In the example there are two different types of outputs specified. One is using the keyword `OUTPUTS`, and this is used to specify a comma-separated list of outputs corresponding to typical `.PRINT` line outputs. Solution variables, user defined parameters, and expressions depending on both are typical.

The other type of output is specified using the keyword `MEASURES`, and this is used to specify a comma-separated list of netlist `.MEASURE` commands. This type of output is only available to `.SAMPLING` and not to `.EMBEDDEDSAMPLING`.

The example sets the SAMPLE_TYPE=LHS, for Latin Hypercube Sampling [13]. The other options is SAMPLE_TYPE=MC for Monte Carlo [14]. LHS is the default option for this parameter, and will generally be a better choice.

7.5.3. Specifying Uncertain Inputs

7.5.3.1. Sampling a Device Instance Parameter

Figure 7-5 provides a simple application of .SAMPLING to a device instance parameter. The circuit is a simple voltage divider driven by a oscillating voltage source. The randomly sampled parameter is the resistance of R1, which is given a normal distribution with a specified mean and standard deviation.

As the circuit is executed multiple times, an output file is generated based on the .PRINT command. The .PRINT statement is still used in the same way that it would be in a non-sampling analysis. However, the output file contains the concatenated output of each .SAMPLING step. Section 7.5.4 provides more details of how .SAMPLING changes .PRINT output files.

Xyce computes statistical outputs for quantities specified by OUTPUTS and MEASURES on the .options SAMPLES line. For these outputs, moments such as mean and variance are computed and sent to standard output. More details of this type of output, as well as examples, are given in section 7.5.5.

```
Voltage Divider with sampling
R2  1  0  7k
R1 1 2 3k
VS1  2  0  SIN(0 1.0e3 1KHZ 0 0)

.tran 0.1ms 1ms
.meas tran maxSine max V(1)

.SAMPLING param=R1:R
+ type=normal means=1K std_deviations=0.1K

.options SAMPLES numsamples=1000 OUTPUTS={R1:R}
+ MEASURES=maxSine SAMPLE_TYPE=MC stdoutput=true
```

Figure 7-5. Voltage divider circuit netlist with sampling of a device instance parameter. Sampling commands are in red.

7.5.3.2. Sampling a Device Model Parameter

Sampling a model parameter can be done in an identical manner to an instance parameter. Figure 7-6 provides a simple application of .SAMPLING to a device model and a device instance. This is the same diode clipper circuit as was used in the Transient Analysis section (see section 3.3), except that several lines of sampling commands (in red) have been added.

The sampling commands will cause Xyce to sample the model parameter, D1N3940:IS as well as the resistance of the resistor, R4. Both of the parameters are randomly sampled using uniform distributions with specified lower and upper bounds.

```
Transient Diode Clipper with Sampling Analysis

* Primary Analysis Command
.tran 2ns 0.5ms
* Output
.print tran format=tecplot V(3) V(2) V(4)
.meas tran maxSine max V(4)

* Sampling statements
.SAMPLING
+ param=R4:R,D1N3940:IS
+ type=uniform,uniform
+ lower_bounds=3.0k,2.0e-10
+ upper_bounds=15.0k,6.0e-10

.options SAMPLES numsamples=10 SAMPLE_TYPE=LHS
+ OUTPUTS=R4:R,D1N3940:IS MEASURES=maxSine stdoutput=true
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
.MODEL D1N3940 D(
+ IS=4E-10 RS=.105 N=1.48 TT=8E-7
+ CJO=1.95E-11 VJ=.4 M=.38 EG=1.36
+ XTI=-8 KF=0 AF=1 FC=.9 BV=600 IBV=1E-4)
.END
```

Figure 7-6. Diode clipper circuit netlist with sampling analysis. This example uses a model parameter as one of the sampling parameters. Sampling commands are in red.

7.5.3.3. Sampling a User-Defined Parameter

Sampling can be applied to user-defined parameters, as long as they are declared in the top level netlist (global scope). It cannot be applied to user-defined parameters which are defined in a subcircuit. An example of such usage, with a global parameter, is given in figure 7-7. This circuit identical to the circuit in figure 7-5, except that it uses global parameters. If the global parameters in this netlist are changed to normal parameters (`.PARAM`), this circuit will also work.

```
Voltage Divider, sampling global params
.global_param Vmax=1000.0
.global_param testNorm=1.5k
.global_param R1value={testNorm*2.0}

R2  1  0  7k
R1 1 2 {R1value}
VS1  2  0  SIN(0 {Vmax} 1KHZ 0 0)
.tran 0.1ms 1ms
.meas tran maxSine max V(1)

.SAMPLING param=testNorm
+ type=normal means=0.5K std_deviations=0.05K

.options SAMPLES numsamples=1000 SAMPLE_TYPE=MC
+ OUTPUTS={R1:R} MEASURES=maxSine
```

Figure 7-7. Voltage divider circuit netlist with sampling of a global parameter. Sampling commands are in red. This circuit will also work if the `.global_param` statements are changed to `.param`.

7.5.3.4. Sampling over Temperature

It is also possible to sample temperature. To do so, simply specify `temp` as the parameter name. It will work in the same manner as `.SAMPLING` when applied to model and instance parameters.

7.5.3.5. Special cases: Sampling Independent Sources, Resistors, Capacitors and Inductors

For some devices, there is generally only one parameter that one would want to sample. For example, a linear resistor's only parameter of interest is resistance, R. Similarly, for a DC voltage or current source, one is usually only interested in the magnitude of the source, DCV0. Finally, linear capacitors generally only have capacitance, C, as a parameter of interest, while inductors generally only have inductance, L, as a parameter of interest.

For these simple devices, it is not necessary to specify both the parameter and device on the `.SAMPLING` line: only the device name is strictly required, as these devices have default parameters that are assumed if no parameter name is given explicitly.

Examples of usage are given below. The first two lines are equivalent — in the first line, the resistance parameter of `R4` is named explicitly, and in the second line the resistance parameter is implicit. A similar example is then shown for the DC voltage of the voltage source `VCC`. In the remaining lines, parameter names are all implicit, and the default parameters of the associated devices are used.

Example:

```
.SAMPLING param=R4:R,C1:C  type=normal,normal  means=3k,1u  std_deviations=1k,0.1u
.SAMPLING param=R4,C1      type=normal,normal  means=3k,1u  std_deviations=1k,0.1u
.SAMPLING VCC:DCV0         type=uniform        means=6.0    std_deviations=1.0
.SAMPLING VCC              type=uniform        means=6.0    std_deviations=1.0
.SAMPLING param=C1         type=normal         means=0.5u   std_deviations=0.05u
.SAMPLING param=L1         type=normal         means=0.5m   std_deviations=0.01m
```

Independent sources require further explanation. Their default parameter, DCV0, only applies to `.DC` analyses. They do not have default parameters for their transient forms, such as `SIN` or `PULSE`.

7.5.3.6. Sampling using random operators from expressions

The Xyce expression library supports random operators in expressions, including `AGAUSS`, `GAUSS`, `AUNIF`, `UNIF` and `RAND`. Xyce `.SAMPLING` and `.EMBEDDEDSAMPLING` analyses are able to use these, but they are not connected by default. In order to use random expression operators with `.SAMPLING`, the following commend must be present in the netlist:

Example:

```
.SAMPLING useExpr=true
```

This capability is potentially very powerful, as the random operators can act as operators within complex expressions. As expressions can be applied to `.param`, `.global_param`, device instance parameters and device model parameters, this capability naturally can be applied to any of these types of input parameters. A simple example of this feature, applied to several instance parameters, can be seen in figure 7-8. Other examples of this capability can be found in the netlists given in figures 7-11, 7-13 and 7-18.

This feature cannot be used in the same netlist as the specification described in section 7.5.1. If `useExpr=true`, Xyce will ignore the comma-separated lists such as `param`. Likewise, if `useExpr=false` (the default), then Xyce will not randomly sample any random operators present in the circuit. It will only sample the parameters listed by comma-separated lists.

7.5.3.7. Local and global variation

Similar to other simulators, Xyce uses the following convention for local and global variations sampling and other uncertainty quantification methods. This is based on how many layers of `.param` indirection sit between a device parameter and the random operator. If there is only a single layer, then the random operator is treated as a local variation, and every affected device parameter will receive a unique random number at each sample point. If there is more than one layer, then it is instead treated as global, meaning that each affected device parameter will receive the same random number from the operator.

For example, the following netlist fragment will result in local variation, where the resistors $r1$, $r2$ and $r3$ each get unique random numbers.

Example:

```
sampling example with random expression operators
R4 b 0 {agauss(4k,0.4k,1)}
R3 a b {agauss(1k,0.1k,1)}
R2 1 a {agauss(2k,0.2k,1)}
R1 1 2 {agauss(3k,0.1k,1)}
v1 2 0 10V

.dc v1 10 10 1
.print dc v(1)

.SAMPLING
+ useExpr=true

.options SAMPLES numsamples=10
+ outputs={v(1)}
+ sample_type=lhs
+ stdoutput=true
.end
```

Figure 7-8. Voltage divider circuit netlist using random expression operators.

```
.param resval=aunif(1000,400)
r1 1 2 resval
r2 2 3 resval
r3 3 4 resval
```

In contrast, the following netlist fragment will result in global variation, where the resistors will all get the same random number.

Example:

```
.param globalval=aunif(1000,400)
.param resval=globalval
r1 1 2 resval
r2 2 3 resval
r3 3 4 resval
```

7.5.4. Output files

The two types of sampling produce different types of output files. This is related to the fact that for `.SAMPLING` the calculations for each sample point are performed sequentially, while for `.EMBEDDEDSAMPLING` the calculations for each sample point are performed simultaneously. As such, for `.SAMPLING` statistical moments can only be computed at the end of the calculation, once all the sample points are avaible. In contrast, for `.EMBEDDEDSAMPLING`, statistical moments can be computed throughout the calculation, as all the sample points are available for each step.

7.5.4.1. `.SAMPLING` output files

Users can think of `.SAMPLING` simulations as being very similar to `.STEP` and thus comprised of a sequential set of distinct executions of the same circuit netlist. The main difference is the source of the

specific sample points.

Similar to .STEP, the output data can be requested by a standard .PRINT command, for the underlying primary analsis. For example, if applying .SAMPLING to a transient analysis, then the appropriate .PRINT command will be .PRINT TRAN. The output data will go to a single file, with each sample point in a different section of the output file. If using the default .prn file format (format=std), the resulting .SAMPLING simulation output file will be a simple concatenation of each step's underlying analysis output. If using format=probe, the data for each execution of the circuit will be in distinct sections of the file, and it should be easy to plot the results using PROBE. If using format=tecplot, the results of each .SAMPLING simulation will be in a distinct Tecplot zone. Finally, format=raw will place the results for each .SAMPLING simulation in a distinct "plot" region.

Another similarity to .STEP is that Xyce creates a second auxilliary file with the *.res suffix, which contains the parameter values. This can be useful for seeing the exact sampling parameter values in a compact file.

Statistical outputs can also be produced (see section 7.5.5) for .SAMPLING in the form of moments, but these can only be computed at the end of the calculation once all the samples are available. These statistical outputs will not be present in the .PRINT specified output file, since it is output on-the-fly.

Figure 7-6 shows an example file named clip.cir, which when run will produce files clip.cir.res and clip.cir.prn. The file clip.cir.res contains one line for each step, showing what parameter value was used on that step. clip.cir.prn is the familiar output format, but the INDEX field recycles to zero each time a new step begins. As the default behavior distinguishes each step's output only by recycling the INDEX field to zero, it can be beneficial to add the sampling parameters to the .PRINT line. For the default file format (format=std), Xyce will not automatically include these sampling parameters, so for plotting it is usually best to specify them by hand.

Note that for sampling calculations involving a really large number of sample points, the single output file can become prohibitively large. Be careful when using .PRINT if the number of samples is large. If the number is really large (thousands) consider excluding any .PRINT output statements and just rely on statistical outputs, described next in section 7.5.5. Similarly, the generation of the measure output can be suppressed with .OPTIONS MEASURE MEASPRINT.

7.5.4.2. .EMBEDDEDSAMPLING output files

As .EMBEDDEDSAMPLING computes all the sample points simultaneously, outputs for all sample points are also available simulataneously. For transient simulations, this means that all samples are available at every time step, and all statistical outputs are available at every time step. This is very useful if the interest is in seeing statistical moments with respect to time

The output command used to produce this kind of output is .PRINT ES, where ES stands for "embedded sampling". The .PRINT ES command will automatically output several moments of the specified PCE outputs without any additional arguments on the .PRINT ES line. Also, the .PRINT ES statement will result in a file that has "ES" as part of the file name suffix. Otherwise, .PRINT ES behaves very similarly to any other type of .PRINT command, and supports all of the same file formats and options. For a full description see the Xyce Reference Guide.

7.5.5. Statistical Outputs

When performing uncertainty quantification analysis, the outputs of interest are often statistical moments of different circuit metrics. Xyce will compute these at the end of a sampling calculation upon request. The requested statistical outputs are specified on the `.options SAMPLING` line in the netlist. For an example see the `OUTPUTS` and `MEASURES` fields in section 7.5.2. Both `OUTPUTS` and `MEASURES` are specified as comma-separated lists. The list of `OUTPUTS` must contain valid expressions that can be processed by the Xyce expression library. The list of `MEASURES` must consist valid `.MEASURE` statement names, which are present elsewhere in the netlist.

The example netlist given by figure 7-5 will produce the result given by figure 7-9. The example netlist given by figure 7-6 will produce the result given by figure 7-10. In both examples, the number of samples is relatively small, so the exact quantities computed will vary somewhat from run to run.

```
MC sampling mean of R1:R = 1009.58
MC sampling stddev of R1:R = 100.131
MC sampling variance of R1:R = 10026.3
MC sampling skew of R1:R = -0.0208347
MC sampling kurtosis of R1:R = 2.79565
MC sampling max of R1:R = 1308.03
MC sampling min of R1:R = 656.039

MC sampling mean of maxSine = 874.063
MC sampling stddev of maxSine = 10.9115
MC sampling variance of maxSine = 119.061
MC sampling skew of maxSine = 0.08628
MC sampling kurtosis of maxSine = 2.83029
MC sampling max of maxSine = 914.274
MC sampling min of maxSine = 842.547
```

Figure 7-9. Typical statistical outputs for figure 7-5

```
LHS sampling mean of R4:R = 9198
LHS sampling stddev of R4:R = 4293.89
LHS sampling variance of R4:R = 1.84375e+07
LHS sampling skew of R4:R = -0.0939244
LHS sampling kurtosis of R4:R = 1.39595
LHS sampling max of R4:R = 14996
LHS sampling min of R4:R = 3315.6

LHS sampling mean of D1N3940:IS = 4.2571e-10
LHS sampling stddev of D1N3940:IS = 9.10744e-11
LHS sampling variance of D1N3940:IS = 8.29455e-21
LHS sampling skew of D1N3940:IS = 0.0486504
LHS sampling kurtosis of D1N3940:IS = 2.05108
LHS sampling max of D1N3940:IS = 5.78543e-10
LHS sampling min of D1N3940:IS = 2.85753e-10

LHS sampling mean of maxSine = 4.46494
LHS sampling stddev of maxSine = 0.0901329
LHS sampling variance of maxSine = 0.00812394
LHS sampling skew of maxSine = -0.682188
LHS sampling kurtosis of maxSine = 2.07339
LHS sampling max of maxSine = 4.5536
LHS sampling min of maxSine = 4.28396
```

Figure 7-10. Typical statistical outputs for figure 7-6

7.5.6. Random Sampling and Running in Parallel

It is well known that sampling is a good opportunity to apply parallelism, as zero communication is required between sample points. However, it has only been applied to a limited degree to random sampling in Xyce. This is mostly because sampling methods are a relatively new feature to Xyce. To get the full benefit of parallel sampling, where each sample can be launched and executed on a different unique processor, it is recommended that Xyce users instead use the Dakota library [15], [16]. However, despite the limitations of Xyce sampling in parallel, note that:

- Sampling in Xyce does have *some* available parallelism, just not applied on a sample basis.
- The cost of sampling can be significantly mitigated using non-intrusive Polynomial Chaos Expansion (PCE) methods, described in section 7.6
- Xyce sampling is performed entirely internal to the Xyce binary. So unlike the black-box form of Dakota, it does not involve any file I/O. Also it does not require any scripting, which can be error-prone. Finally, the netlist parsing and setup only happens once, rather than for every sample point, and this mitigates some of the computational cost.

7.5.6.1. `.SAMPLING` in Parallel

The `.SAMPLING` analysis will execute the underlying analysis sequentially, once for each sample point. If using parallel Xyce, then each of these sequential calculations will have the same parallel methods applied to them as would be applied to a single forward calculation. For example, if running a calculation of 100 samples on a large parallel transient calculation, Xyce would perform the first transient simulation in parallel, followed by the second transient simulation in parallel, etc, until all 100 samples were complete. This means that if the underlying analysis is large enough to get a benefit from parallel, then the `.SAMPLING` analysis will get this same benefit. The parallel scaling for the underlying analysis is unlikely to be perfect, as there are communication costs involved in the forward calculation. However, one of the main parallel bottlenecks, the parsing and setup phase, is substantially reduced as it only happens one time.

7.5.6.2. `.EMBEDDEDSAMPLING` in Parallel

The `.EMBEDDEDSAMPLING` algorithm sets up a large block matrix system, where each block represents a different sample point. This means that for each step of the calcalculation, all the samples are computed simultaneously. This block system can be solved in parallel. However, as this method is less mature in Xyce, more refinements to the algorithm are needed for this `.EMBEDDEDSAMPLING` to fully benefit from running in parallel.

7.6. Polynomial Chaos Expansion (PCE) methods

Xyce supports several styles of Polynomial Chaos expansion (PCE) methods, which are stochastic expansion methods that approximate the functional dependence of a simulation response on uncertain model parameters by expansion in a polynomial basis [17, 18]. In Xyce the Stokhos library [19] has been used to implement generalized PCE methods using the Wiener-Askey scheme[17].

To propagate input uncertainty through a model using PCE, Xyce performs the following steps: (1) input uncertainties are transformed to a set of uncorrelated random variables, (2) a basis such as Hermite polynomials is selected, and (3) the parameters of the functional approximation are determined. The general PCE for a response O has the form:

$$O(p) \approx \hat{O}(p) = \sum_{i=0}^{P} \alpha^i \Psi^i(p), \tag{7.1}$$

where each multivariate basis polynomial $\Psi^i(p)$ involves products of univariate polynomials that are tailored to the individual random variables. More details about these methods and their Xyce implementation can be found in reference [20] and the references therein.

Xyce supports three versions of polynomial chaos. They mainly differ in how the coefficients α in equation 7.1 are computed. These three methods are:

- Non-intrusive regression polynomial chaos (section 7.6.1)
- Non-intrusive spectral projection (NISP) (section 7.6.2)
- Fully intrusive spectral projection (section 7.6.3)

The first two versions are non-intrusive, meaning that they don't directly modify the forward problem being solved. Instead, they are augmentations to the `.SAMPLING` and `.EMBEDDEDSAMPLING` analysis methods. The fully intrusive method is a separate analysis type, and is invoked using the `.PCE` command.

7.6.1. Regression-based Polynomial Chaos

Regression-based PCE is briefly described in this section. Regression-based PCE approaches aim to find the best fit for the linear system:

$$\Psi\alpha \approx R \tag{7.2}$$

for a set of PCE coefficients α that best reproduce a set of response values R. The orthogonal polynomials used in the PCE approximation are represented by the matrix Ψ. This matrix is usually a rectangular matrix.

The regression approach finds a set of PCE coefficients α^i which best fit a set of response values obtained from a sampling study on the density function of the uncertain parameters [21]. One advantage of regression-based methods is that they are very flexible with respect to the exact sample points used to evaluate the responses, and can readily be used with standard sampling methods such as Monte Carlo and Latin Hypercube Sampling. The method that has been implemented in Xyce for solving Eq. (7.2) is least squares regression.

Regression-based PCE can be applied as a post-process step to `.SAMPLING` and `.EMBEDDEDSAMPLING`. It is invoked by adding `regression_pce=true` to the `.options SAMPLES` or `.options EMBEDDEDSAMPLES` command lines.

Regression PCE will use the samples selected by the sampling algorithm (Monte Carlo or Latin Hypercube). The regression PCE algorithm will *not* estimate the optimal number of samples to use, and so this must be set by the user. For regression PCE, the number of samples needed for an accurate answer can be smaller than the number that would have been required by conventional sampling. For regression problems, to get an accurate answer it is considered good practice to oversample the basis size by at least a factor of 2.

An example netlist, which uses regression PCE with `.SAMPLING` is given in figure 7-11. Results from this calculation, which are sent to terminal output are given in figure 7-12. In this calculation, the polynomial order has been set to `5`, the output which is subject to the PCE analysis is `v(1)`, the type of sampling used is Latin Hypercube Sampling (LHS), and the results of the analysis are sent to terminal output. For this example, the uncertain parameter is `testNorm`, which is set by the `.param testNorm={aunif(2k,1k)}` statement. On the `.SAMPLING` line, the command `useExpr=true` instructs the sampling method to rely on expression-based random expressions. Without this line, the sampling algorithm would ignore `testNorm`.

The parameter `resample=true` is a PCE-specific command. Once the PCE approximation has been completed, the resulting polynomial can be thought of as a surrogate model, and this surrogate can then be sampled, much like the original forward problem. When the surrogate is sampled, it uses 1000 samples. Since this calculation is sampling a polynomial, it can do a really large number of evaluations without incurring much computational expense. The result of this resampling is (in the example) also sent to the console, as can be seen at the bottom of figure 7-12.

```
Regression PCE example
.param testNorm={aunif(2k,1k)}
.param R1value={testNorm*2.0}
R2 1 0 6K
R1 1 2 {R1value}
v1 2 0 1000V

.dc v1 1000 1000 1
.print dc format=tecplot v(1)
*
.SAMPLING useExpr=true

.options SAMPLES numsamples=12
+ regression_pce=true
+ order=5
+ outputs={v(1)}
+ sample_type=lhs
+ stdoutput=true
+ resample=true
.end
```

Figure 7-11. Voltage divider regression PCE netlist. The sampling-related commands are in red.

```
***** Beginning Latin Hypercube Sampling simulation....

***** Regression PCE enabled.  Number of sample points = 12

***** PCE Basis size = 6

Seeding random number generator with 2412823737

LHS sampling mean of V(1) = 614.666
LHS sampling stddev of V(1) = 71.4563
LHS sampling variance of V(1) = 5106
LHS sampling skew of V(1) = 0.299928
LHS sampling kurtosis of V(1) = 2.07869
LHS sampling max of V(1) = 733.936
LHS sampling min of V(1) = 503.315

(traditional sampling) regression PCE mean of V(1) = 608.197
(traditional sampling) regression PCE stddev of V(1) = 71.3831

Statistics from re-sampling regression PCE approximation of V(1):
mean     = 608.071
stddev   = 70.8532
variance = 5020.18
skew     = 0.298245
kurtosis = 1.93151
max      = 749.652
min      = 500.054
```

Figure 7-12. Voltage divider regression PCE result. There are 3 sets of statistics in this ouptut. The first set is computed from the 12 LHS sample points. As this is a PCE calculation, the number of points is small and so the statistics based on them are probably not very accurate. The second set is computed from the PCE approximation, which are purely analytical. The third set is from running 1000 samples on the PCE approximation. This output is enabled by `resample=true`.

7.6.2. Non-Intrusive Spectral Projection (NISP)

The spectral projection PCE approach projects the response against each basis function $\Psi_j(p)$ using inner products and employs the polynomial orthogonality properties to extract each coefficient. Each inner product involves a multidimensional integral over the support range of the weighting function, which can be evaluated numerically using sampling, tensor-product quadrature, Smolyak sparse grid [22], or cubature [23] approaches. For the Xyce implementation this means evaluating the following equation for each coefficient using quadrature rules:

$$\alpha^i = \frac{\langle O\Psi^i \rangle}{\langle (\Psi^i)^2 \rangle} = \frac{1}{\langle (\Psi^i)^2 \rangle} \int_\Gamma O(p;y)\Psi^i(y)\rho(y)dy = 0, \quad i = 0,\ldots,P. \tag{7.3}$$

In this equation ρ is the joint density function, with respect to which the polynomials Ψ^i are orthogonal. Evaluating this equation requires solving the integral on the right hand side. While many methods for solving this integral could be used, for the NISP algorithm the integral is solved using Gaussian Quadrature and related methods. For this reason, these methods are sometimes referred to as "quadrature PCE" methods. Stokhos supports several quadrature methods, and supplies the necessary quadrature points to Xyce for the specified details.

To solve the integral using quadrature, Xyce has to evaluate the forward problem at the various quadrature points. In this reguard, this is similar to the regression methods described in section 7.6.1, which has to do forward evaluations at randomly sampled points. Since NISP uses quadrature points, the points are completely deterministic, and there are no random numbers involved. The number of quadrature points can increase exponentially for larger numbers of uncertain parameters, but this can be mitigated by using sparse grid techniques.

Non-intrusive spectral projection PCE can be applied as a post-process step to `.SAMPLING` and `.EMBEDDEDSAMPLING`. It is invoked by adding `projection_pce=true` to the `.options SAMPLES` or `.options EMBEDDEDSAMPLES` command lines.

As noted, when using NISP it is not necessary to specify the number of sample points, as they will be set by the quadrature algorithm. To enable quadrature on a sparse grid, set the option `sparse_grid=true` on the `.options EMBEDDEDSAMPLES` command line.

An example netlist, which uses NISP with `.EMBEDDEDSAMPLING` is given in figure 7-13. Results from this calculation, which are output using the `.PRINT ES` command are given in figure 7-14. The `.PRINT ES` command will automatically output several moments of the specified PCE outputs (in this case `V(drain)` and `V(gate)`) without any additional arguments on the `.PRINT ES` line. In the example, additional information is requested using the `OUTPUT_PCE_COEFFS=TRUE` command, which will result in all of the PCE coefficients (the various α^i given by equation 7.3).

A `.SAMPLING` version of the netlist given in figure 7-13 can also be constructed, by simply replacing `.EMBEDDEDSAMPLING` with `.SAMPLING` and replacing `.options EMBEDDEDSAMPLES` with `.SAMPLES`. However, the behavior of `.SAMPLING` analysis will be a bit different. As `.SAMPLING` performs the calculation for each quadrature point sequentially, rather than simultaneously, there is no equivalent of `.PRINT ES` for conventional sampling. All of the `.SAMPLING` output is handled by the `.PRINT TRAN` command.

Additionally, since the PCE analysis cannot be performed until the end of a `.SAMPLING` analysis, it is more useful to have the analysis applied to `.MEASURE` commands which produce metrics for the entire simulation. Applying the PCE analysis to voltage node values in this case is less useful, as it will simply use the final values for each voltage node (which will be mostly invariant in this example).

```
* embedded sampling with NISP, CMOS inverter
.param inFreq=1e6, inPer={1/inFreq}
.param td={inPer/2-inPer/10}
.param tr={inPer/10}, tf={inPer/10}
.param pw={inPer/2-inPer/10}, per={inPer}

VS_VDD vdd 0 DC 1.0
Vin gate 0 pulse( 0.0 1.0 {td} {tr} {tf} {pw} {per} )
M1 drain gate 0 0 NMOS w=0.18e-6 l=0.18e-6
M2 drain gate vdd vdd PMOS  w =0.54e-6

.model nmos nmos (level=9 tnom=27 nch=2.3549e17 vth0={aunif(0.55,0.15)}
+ uc=5.105195e-11 ags=0.4044483 keta=-2.730673e-3 rdsw=105 wr=1
+ dwg=-3.773529e-9 dwb=5.239518e-9 cit=0 cdscb=0 dsub=0.0167866
+ pdiblc2=3.38377e-3 pscbe1=8e10 delta=0.01 prt=0 kt1l=0 ub1=-7.61e-18
+ wl=0 wwn=1 lln=1 lwl=0 cgdo=7.9e-10 cj=9.539798e-4
+ cjsw=2.53972e-10 cjswg=3.3e-10 cf=0 pk2=-4.42044e-4 pu0=10.8203648
+ pvsat=1.388017e3 tox=4.1e-9 k2=1.63871e-3 w0=1e-7 dvt1w=0
+ dvt1=0.3683763 ua=-1.504141e-9 vsat=9.896282e4 b0=-4.706134e-8
+ a1=5.916677e-4 prwg=0.5 wint=0 voff=-0.0883818 cdsc=2.4e-4
+ eta0=2.470369e-3 pclm=0.7326932 pdiblcb=-0.1 pscbe2=1.254966e-9
+ rsh=6.5 ute=-1.5 kt2=0.022 uc1=-5.6e-11 wln=1 wwl=0 lw=0
+ capmod=2 cgso=7.9e-10 pb=0.8 pbsw=0.8 pbswg=0.8 pvth0=7.505753e-4
+ wketa=3.100384e-3 pua=2.896652e-11 peta0=8.758549e-5 toxm=4.1e-9
+ xj=1e-7 k1=0.6064385 k3b=2.763267 dvt0w=0 dvt0=1.3330881 u0=258.9066683
+ nlx=1.71872e-7 dvt2w=0 dvt2=0.0540199 ub=2.428646e-18 a0=1.8904342
+ k3=1e-3 b1=1.294942e-6 a2=0.9069159 prwb=-0.2 lint=1.69494e-8
+ nfactor=2.1821266 cdscd=0 etab=1.047744e-5 pdiblc1=0.1823102
+ drout=0.7469045 pvag=0 mobmod=1 kt1=-0.11 ua1=4.31e-9 at=3.3e4
+ ww=0 ll=0 lwn=1 xpart=0.5 cgbo=1e-12 mj=0.380768 mjsw=0.1061193
+ mjswg=0.1061193 prdsw=-2.7650517 lketa=-0.0104103
+ pub=1.684125e-23 pketa=1.549791e-3)

.model pmos pmos (level=9 tnom=27 nch=4.1589e17 vth0={-aunif(0.55,0.15)}
+ k2=0.0258663 w0=1e-6 dvt1w=0 dvt0=0.6117215 u0=106.5280265 uc=-1e-10
+ ags=0.3667554 keta=0.0237092 rdsw=304.9893888 wr=1 dwg=-2.44019e-8
+ dwb=-9.06003e-10 cit=0 cdscb=0 dsub=1.0998181 pdiblc2=0.0420477
+ pscbe1=1.073111e10 delta=0.01 prt=0 kt1l=0 ub1=-7.61e-18
+ wl=0 wwn=1 lln=1 lwl=0 cgdo=6.41e-10 cj=1.200422e-3
+ cjsw=2.001802e-10 cjswg=4.22e-10 cf=0 pk2=1.799383e-3 pu0=-1.3399122
+ pvsat=-50 l=0.18e-6 tox=4.1e-9 k3=0 nlx=1.20187e-7
+ dvt2w=0 dvt1=0.2286816 ua=1.125454e-9 vsat=1.593712e5 b0=5.263128e-7
+ a1=0.2276342 prwg=0.5 wint=0 voff=-0.0878287 cdsc=2.4e-4 eta0=0.1672562
+ pclm=2.2249148 pdiblcb=-1e-3 pscbe2=3.099395e-9 rsh=7.4 ute=-1.5
+ kt2=0.022 uc1=-5.6e-11 wln=1 wwl=0 lw=0 capmod=2 cgso=6.41e-10
+ pb=0.8478616 pbsw=0.8483594 pbswg=0.8483594 pvth0=2.098588e-3
+ wketa=0.0295614 pua=-5.27759e-11 peta0=1.003159e-4 toxm=4.1e-9
+ xj=1e-7 k1=0.59111 k3b=7.9143108 dvt0w=0 dvt2=0.1 ub=1e-21 a0=1.6904754
+ b1=1.496707e-6 a2=0.6915706 prwb=0.2553725 lint=3.217673e-8
+ nfactor=1.8560303 cdscd=0 etab=-0.1249603 pdiblc1=8.275696e-4
+ drout=0 pvag=15 mobmod=1 kt1=-0.11 ua1=4.31e-9 at=3.3e4 ww=0 ll=0
+ lwn=1 xpart=0.5 cgbo=1e-12 mj=0.4105254 mjsw=0.3400571
+ mjswg=0.3400571 prdsw=4.4771801 lketa=-1.935751e-3
+ pub=1e-21 pketa=-3.434535e-3)

.EMBEDDEDSAMPLING useExpr=true
.options EMBEDDEDSAMPLES outputs={v(drain)},{V(gate)} projection_pce=true
.PRINT ES OUTPUT_PCE_COEFFS=TRUE FORMAT=tecplot
.tran 1ns 0.75e-6
.PRINT TRAN v(drain) V(gate)
.END
```

Figure 7-13. CMOS Inverter Non-intrusive Spectral Projection (NISP) netlist. The NISP-related commands are in red.

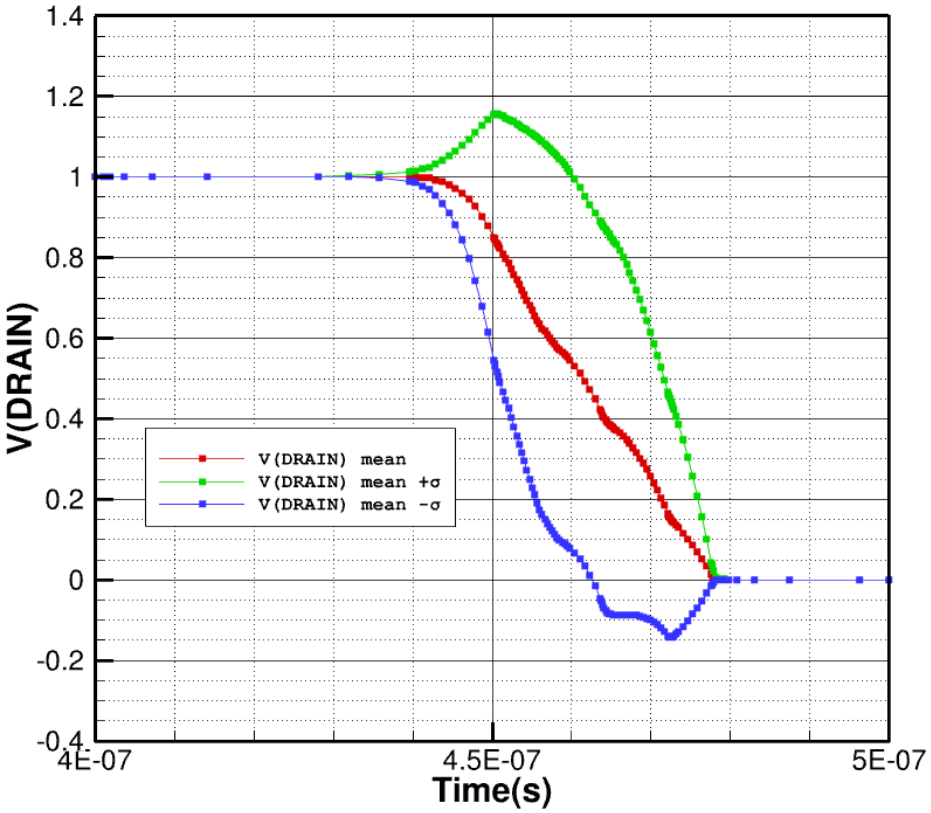

Figure 7-14. CMOS Inverter NISP Output

7.6.3. Fully Intrusive Spectral Projection

The fully intrusive form of PCE involves solving a system of equations which is constructed by applying the orthogonal polynomial expansion defined by Eq. (7.1) to all the terms of the differential-algebraic system of equations solved by Xyce. The details of this construction are given in reference [20]. This method is primarily experimental at this point, and is inherently more expensive, so it is not the best choice of PCE method in Xyce. Most `.PCE` use cases of interest can be computed much more cheaply and robustly using `.EMBEDDEDSAMPLING` combined with one of the non-intrusive PCE methods (regression PCE or NISP). Similar to `.EMBEDDEDSAMPLING` based calculations, `.PCE` computes PCE coefficients at every stage of the calculation.

An example netlist, for a diode clipper circuit which uses intrusive PCE, is given in figure 7-15. For this type of analysis, it is not necessary to specify `projection_pce=true`. The result for this circuit is given in figure 7-16.

```
Transient Diode Clipper with intrusive PCE analysis

.tran 2ns 2.0ms

.PCE useExpr=true
.options PCES
+ OUTPUTS=R4:R,D1N3940:IS outputs={V(4)}
.print pce format=tecplot

VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
D1 2 1 D1N3940
D2 0 2 D1N3940
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0  {aunif(9.0k,6.0k)}
C1 2 4 0.47u

.MODEL D1N3940 D(
+  IS={aunif(4.0e-10,2.0e-10)}
+ RS=.105 N=1.48 TT=8E-7
+ CJO=1.95E-11 VJ=.4 M=.38 EG=1.36
+ XTI=-8 KF=0 AF=1 FC=.9 BV=600 IBV=1E-4)
.END
```

Figure 7-15. Diode clipper fully intrusive PCE netlist. The PCE-related commands are in red.

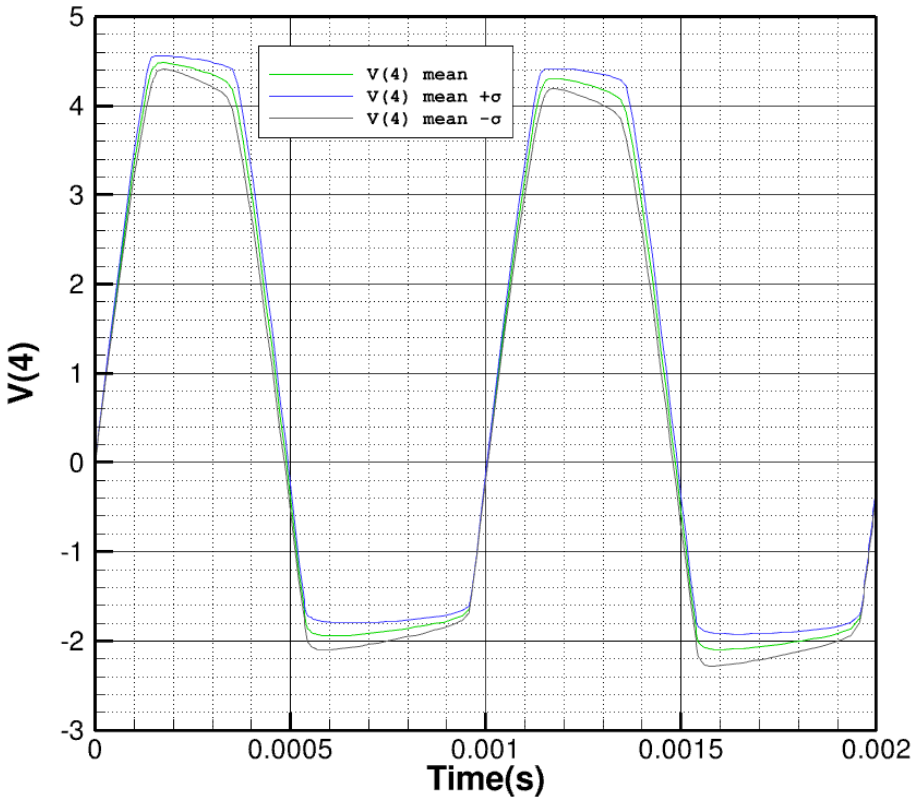

Figure 7-16. Diode clipper fully intrusive PCE Output

7.6.4. Comparing PCE methods: Gilbert Cell Mixer PCE example

A Gilbert cell mixer circuit is given in figure 7-17 [24]. A corresponding netlist is given in Fig. 7-18. In this circuit, the LO voltage is given by `v(6,2)` and `v(15,10)` is the input. The output is the difference between collector voltages of Q4/Q6 and Q3/Q5 (`V(5,3)`), and should be the approximate product of the LO and input voltages. In this example, all eight resistors are given uncertain resistances, distributed with the following normal distributions: R1 and R2 are N(10,1), R5 and R6 are N(100,5), and R3, R4, R7, and R8 are N(1500,50). These large standard deviations of the resistances are not realistic, but are chosen for demonstration purposes.

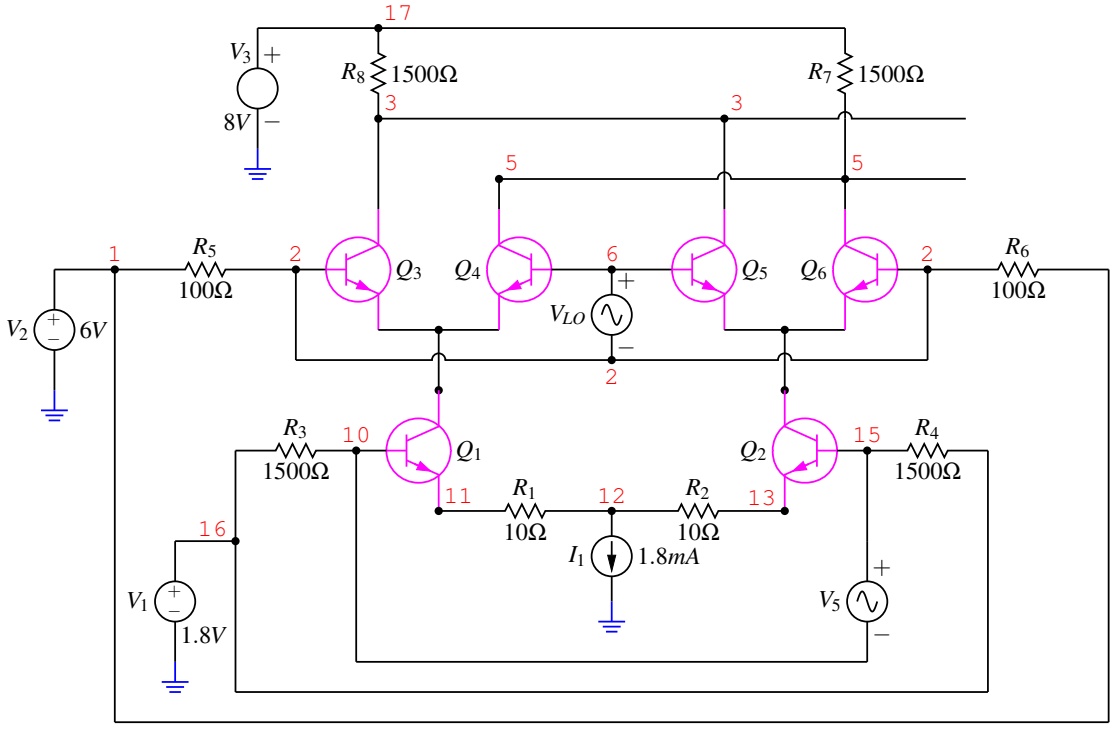

Figure 7-17. Gilbert cell mixer schematic.

Ensemble and PCE results for the Gilbert cell mixer are shown in figure 7-19. As this calculation was concerned with the transient behavior, only `.EMBEDDEDSAMPLING` and `.PCE` are used in this example. For `.EMBEDDEDSAMPLING`, both types of non-intrusive PCE are used. The light grey lines in Fig. 7-19 are for all the sample points of an embedded LHS study with 1000 samples. The red lines (which are obscured by other results) show the mean and the mean +/- twice the standard deviation for the 1000 sample study. This can be considered the "gold" standard for this simulation. The other solid lines show the moments from the various PCE calculations, and they lie on top of the 1000 LHS sample moments.

This version of the netlistgiven in Fig. 7-18 is using non-intrusive projection PCE as an augmentation to `.EMBEDDEDSAMPLING`. To convert this netlist to use the fully intrusive method, change `.EMBEDDEDSAMPLING` to `.PCE`, and change `.options EMBEDDEDSAMPLES` to `.options PCES`. Otherwise, the commands are completely the same. As the number of uncertain parameters is a little bit larger (8), for the projection-based methods a Smolyak sparse grid [22] method is used. This is set with the `sparse_grid=true` parameter. Using the sparse grid, the number of required quadrature points for this problem is 129. Using the non-sparse Gaussian quadrature would have required 6561 quadrature points.

This would have rendered the method much less efficient than sampling, and basically not usable, especially for the `.PCE` method.

To change the netlist to use non-intrusive regression PCE, change `projection_pce=true` to `regression_pce=true`. Delete the `sparse_grid=true` parameter, as it isn't relevant to regression PCE.

As noted, for regression PCE the number of parameters needs to be set. Unlike projection-based methods, regression PCE does not have a rigid requirement for the number of samples. For this problem the size of the polynomial basis is 45, and for regression problems it is considered good practice to oversample by at least a factor of 2, which implies that a good choice for this circuit is 90 sample points. To set this, add `numsamples=90` on the `.options EMBEDDEDSAMPLES` line. It appears that 90 sampling points was adequate for the regression form of PCE, as the all the statistical moments match well throughout the transient.

NISP is used in conjunction with embedded sampling, and as noted requires 129 sample points if using sparse grid quadrature. Fully intrusive PCE, which uses all the same machinery as NISP also requires 129 quadrature points.

```
* Gilbert cell mixer
R5 1 2 {agauss(100,5,1)}
Q3 3 2 4 QB2T2222
Q4 5 6 4 QB2T2222
Q5 3 6 8 QB2T2222
Q6 5 2 8 QB2T2222
R6 2 1 {agauss(100,5,1)}
*the local oscillator
VLO 6 2 DC 0 SIN(0 .05V 4e7 0 0)
Q1 4 10 11 QB2T2222
R1 11 12 {agauss(10,1,1)}
R2 12 13 {agauss(10,1,1)}
* input bias current
I1 12 0 DC 1.8mA
Q2 8 15 13 QB2T2222
R4 15 16 {agauss(1500,50,1)}
R3 16 10 {agauss(1500,50,1)}
V1 16 0 DC 1.8V
*the input voltage to be mixed with the LO
V5 15 10 DC 0 sin(0 .05V 3e6 0 0)
R7 5 17 {agauss(1500,50,1)}
R8 3 17 {agauss(1500,50,1)}
V3 17 0 DC 8V
V2 1 0 DC 6V

.MODEL QB2T2222  NPN (
+ IS=3.136905E-14 BF=189 NF=0.9977664 VAF=29.7280913
+ IKF=0.7405619 ISE=8.49314E-15 NE=1.3186316 BR=38.9531224
+ NR=0.9833179 VAR=28.5119855 IKR=4.632395E-3 ISC=4.624043E-14
+ NC=1.225899 RB=2.0158781 IRB=0.0227681 RBM=0.9730261
+ RE=0.0501513 RC=0.6333 CJE=2.369655E-11 VJE=0.6884357
+ MJE=0.3054743 TF=5.3E-10 XTF=48.5321578 VTF=5.3020062
+ ITF=1.1 PTF=14.6099207 CJC=8.602583E-12 VJC=0.4708887
+ MJC=0.3063885 XCJC=1 TR=74E-9 CJS=1e-12 VJS=.75
+ MJS=0 XTB=0.87435 EG=1.11 XTI=5.825
+ KF=0 AF=1 FC=0.5)

.TRAN 1ns 3e-7
.PRINT TRAN format=tecplot v(6,2) v(15,10) v(5,3)
.options timeint reltol=1.0e-5

.EMBEDDEDSAMPLING useExpr=true
.options EMBEDDEDSAMPLES projection_pce=true
+ outputs=v(5,3) order=2 sparse_grid=true
.end
```

Figure 7-18. Gilbert Cell Non-Instrusive Spectral Project (NISP) netlist. UQ commands are in red, and NISP is specified using the `projection_pce` parameter.

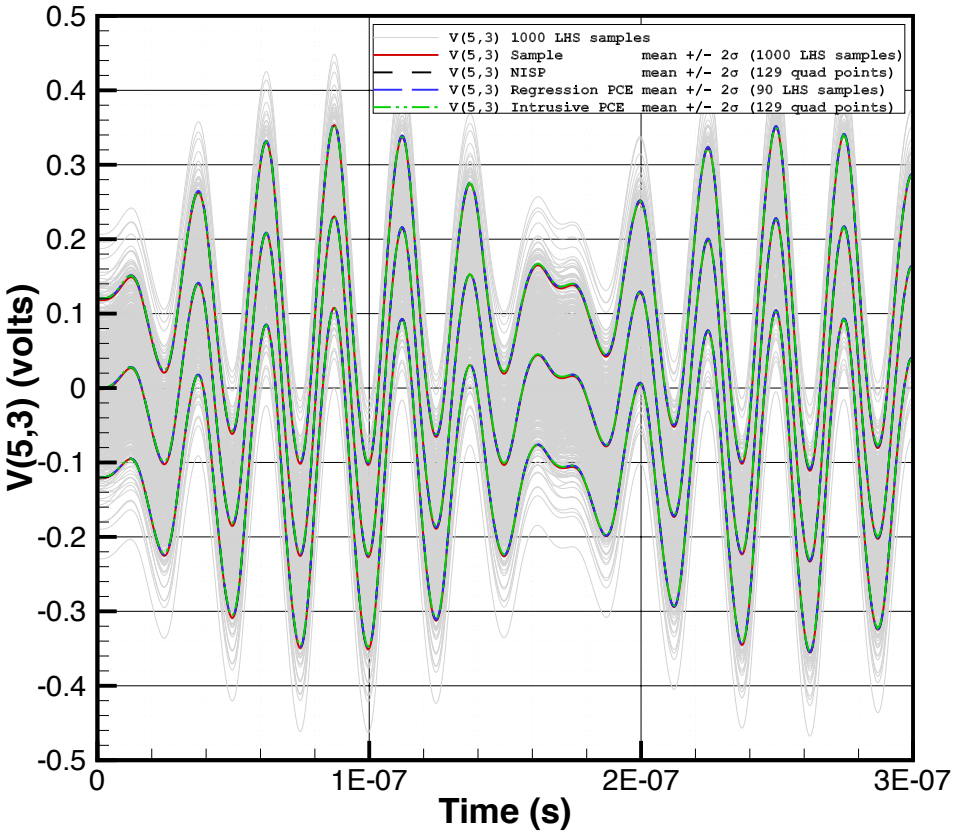

Figure 7-19. Gilbert Cell Mixer Output for several Xyce embedded UQ methods. The mean and standard deviations for all the methods (Embedded LHS sampling, NISP, Regression PCE and fully intrusive PCE) all match very well.

7.7. Harmonic Balance Analysis

Harmonic balance (HB) is a technique that solves for the steady state solution of nonlinear circuits in the frequency domain. In harmonic balance simulation, voltages and currents in a nonlinear circuit are represented by truncated Fourier series. HB directly computes the frequency spectrum of voltages and currents at the steady state solution. This can be more efficient than transient analysis in applications where a transient analysis may take a long time to reach the steady state solution. In particular, HB is well suited for simulating analog RF and microwave circuits.

HB supports an unlimited number of independent input tones for driven circuits in both serial and parallel builds of Xyce. The output of HB analysis is the real and imaginary components of voltages and currents in the frequency domain. Xyce also provides time domain responses of a circuit. By default, the numbers of samples in both time and frequency domain outputs are the same. However, the number of time domain samples can be much larger than the number of frequency domain samples by setting `numtpts` in `.options hbint`. This enables the users to use a small number of harmonics for each tone and produce well-resolved results in time domain.

For HB analysis, an initial guess of the solution is required. Often, a good initial guess is necessary for HB to converge. By default, Xyce uses transient analysis to determine the initial guess, often called "Transient Assisted HB". This option is controlled by the parameter, `TAHB`, which is enabled if `TAHB=1` and disabled if `TAHB=0` in `.options hbint`. If enabled, the starting time of the transient analysis can be modified by specifying the parameter `STARTUPPERIODS` in `.options hbint`. The Xyce Reference Guide [3] provides a detailed explanation of the HB options.

7.7.1. .HB Statement

To run a HB simulation, the circuit netlist file must contain a `.HB` command.

Example:

```
.HB 1e4
.HB 1e4 2e2
```

The parameters following `.HB` are the fundamental frequencies and **must** be specified by the user. The Xyce Reference Guide [3] provides a detailed explanation of the `.HB` statement.

7.7.2. HB Options

Key parameters for `.HB` simulation can be specified by `.options hbint`.

Example:

```
.options hbint numfreq=5,3 STARTUPPERIODS=2 tahb=1 intmodmax=3
numtpts=100
.HB 1e4 2e2
```

As shown in the example, `numfreq` specifies the number of harmonics to be calculated for each tone. It must have the same number of entries as the fundamental frequencies in the `.HB` statement. This example

shows the options for a two tone HB analysis. The `.HB` statement is `.HB` f_1 f_2 . For the first tone f_1, 5 harmonics are used and for the second tone f_2, 3 harmonics are used.

The `intmodmax` option is the maximum intermodulation product order used in the spectrum.

As stated above, `TAHB` is the Transient Assisted HB option. In the case of multi-tone HB analysis, the initial guess is generated by a single tone transient simulation. The single tone that is used is the first tone in the `.HB` statement. So, when ordering the fundamental frequencies in the `.HB` statement, the first tone should be the frequency that produces the most nonlinear response by the circuit.

As stated above, the `STARTUPPERIODS` option specifies the number of time periods that Xyce will integrate through using normal transient analysis **before** generating the initial conditions for HB analysis. For this example, Xyce will integrate through two periods before computing the initial conditions, which requires an additional period. Thus, Xyce will integrate through a total of three periods to compute the initial guess for HB analysis.

The `numtpts` option specifies the number of time points in the output. The number of samples in the time domain output is an odd number. If an even number is specified, the number of time points will be `numtpts` plus one. For this example, the time domain output has 101 samples.

Example:

```
.options hbint numfreq=20 STARTUPPERIODS=2
```

For the single tone example given above, this statement will return 20 negative and 20 positive harmonics plus the DC component.

The Xyce Reference Guide [3] provides a detailed explanation of the `.options hbint`.

7.7.2.1. Nonlinear Solver Options

The HB analysis uses a different set of default nonlinear solver parameters than that of transient and DC analysis. Nonlinear solver parameters for HB analysis can be specified by using `.options nonlin-hb`.

Example:

```
.options nonlin-hb abstol=1e-6
```

The Xyce Reference Guide [3] provides a detailed explanation of the `.options nonlin-hb` statement.

7.7.2.2. Linear Solver Options

The HB analysis provided by Xyce can employ both iterative and direct solvers. If iterative solvers are used, then the HB Jacobian matrix is not assembled and only HB-specific preconditioners can be used for this matrix-free approach. If direct solvers are used, then the HB Jacobian is assembled and the complex-valued matrix is solved using the selected method. Direct solvers are memory intensive and computationally expensive, so it is suggested that they are only used when iterative methods fail during HB analysis. You can set iterative solver and preconditioner options using the `.options linsol-hb` statement.

Example:

```
.options linsol-hb type=aztecoo prec_type=block_jacobi AZ_tol=1e-9
```

In this example, `type` specifies the iterative solver to use in the HB analysis and `AZ_tol` is the relative tolerance for the iterative solver. Any of the iterative solver options in section 11.4 are valid. However, `prec_type` specifies which HB-specific preconditioner to use. The choices for this option are `none` and `block_jacobi` (default).

7.7.3. Output

HB analysis can generate a variety of output files. The selection of which files are generated most often depends on what is specified for the `.PRINT HB` command. Table 7-5 lists the format options and files created. The column labeled "Additional Columns" lists the additional data that is written, though not specified on the `.PRINT HB` line.

7.7.4. User Guidance

One of the most common errors in the HB simulation setup is the use of too few harmonic frequencies (i.e., `numfreq` is too small). One way to determine the optimum number of harmonic frequencies is to first simulate the circuit with a small `numfreq`, then increase the `numfreq` until the solution stops changing within a significant bound. Requesting too many harmonic frequencies is wasteful of memory and simulation time, so it is not practical to request a very high `numfreq` either.

7.8. AC Analysis

The AC small-signal analysis of Xyce computes AC output variables as a function of frequency. The program first computes the DC operating point of the circuit and linearizes the circuit. The resultant linear circuit is then analyzed over a user-specified range of frequencies. The desired output of an AC small-signal analysis is usually a transfer function (voltage gain, transimpedance, etc). If the circuit has one AC input, it is convenient to set that input to unity and zero phase so output variables have the same value as the transfer function of the output variable with respect to input.

7.8.1. .AC Statement

One may specify `.AC` analyses by adding a `.AC` line in the netlist. Some examples of typical `.AC` lines include:

Example:

```
.AC DEC 10 1K 100MEG
.AC DEC 10 1 10K
.AC LIN 100 1 100HZ
.AC DATA=table
.param init=10, final=1K, step=100MEG
.AC DEC {init} {final} {step}
```

Table 7-5. Output generated for HB analysis

Command	Files	Additional Columns
`.PRINT HB`	*circuit-file*.HB.TD.prn *circuit-file*.HB.FD.prn	INDEX TIME INDEX FREQUENCY
`.PRINT HB_TD FORMAT=NOINDEX` `.PRINT HB_FD FORMAT=NOINDEX`	*circuit-file*.HB.TD.prn *circuit-file*.HB.FD.prn	TIME FREQUENCY
`.PRINT HB_TD FORMAT=CSV` `.PRINT HB_FD FORMAT=CSV`	*circuit-file*.HB.TD.csv *circuit-file*.HB.FD.csv	TIME FREQUENCY
`.PRINT HB_TD FORMAT=TECPLOT` `.PRINT HB_FD FORMAT=TECPLOT`	*circuit-file*.HB.TD.dat *circuit-file*.HB.FD.dat	TIME FREQUENCY
`.options hbint STARTUPPERIODS=n`	Falls back to .PRINT HB variables	
`.PRINT HB_STARTUP` `.options hbint STARTUPPERIODS=n`	*circuit-file*.startup.prn	INDEX TIME
`.PRINT HB_STARTUP FORMAT=CSV` `.options hbint STARTUPPERIODS=n`	*circuit-file*.startup.csv	TIME
`.PRINT HB_STARTUP FORMAT=TECPLOT` `.options hbint STARTUPPERIODS=n`	*circuit-file*.startup.dat	TIME
`.options hbint SAVEICDATA=1`	Falls back to .PRINT HB variables	
`.PRINT HB_IC` `.options hbint SAVEICDATA=1`	*circuit-file*.hb_ic.prn	INDEX TIME
`.PRINT HB_IC FORMAT=CSV` `.options hbint SAVEICDATA=1`	*circuit-file*.hb_ic.csv	TIME
`.PRINT HB_IC FORMAT=TECPLOT` `.options hbint SAVEICDATA=1`	*circuit-file*.hb_ic.dat	TIME
`.OP`	*log-file*	Operating point information

These examples include some types of sweep (linear, decade and data). The Xyce Reference Guide [3] provides a complete description of all types of sweep. Note that `DATA` is a special case, which will allow sweeping of frequency, magnitude, phase and linear model parameters simultaneously.

7.8.2. AC Voltage and Current Sources

Xyce assumes the AC source to be a cosine waveform at a specified phase angle. Its frequency must be defined in a separate ".AC" command defining the frequency for all the sources in the circuit. The unique information for the individual source is the name (which must start with "V" or "I"), the node numbers, the magnitude of the source, and its phase angle. Some examples are as follows:

Example:

```
Vac 4 1 AC 120V 30
Vin 1 0 1.44 ac .1
Iin 1 0 1.44e-5 ac 0.1e-5 sin(0 1 1e+5 0 0)
```

NOTE: The type, AC, must be specified because the default is DC. If not specified, Xyce assumes the phase angle to be zero degrees. The units of the phase angle are in degrees.

7.8.3. Example

An example AC analysis netlist is given in figure 7-20. This particular example uses a `.data` statement to specify a more complicated sweep. Each line of the data table corresponds to a different sweep point, so there are 3 points evaluated, and there is a different magnitude, phase, frequency and resistance for each point.

7.8.3.1. Linear Solver Options

The AC analysis provided by Xyce can employ both iterative and direct solvers. You can set direct, iterative solver, and preconditioner options using the `.options linsol-ac` statement.

Example:

```
.options linsol-ac type=ksparse
```

In this example, `type` specifies a non-default direct solver to use in the AC analysis. Any of the direct and iterative solver options in section 11.4 are valid.

7.8.4. Output

During analysis a number of output files may be generated. The selection of which files are created depends on a variety of factors, most obvious of which is the `.PRINT` command. Table 7-6 lists the format options and files created. The column labeled "Additional Columns" lists the additional data that is written, though not specified on the `.PRINT` line.

Running Xyce on AC analysis produces an output results file named `.cir.FD.prn`. Obtaining this file requires that the `.PRINT AC` line be specified.

```
* AC Analysis example
.param mag=1
.param phase=0.1
.param cmplxParam1={log(-1)}  ; example complex parameter
.param cmplxParam2={2.0+3.0J} ; example complex parameter

Isrc 1 0 AC {mag} {phase}
R1 1 0 1e3
C1 1 0 2e-6

.data table
+  mag  phase  freq   r1
+  1.0   0.1  1.0e0  1e3
+  2.0   0.2  1.0e1  2e3
+  3.0   0.3  1.0e2  3e3
.enddata

* AC commands
.AC data=table

* AC file output
.print ac {mag} {phase} {Isrc:acmag} {Isrc:acphase}
+ {r1:r} v(1)
+ {cmplxParam1 * V(1)}
+ {cmplxParam2 * V(1)}

.end
```

Figure 7-20. AC Example Netlist. This example uses a .data statement to simultaneously set frequency, magnitude, phase and the resistance of r1.

Table 7-6. Output generated for AC analysis

Command	Files	Additional Columns
`.PRINT AC`	*circuit-file*.FD.prn	INDEX FREQUENCY
`.PRINT AC FORMAT=NOINDEX`	*circuit-file*.FD.prn	INDEX FREQUENCY
`.PRINT AC FORMAT=CSV`	*circuit-file*.FD.csv	FREQUENCY
`.PRINT AC FORMAT=RAW`	*circuit-file*.raw	FREQUENCY
`.PRINT AC FORMAT=TECPLOT`	*circuit-file*.FD.dat	TIME
`.PRINT AC FORMAT=PROBE`	*circuit-file*.csd	TIME
Xyce `-r raw-file-name`	*raw-file-name*	All circuit variables printed
Xyce `-r raw-file-name -a`	*raw-file-name*	All circuit variables printed
`.OP`	falls back to .PRINT AC	Operating point information
`.PRINT AC_IC FORMAT=NOINDEX`	*circuit-file*.TD.prn	TIME
`.PRINT AC_IC FORMAT=CSV`	*circuit-file*.TD.csv	TIME
`.PRINT AC_IC FORMAT=RAW`	*circuit-file*.raw	TIME
`.PRINT AC_IC FORMAT=TECPLOT`	*circuit-file*.TD.dat	TIME
`.PRINT AC_IC FORMAT=PROBE`	*circuit-file*.TD.csd	TIME
`.OPTIONS NONLIN CONTINUATION=<method>...`	falls back to AC	
`.PRINT HOMOTOPY FORMAT=NOINDEX`	*circuit-file*.HOMOTOPY.prn	INDEX TIME

Xyce supports printing the real and imaginary parts of phasor values (complex numbers) for AC analysis output as voltages or currents. For instance, specify "V(1)" to print the real part and imaginary part of a voltage at node 1. Additional output variable formats are also available:

- `VR(<circuit node>)` output the real component of the voltage response
- `VI(<circuit node>)` output the imaginary component of voltage response
- `VM(<circuit node>)` output the magnitude of voltage response
- `VDB(<circuit node>)` output the magnitude of voltage response in decibels
- `VP(<circuit node>)` output the phase of voltage response, in degrees

Also, Xyce supports printing the real and imaginary parts of complex expressions. The user can specify the real or imaginary parts of an expression using the `Re()` and `Img()` operators within the expression. However, it usually isn't necessary to do this, as Xyce will automatically output both the real and imaginary parts, if the expression produces a complex result. If an expression produces a purely real result (even if the inputs are complex), then Xyce will only output the real part.

Some examples are as follows:

Example:

```
.print AC V(3)
.print AC VI(15)
.print AC VP(OUTPUT)
.print AC {Re(V(OUTPUT))}
.print AC {Img(V(OUTPUT))}
.print AC { V(OUTPUT)*2.0 }
```

As shown in the examples, the expression library can handle both real and complex valued solution variables. See the Xyce Reference Guide [3] for more details.

7.9. NOISE Analysis

Xyce supports small-signal noise analysis, which is closely related to AC analysis. For noise analysis, input and output noise is computed for a specified output node, relative to an input source, as a function of frequency.

The program first computes the DC operating point of the circuit and linearizes the circuit. Next, at each frequency, an AC calculation is performed to obtain the AC gain. This gain is used later to compute the equivalent input noise from the output noise. After the AC gain is computed, the noise calculation is performed at the current frequency using the adjoint method. This involves solving the transpose of the AC matrix.

If the circuit has one NOISE input, it is convenient to set that input to unity and zero phase so output variables have the same value as the transfer function of the output variable with respect to the input.

7.9.1. .NOISE Statement

One may specify .NOISE analyses by adding a .NOISE line in the netlist. Some examples of typical .NOISE lines include:

Example:

```
.NOISE V(5) VIN LIN 101 100Hz 200Hz
.NOISE V(5,3) V1 OCT 10 1kHz 16kHz
.NOISE V(4) V2 DEC 20 1MEG 100MEG
.NOISE V(4) V2 DATA=table
```

The .NOISE command can specify a linear sweep, decade logarithmic sweep, octave logarithmic sweep, or a data table of multivariate values. See the Xyce Reference Guide [3] for details.

7.9.2. NOISE Voltage and Current Sources

Noise analysis requires a reference AC source. Xyce assumes the AC source to be a cosine waveform at a specified phase angle. Its frequency must be defined in a separate ".NOISE" command defining the frequency for all the sources in the circuit. The unique information for the individual source is the name (which must start with "V" or "I"), the node numbers, the magnitude of the source, and its phase angle. Some examples are as follows:

Example:

```
Vac 4 1 AC 120V 30
Vin 1 0 1.44 ac .1
Iin 1 0 1.44e-5 ac 0.1e-5 sin(0 1 1e+5 0 0)
```

NOTE: The type, AC, must be specified because the default is DC. If not specified, Xyce assumes the phase angle to be zero degrees. The units of the phase angle are in degrees.

7.9.3. Examples

Example noise analysis netlists are given in figures 7-21 and 7-22. In those two examples, the only noise source is the resistor, rlp1. The input source is v1 and the output node is v(4). A decade sweep is used in the first example, while a .DATA statement is used in the second example.

7.9.4. Output

During analysis a number of output files may be generated. The selection of which files are created depends on a variety of factors, most obvious of which is the .PRINT command. Table 7-7 lists the format options and files created. The column labeled "Additional Columns" lists the additional data that is written, though not specified on the .PRINT line.

Running Xyce NOISE analysis produces an output results file named .cir.NOISE.prn. Obtaining this file requires that the .PRINT NOISE line be specified. Using the .PRINT line, Xyce supports printing

```
* Noise Analysis example
v1  1 0 DC 5.0 AC  1.0
r1  1 2 100K
r2  2 0 100K

eamp  3 0 2 0 1
rlp1  3 4 100
clp1  4 0 1.59nf

* Noise commands
.noise v(4) v1 dec 5 100 100meg 1

* Noise file output
.print noise inoise onoise

.end
```

Figure 7-21. Noise Example Netlist With Decade Sweep

```
* Noise Analysis example
.global_param mag=1
.global_param phase=0.1

v1  1 0 DC 5.0 AC {mag} {phase}
r1  1 2 100K
r2  2 0 100K

eamp  3 0 2 0 1
rlp1  3 4 100
clp1  4 0 1.59nf

.data table
+  mag  phase  freq  rlp1
+  1.0   0.1  1.0e6  1e2
+  2.0   0.2  1.0e7  2e2
+  3.0   0.3  1.0e8  3e2

* Noise commands
.noise v(4) v1 data=table

* Noise file output
.print noise inoise onoise

.end
```

Figure 7-22. Noise Example Netlist. This example uses a .data statement to simultaneously set frequency, magnitude, phase and the resistance of rlp1.

Table 7-7. Output generated for NOISE analysis

Command	Files	Additional Columns
`.PRINT NOISE`	*circuit-file*.NOISE.prn	INDEX FREQUENCY
`.PRINT NOISE FORMAT=NOINDEX`	*circuit-file*.NOISE.prn	FREQUENCY
`.PRINT NOISE FORMAT=CSV`	*circuit-file*.NOISE.csv	FREQUENCY
`.PRINT NOISE FORMAT=TECPLOT`	*circuit-file*.NOISE.dat	FREQUENCY

the total noise spectral density curves. The output units for spectral densities are V^2/Hz and A^2/Hz. Some examples are as follows:

Example:
```
.print noise inoise onoise
.print noise log(inoise) log(onoise)
.print noise inoise onoise V(4) VR(1)
```

Xyce will also compute the total integrated output noise, and the total integrated input noise for the specified output node. This information is sent to standard output, or to the user-specified log file. A typical output (which in this case was generated by the netlist given in figure 7-21) will look like this:

Example:
```
Total Output Noise = 1.28592e-09
Total Input Noise = 5.22876e-07
```

If `.STEP` is used with a noise analysis then the Xyce standard output will contain the values of the stepped parameters, along with the values for the total integrated output noise and the total integrated input noise for the specified output node.

Nodal voltages may be requested with a `.PRINT NOISE` line, and the values are the AC solution for the circuit. All of the voltage operators in Table 9-4 are supported for this case.

The DC values of voltage sources (e.g., `V1`) that are changed with a `.STEP` line during a NOISE analysis can be printed out via the following syntax. Alternately, if a `.OP` line is included in the netlist then the output from the `-r` command line option will contain the DC Operating Point (DCOP) values for all of the nodal voltages. (Note: Without the `.OP` line, only the AC solution will be included in the `-r` output for a NOISE analysis.)

Example:
```
.print noise V1:DCV0
```

7.9.5. Device Model Support

Note that stationary noise is not supported in every Xyce device. A list of Xyce devices, and which of them support stationary noise, is given in the Xyce Reference Guide [3].

The DNI and DNO operators can print out the individual input and output noise contributions for each noise source within a device via .PRINT NOISE lines. Figure 7-23 provides an example.

The -noise_names_file command line option can generate a listing of the noise source names for each device in a netlist. The output of that command for the netlist given in figure 7-23 is shown in figure 7-24. In this example, DNI(M1) would output the input noise contribution for all noise sources in MOSFET M1, while DNI(Q1,FN) would only output the input flicker-noise contribution from MOSFET M1.

```
* Noise Analysis: DNI and DNO Usage
* A Simple MOSFET Gain Stage.

M1 3 2 0 0 nmos w=4u l=1u
.model nmos nmos level=1 tox=1e-7

Rsource 1 2 100k
Rload 3 vdd 25k

Vdd1 vdd 0 5
Vin 1 0 1.44 ac .1 sin(0 1 1e+5 0 0)

* Noise commands
.noise v(3) Vin dec 10 100 1000Meg  1

* Noise file output
.print noise inoise onoise
+ DNI(rsource) DNI(rload) DNI(M1)
+ DNI(M1,RD) DNI(M1,RS) DNI(M1,ID) DNI(M1,FN)
+ DNO(rsource) DNO(rload) DNO(M1)
+ DNO(M1,RD) DNO(M1,RS) DNO(M1,ID) DNO(M1,FN)

.end
```

Figure 7-23. Noise Example: Use of DNI and DNO Operators

```
Valid DNO() and DNI() operators for this netlist.
Format is DNO(deviceName) for the total output noise for
a device or DNO(deviceName, noiseSource) for an individual
output noise contribution.  If a device (e.g., R1) only
has one noise source then use DNO(R1) to get the total
output noise.

deviceName       noiseType
RLOAD
M1
M1      fn
M1      id
M1      rd
M1      rs
RSOURCE
```

Figure 7-24. Example Output for -noise_names_file Command Line Option

7.10. Sensitivity Analysis

The .SENS command instructs Xyce to calculate the sensitivities of an output expression with respect to a specified list of circuit parameters. This capability works for steady-state (.DC), transient (.TRAN) and small-signal (.AC) analysis.

General Format:

```
.SENS objfunc={<output expression(s)>} param=<circuit parameter(s)>
.options SENSITIVITY [direct=<1 or 0>] [adjoint=<1 or 0>]
```

At least one objective function parameter, objfunc, is required. It is possible to specify a comma-separated list of objective functions. The list of circuit parameters must have at least one entry as well. If there is more than one, the parameters are specified as a comma-separated list. Xyce will not assume any particular parameter list. Unlike the Spice version of .SENS, Xyce will not automatically compute the sensitivity of every parameter; only the parameters the user requested.

7.10.1. Steady-State (DC) sensitivities

For DC analysis, Xyce can compute a direct sensitivity, an adjoint sensitivity, or both. In general, adjoint sensitivities are much more efficient when computing the derivatives with respect to really large numbers of parameters, but a small number of objective functions. Direct sensitivities are most efficient for small numbers of parameters, but large numbers of objective functions. Xyce allows the user to specify multiple objective functions, as well as multiple parameters, so either scenario could apply.

An example of using .SENS on a DC problem is shown in the following netlist: The output to stdout,

```
Example Circuit using sensitivity analysis
R1 A B 10.0
R2 B 0 10.0
Va A 0 5

.dc Va 5 5 1
.print dc v(A) v(B)

.print sens
.SENS objfunc={0.5*(V(B)-3.0)**2.0} param=R1:R,R2:R
.options SENSITIVITY direct=1 adjoint=1 stdoutput=1
.END
```

Figure 7-25. Steady-State Sensitivity Example Netlist

which was requested by setting stdoutput=1, for this example is:

```
Direct Sensitivities of objective function:{0.5*(V(B)-3.0)**2.0}
Name         Value    Sensitivity     Normalized
R1:R    1.0000e+01     6.2500e-02     6.2500e-03
R2:R    1.0000e+01    -6.2500e-02    -6.2500e-03
```

```
Adjoint Sensitivities of objective function:{0.5*(V(B)-3.0)**2.0}
Name          Value    Sensitivity    Normalized
R1:R     1.0000e+01     6.2500e-02     6.2500e-03
R2:R     1.0000e+01    -6.2500e-02    -6.2500e-03
```

In the above example, both adjoint and direct sensitivities are computed. This is a linear problem, so it is easy to compare them to the analytic solution.

7.10.2. Transient sensitivities

Sensitivities can also be computed for transient analysis. As with steady-state (DC) analysis, both direct and adjoint sensitivities are supported.

7.10.2.1. Transient Direct Sensitivities

Transient direct sensitivities are a good choice if the number of parameters is modest. Also, if the output of interest is a full waveform sensitivity, they will usually outperform adjoint sensitivities, as from the perspective of adjoint calculations, each time point constitutes a separate objective function, requiring a separate integration.

Set `direct=1 adjoint=0` to specify transient direct simulations. The transient direct forumulation in Xyce is based on the one described by Hocevar [25]. An example netlist for a transient sensitivity calculation is given in figure 7-26. This is a simple linear problem (RC decay), so it has an analytic solution. Results for the transient example are given in figure 7-27. The analytic sensitivity solution is

```
* Transient sensitivity example
.param cap=1u
.param res=1K
c1 1 0 cap
R1 1 0 res
.IC V(1)=1.0
* Transient commands
.tran 0 5ms uic
.options timeint reltol=1e-6 abstol=1e-6

* Conventional file output
.print tran format=tecplot v(1)
* Sensitivity file output.
.print sens format=tecplot
+  exp(-time/(res*cap)) ; analytic solution (V(1))
+ {(time/(res*res*cap))*exp(-time/(res*cap))}; analytic dV(1)/dR
+ {(time/(res*cap*cap))*exp(-time/(res*cap))}; analytic dV(1)/dC
* Sensitivity commands
.SENS objfunc={V(1)} param=R1:R,C1:C
.options SENSITIVITY direct=1 adjoint=0
.end
```

Figure 7-26. Direct Transient Sensitivity Example Netlist

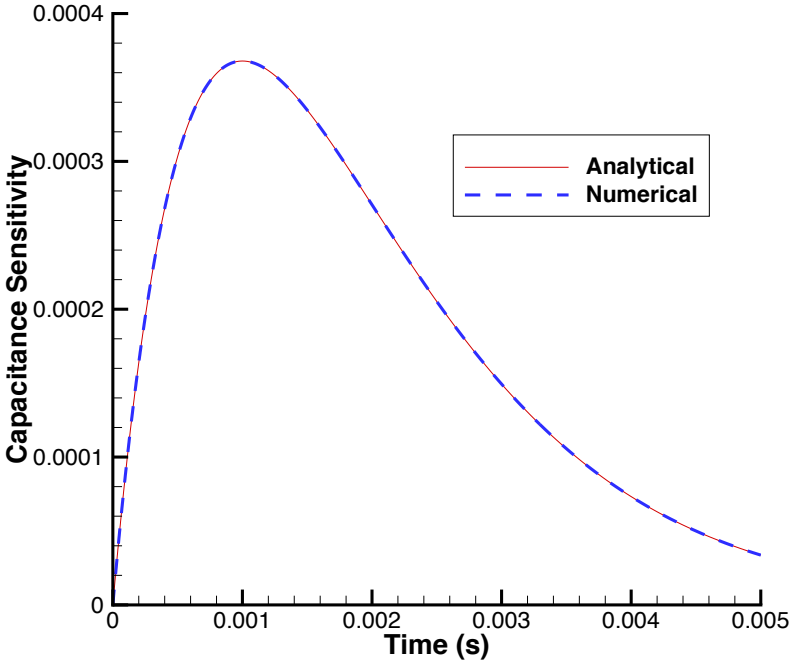

Figure 7-27. Transient direct sensitivity result.

given by the solid line and the computed numerical sensitivity is given by the dashed line. The results match very well in this case, with the two lines right on top of each other.

7.10.2.2. Transient Adjoint Sensitivities

Transient adjoint sensitivities are a good choice for really large numbers of parameters, and when the number of objective functions is modest. For transient calculations, each time point is considered a separate objective function, so it is best to use adjoints when the sensitivity of interest concerns only one or a handful of time points.

Set `direct=0 adjoint=1` to specify transient adjoint simulations. The transient adjoint forumulation in Xyce has similarities to the ones described by Liu [26] and Meir [27]. An example netlist for a transient adjoint sensitivity calculation is given in figure 7-28. This is a simple linear problem (RC driven by a sinewave), so it has an analytic solution. Results for the transient adjoint example are given in figure 7-29.

```
*Test of transient adjoint sensitivities
.param cap=1e-6
.param res=1e3

V1 1 0 0.0 sin (0.0 1.0 200 0.0 0.0 0.0 )
R1 1 2 res
C1 2 0 cap

.tran 1.0e-6 0.5e-2
.print tran format=tecplot V(1) V(2)
.print TRANADJOINT format=tecplot

.options timeint method=gear reltol=1.0e-6 abstol=1e-6

* Sensitivity commands
.SENS objfunc=V(2) param=R1:R,C1:C
.options SENSITIVITY direct=0 adjoint=1
.end
```

Figure 7-28. Adjoint Transient Sensitivity Example Netlist

The analytic sensitivity solution is given by the dashed line and the computed numerical sensitivity is given by the solid line. The results match very well in this case, with the two lines right on top of each other.

Guidance for running transient adjoint analysis Transient adjoint sensitivities need to perform a backward time integration, after the original transient forward problem has completed. So, generally, when running a transient adjoint netlist, the user may notice a pause in the calculation at the very end. This is normal.

It is also important to understand that if using a point objective, the transient adjoint algorithm will consider every time step to be a unique, separate objective, and will need to perform a backward integration for each one. So, it will usually be important to restrict this to a limited number of time steps, or ideally a single time point.

To perform the adjoint calculation on a range of times, this can be set using the `.OPTIONS SENSITIVITY` commands `ADJOINTBEGINTIME` and `ADJOINTFINALTIME`. Each of these must be set equal to a floating point number in units of seconds. To perform the adjoint calculation on a discrete set

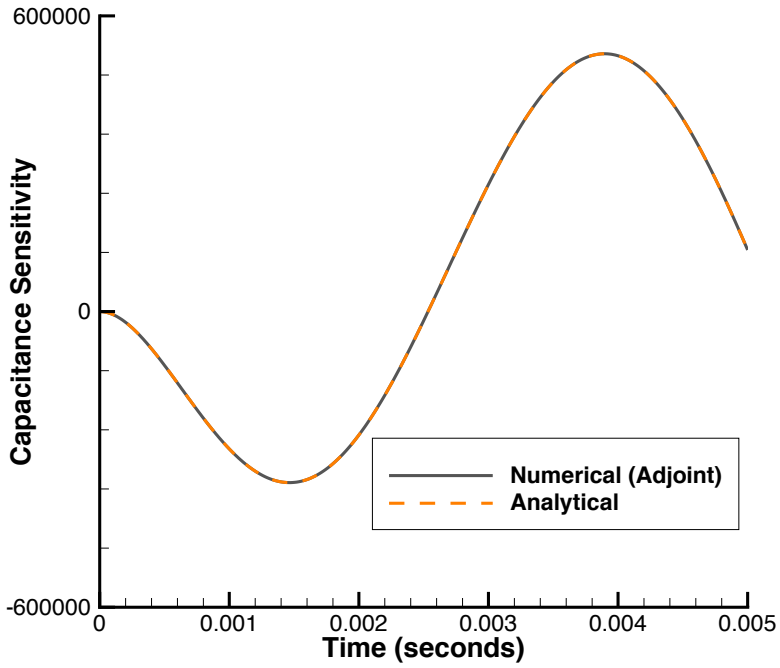

Figure 7-29. Transient adjoint sensitivity result.

of time points (or a single time point), this can be set using using the `.OPTIONS SENSITIVITY` command `ADJOINTTIMEPOINTS` followed by a comma-separated list of times in units of seconds. An example netlist, which uses this capability is shown in figure 7-30. It is identical to the netlist shown in figure 7-28 except that this command has been added.

```
*Test of transient adjoint sensitivities
.param cap=1e-6
.param res=1e3

V1 1 0 0.0 sin (0.0 1.0 200 0.0 0.0 0.0 )
R1 1 2 res
C1 2 0 cap

.tran 1.0e-6 0.5e-2
.print tran format=tecplot V(1) V(2)
.print TRANADJOINT format=tecplot

.options timeint method=gear reltol=1.0e-6 abstol=1e-6

* Sensitivity commands
.SENS objfunc=V(2) param=R1:R,C1:C
.options SENSITIVITY direct=0 adjoint=1
+ adjointTimePoints=0.1e-2, 0.2e-2, 0.3e-2, 0.4e-2
.end
```

Figure 7-30. Adjoint Transient Sensitivity Example Netlist for list of time points

7.10.3. AC Sensitivities

Sensitivities can also be computed for small-signal AC analysis. As with steady-state (DC) and transient analysis, both direct and adjoint sensitivities are supported.

General Format:

```
.SENS acobjfunc={<output expression(s)>}
objvars=<voltage node name(s)>
param=<circuit parameter(s)>
.options SENSITIVITY [direct=<1 or 0>] [adjoint=<1 or 0>]
```

Two example netlists which exercise the capability is shown in figures 7-31 and 7-33. The netlist commands for AC sensitivities are very similar to those for DC, but with some small exceptions.

Setting the AC sensitivity ojbective function For AC sensitivities, objective functions can be set in two ways. One option is to specify outputs using `objvars` followed by a list of node names. An example which uses this method is given in figure 7-31. The other option is to specify the objective function as an expression, with the `acobjfunc` parameter. An example which uses this method is given in figure 7-33. It is also possible to specify both types of objectives in the same netlist.

AC Sensitivities have four objectives per output For each specified output, there are four separate objective functions for which sensitivities will be computed; magnitude and phase for the polar representation, as well as real and imaginary for the cartesian representation of the solution. These will be computed with respect to the every parameter in the list of sensitivity parameters.

Example console output results are shown for the netlists in figures 7-31 and 7-33 in figures 7-32 and 7-34, respectively. The console output shown is what one gets if `stdoutput=1` is set in the netlist on the `.options sensitivity` line. For file-based output, see section 7.10.5.

Output of ΔP For AC sensitivities, it is likely that a numerical derivative will be needed as part of the calculation. This is because, unlike DC and transient sensitivities, the AC calculation often requires a matrix derivative. The Xyce source code has not been set up to compute analytical matrix derivatives, so these must always be computed numerically. While the numerical derivatives are accurate, some information about them has been added to the AC sensitivity console output.

In the console outputs shown in figures 7-32 and 7-34, the two right-most columns pertain to numerical finite differences. The “Delta P” column gives the delta used by the sensitivity calculation if a numerical derivative was needed. The “FD used” or “FD not used” column indicates if finite differences were used in part of the calculation.

```
*Lowpass filter test for AC sensitivities
v1 1 0 ac 10.0
r1 1 2 4.7k
c1 2 0 47n

.ac dec 10 1 10k
.print ac  vm(1) vp(1) vm(2) vp(2)
.print sens

* Sensitivity commands
.sens objvars=2 param=r1:r,c1:c
.options sensitivity direct=1 adjoint=0  stdoutput=1
.end
```

Figure 7-31. AC Sensitivity Example Netlist

```
Direct Sensitivities for node V(2):
VR(2) = 1.0000e+01  VI(2) = -1.3880e-02
VM(2) = 1.0000e+01  VP(2) = -1.3880e-03

Name        Value    Sens_Re    Sens_Im    Sens_Mag   Sens_Phase  Delta P
R1:R     4.7000e+03 -8.1975e-09 -2.9531e-06 -4.0988e-09 -2.9531e-07  7.0035e-05 FD not used
C1:C     4.7000e-08 -8.1975e+02 -2.9531e+05 -4.0988e+02 -2.9531e+04  7.0035e-16 FD not used
V1:ACMAG 1.0000e+01  1.0000e+00 -1.3880e-03  1.0000e+00 -2.7105e-20  1.4901e-07 FD not used
```

Figure 7-32. AC Sensitivity Example Result. This result corresponds to the first objective in the netlist given by figure 7-31.

```
* Operational Amplifier
.param m6w=2u
.param m6l=1u
*Operational Amplifier
M1 bias1 1 cm cm nmos w=10u l=1u
M2 bias2 in2 cm cm  nmos w=10u l=1u
M3 vdd bias1 bias1 vdd pmos w=2u l=1u
M4 bias2 bias1 vdd vdd pmos w=2u l=1u
m5 cm bias vss vss nmos w=2u l=1u
mbias bias bias vss vss nmos w=2u l=1u
rbias 0 bias 195k
m6 8 bias vss vss  nmos w={m6w} l={m6l}
m7 8 bias2 vdd out nmos w=2u l=1u
Cfb bias2 8 2p
Vid 1 c 0 ac 0.1
eid in2 c 1 c -1
vic c 0 dc 0
vss1 vss 0 -5
Vdd1 vdd 0 5

*AC analysis
.ac dec 10 100 100Meg
.print ac v(8)
* Sensitivity commands
.sens acobjfunc={v(8)} param=m6:w,m6:l
.options sensitivity direct=1 adjoint=0  stdoutput=1
.print sens

.model nmos nmos level=9 version=3.2.2
.model pmos pmos level=9 version=3.2.2
.end
```

Figure 7-33. AC Sensitivity Example Netlist, using expression-based objective function

```
Direct Sensitivities for {v(8)}
 Re({v(8)}) = 1.7365e+01  Img({v(8)}) = 3.5597e-04
  M({v(8)}) = 1.7365e+01   Ph({v(8)}) = 6.7295e-02
Name    Value      Sens_Re    Sens_Im    Sens_Mag   Sens_Phase  Delta P
M6:W  2.0000e-06 -2.2615e+05  -1.4915e+02 -2.2615e+05 -4.7680e+02  2.9802e-14 FD used
M6:L  1.0000e-06  8.6598e+05   1.9292e+02  8.6598e+05  5.7798e+02  1.4901e-14 FD used
```

Figure 7-34. AC Sensitivity Example Result. This result is the first sensitivity output produced for the first objective in the netlist given by figure 7-33.

7.10.4. Notes about .SENS accuracy and formulation

The formulation for Xyce sensitivity analysis is briefly described. For more details, see references [28, 29]. The sensitivity calculation in Xyce is based on a chain rule calculation, which will produce the sensitivity dO/dp, where O is a user-specified scalar objective function, and p is a user-specified scalar parameter. dO/dp is equal to:

$$\frac{dO}{dp} = \frac{\partial O}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial O}{\partial p} \tag{7.4}$$

where x is a solution vector and $\partial x/\partial p$ is the sensitivity of that solution vector with respect to the parameter. For DC and Transient, the system of equations solved by Xyce is given as a differential algebraic system, given by:

$$F(x,t,p) = \dot{q}(x(t,p),p) + j(x(t,p),p) - b(t,p) = 0, \tag{7.5}$$

where the vector F is the Xyce residual. The derivative of F with respect to the solution x ($\partial F/\partial x$) is the Jacobian matrix. The terms q, j and b vectors are all standard components of a DAE, and are provided by the various device models. q represents quantities that must be differentiated with respect to time (such as capacitor charge), and j represents algebraic terms that depend on the solution x (such as DC currents), and b are independent sources that only depend on time. Equation 7.5 is solved to obtain the solution, x.

Equation 7.4 can then be expanded as:

$$\frac{dO}{dp} = -\frac{\partial O}{\partial x}\left(\frac{\partial F}{\partial x}\right)^{-1}\frac{\partial F}{\partial p} + \frac{\partial O}{\partial p}, \tag{7.6}$$

All of the terms in equation 7.6 are available in Xyce, which makes it possible to compute dO/dp. Also, most terms in 7.6 purely analytical, which helps ensure accuracy. The order in which the right-hand-side of equation 7.6 is evaluated (left-to-right or right-to-left) determines if the method is a direct sensitivty or an adjoint sensitivity.

7.10.4.1. Direct Sensivity

To obtain direct sensitivities, one must differentiate equation 7.5 with respect to a parameter, p.

$$\frac{dF(x,t,p)}{dp} = \frac{d}{dp}\left[\frac{dq(x(t),p)}{dt} + j(x(t),p) - b(t,p)\right] = 0 \tag{7.7}$$

In steady-state (DC), equation 7.7 simplifies to:

$$\frac{dF(x,t,p)}{dp} = \frac{d}{dp}\left[j(x(t),p) - b(t,p)\right] = 0 \tag{7.8}$$

As j is dependent upon x, the dj/dp term be expanded via chain rule as $\frac{dj}{dp} = \frac{\partial j}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial j}{\partial p}$, where $\frac{\partial j}{\partial x}$ is the DC Jacobian matrix, and the resulting equation must be re-arranged to set up a matrix equation that can be solved to obtain $\frac{\partial x}{\partial p}$:

$$\frac{\partial j}{\partial x}\frac{\partial x}{\partial p} = -\frac{\partial j}{\partial p} + \frac{db}{dp}. \tag{7.9}$$

This linear equation is evaluated by the linear solver at the end of the steady-state solve, using the same LU factors as were used by the final Newton step.

For transient, a similar linear system is solved, which depends on the specific time integration method used. For Backward-Euler the linear system is:

$$\left[\frac{1}{h}\frac{\partial q}{\partial x}+\frac{\partial j}{\partial x}\right]\frac{\partial x}{\partial p}_{n+1}=-\frac{1}{h}\left[\frac{\partial q}{\partial p}_{n+1}-\frac{\partial q}{\partial p}_{n}\right]-\frac{\partial j}{\partial p}+\frac{db}{dp}+\frac{1}{h}\left[\frac{\partial q}{\partial x}\right]\frac{\partial x}{\partial p}_{n} \tag{7.10}$$

Where h is the time step size. As with 7.9, the left hand side of the equation contains the original Jacobian matrix. Similar to the steady-state case, this equation is solved at the end of each converged time step, using the matrix solver with the same LU factors as were used for the final Newton step. For both steady state and transient, the final objective sensitivity is obtained by taking the dot product of $\partial O/\partial x$ and $\partial x/\partial p$.

7.10.4.2. Adjoint Sensivity

The description of adjoint sensitivies is beyond the scope of this document, but the issues affecting accuracy are largely the same. For a description of the formulation, see references [28, 29].

7.10.4.3. Analytical vs Numerical derivatives

As mentioned above, most of the terms in the sensitivity equations are analytical. The possible exception is with the terms $\partial q/\partial p$, $\partial j/\partial p$ and $\partial b/\partial p$, which depend on device model implementation. Most devices in Xyce are instrumented to provide these sensitivities analytically, either with hand-written derivatives or via automatic differentiation. But there are exceptions to this, so some devices do not. For these devices, a numerical derivative is computed instead. A list of Xyce devices, and which of them support analytic sensitivities, is given in the Xyce Reference Guide [3]. When numerical derivatives are used, Xyce automatically selects a small delta just large enough to avoid machine precision, following the procedure suggested by Dennis and Schnabel [30].

7.10.4.4. Time integration error

For transient, note that the transient direct calculation uses the same time steps that are used for the original circuit analysis. It does not impose any additional error control that is specific to the accuracy of $\partial x/\partial p$. The adjoint calculation (not described here), is a discrete form of transient adjoints, and thus uses the same time steps as the forward calculation.

7.10.4.5. AC Sensitivities

While many of the accuracy issues pertaining to AC sensitivities are the same as for DC sensitivities, the formulation is different enough to introduce an additional issue. AC sensitivities will nearly always use some numerical derivatives in their evaluation. AC sensitivies are based on small-signal analysis, so the analysis solves for the response of a circuit to a small sinusoidal signal. It first solves the DC operating point, and then approximates the AC response using a Taylor series expansion. This exansion approximates a small change in the solution as a function of a small change in the AC stimulus. The original AC equation is given by:

$$J\cdot\Delta x=\Delta b \tag{7.11}$$

where J is the AC Jacobian matrix, Δx is the AC change to the solution vector and Δb is the AC change to the source vector. Equation 7.11 assumes a phasor representation, so $\Delta x = Xe^{i\omega t\pm\phi}$ and $\Delta b = Be^{i\omega t\pm\phi}$,

where ω is the frequency and ϕ is the phase. The AC Jacobian is determined from the DC Jacobian matrices for f and q as $J = (\partial f/\partial x + i\omega\partial q/\partial x)$.

For AC sensitivities, we want to compute $\Delta x' = \partial(\Delta x)/\partial p$, where p is the sensitivity parameter. To obtain this, one takes derivative of equation 7.11 with respect to a parameter, p. This gives the direct sensitivity equation

$$J \cdot \Delta x' = \Delta b' - J' \cdot \Delta x, \tag{7.12}$$

where the apostrophe indicates a parameter derivative. The linear system in equation 7.12 must be solved to obtain $\Delta x'$, the sensitivity of Δx with respect to a parameter. This is the equation for the direct method, but a related equation can be obtained for the adjoint method. To solve equation 7.12, the terms on the right hand side must be provided by Xyce. The Δx term is the AC solution, which must be computed prior to sensitivity analysis. The term $\Delta b'$ is trivial to provide analytically, as it is generally from ideal independent source models. The matrix derivative J', however, is more complicated so it usually contains numerical derivatives. To see why, consider that when J' is fully expanded it is given as:

$$J' = \frac{\partial J}{\partial p} + \frac{\partial J}{\partial x} \cdot \frac{\partial x}{\partial p}. \tag{7.13}$$

The right hand side terms $\partial J/\partial p$ and $\partial J/\partial x$ of 7.13 are matrix derivatives, and Xyce is not instrumented to provide these using the usual method of automatic differentiation. For complex device models, implementing matrix derivatives manually is prohibitive. Therefore, with the exception of simple linear devices, the terms in equation 7.13 are evaluated numerically.

7.10.5. Output

For a full list and explanation of options related to `.SENS` output see the Xyce Reference Guide [3].

Sensitivity output can be sent to standard output, or to a user-specified log file. The format for this output is similar to that generated by the circuit given in figure 7-25. This feature is mainly made available in Xyce so as to be similar to the sensitivity analysis of older circuit simulators. However, for most uses it isn't the most practical output, so it is disabled by default. To enable standard output, one should set the following:

```
.options SENSITIVITY STDOUTPUT=1
```

In addition to the screen output, Xyce can also produce a plottable file containing all the requested sensitivities. This file can be requested by adding either a `.PRINT SENS` or a `.PRINT TRANADJOINT` command to the input file. Steady-state sensitivities (adjoint or direct) and transient direct sensitivities will be handled by the `.PRINT SENS` command. Transient adjoint, on the other hand, is handled by the `.PRINT TRANADJOINT` command.

Unlike the traditional `.PRINT` line, both `.PRINT SENS` and `.PRINT TRANADJOINT` will assume that the user wants all the sensitivities specified on the `.SENS` line. As such it is not necessary (or possible) to specify specific sensitivities on the `.PRINT SENS` line. If the line exists, that is sufficient to produce the file, and it will contain a column for every objective function and every derivative. The file name is the same as the one produced with `.PRINT`, but with "`.SENS`" included just before the suffix. Similarly, for transient adjoint, the output file name has the string "`.TRADJ`" included before the suffix. Note also, that most of the same output formats (`std`, `tecplot`, etc.) are available for `.PRINT SENS` and `.PRINT TRANADJOINT` as they are for conventional `.PRINT`. The available formats are listed in table 7-8, 7-9and 7-10. As noted, transient adjoints are specified separately from `.PRINT SENS`. This is

Table 7-8. Output generated for SENS analysis for .TRAN

Command	Files	Additional Columns
`.PRINT SENS`	*circuit-file*.SENS.prn	INDEX TIME
`.PRINT SENS FORMAT=GNUPLOT`	*circuit-file*.SENS.prn	INDEX TIME
`.PRINT SENS FORMAT=SPLOT`	*circuit-file*.SENS.prn	INDEX TIME
`.PRINT SENS FORMAT=NOINDEX`	*circuit-file*.SENS.prn	INDEX TIME
`.PRINT SENS FORMAT=CSV`	*circuit-file*.SENS.csv	TIME
`.PRINT SENS FORMAT=TECPLOT`	*circuit-file*.SENS.dat	TIME

because the transient adjoint calculation is performed as a post-process, after the original forward calculation has been completed, and Xyce's forward outputters are no longer active. To specify transient adjoint output, one must use `.PRINT TRANADJOINT` instead. As it is possible to perform both a transient direct and a transient adjoint calculation as part of the same computation, and most of Xyce's output files are in column format, there wasn't an easy way to have them use the same outputter.

Table 7-9. Output generated for SENS analysis for .AC

Command	Files	Additional Columns
`.PRINT SENS`	*circuit-file*.FD.SENS.prn	INDEX FREQ
`.PRINT SENS FORMAT=GNUPLOT`	*circuit-file*.FD.SENS.prn	INDEX FREQ
`.PRINT SENS FORMAT=SPLOT`	*circuit-file*.FD.SENS.prn	INDEX FREQ
`.PRINT SENS FORMAT=NOINDEX`	*circuit-file*.FD.SENS.prn	INDEX FREQ
`.PRINT SENS FORMAT=CSV`	*circuit-file*.FD.SENS.csv	FREQ
`.PRINT SENS FORMAT=TECPLOT`	*circuit-file*.FD.SENS.dat	FREQ

Table 7-10. Output generated for transient adjoint SENS analysis

Command	Files	Additional Columns
`.PRINT TRANADJOINT`	*circuit-file*.TRADJ.prn	INDEX TIME
`.PRINT TRANADJOINT FORMAT=NOINDEX`	*circuit-file*.TRADJ.prn	INDEX TIME
`.PRINT TRANADJOINT FORMAT=CSV`	*circuit-file*.TRADJ.csv	TIME
`.PRINT TRANADJOINT FORMAT=TECPLOT`	*circuit-file*.TRADJ.dat	TIME

7.11. S-parameter Analysis

The S-parameter small-signal analysis of Xyce computes linear transfer parameters as a function of frequency for a general multi-port network. The program first computes the DC operating point of the circuit and then linearizes the circuit. The resultant linear circuit is then analyzed over a user-specified range of frequencies. The output of a S-parameter analysis is multi-port scattering (S) parameters. The S-parameters represent the ratio of incident and scattered normalized voltage waves. The analysis results can also be output as Y-parameter or Z-parameter values.

7.11.1. .LIN Statement

One may specify S-parameter analyses by using a `.LIN` command with a `.AC` command in the netlist. Here is an example of typical `.AC` and `.LIN` lines:

Example:

```
.AC DEC 10 1K 100MEG
.LIN sparcalc=1 format=touchstone
```

The `.LIN` analysis is similar to a basic small signal `.AC` analysis, but it also calculates small signal transfer parameters between terminals identified using port (P) devices. The Xyce Reference Guide [3] provides a complete description of `.LIN` analysis and the P device. To output Y-parameter or Z-parameter values instead the `LINTYPE=Y` or `LINTYPE=Z` argument can be used on the `.LIN` line.

7.11.2. Port Devices

The `.LIN` analysis computes the S-parameters based on the location of the port (P) devices and the values of their reference impedances. The port devices identify the ports used in `.LIN` analysis. The Xyce Reference Guide [3] provides a complete description of the port devices. Some examples are as follows:

Example:

```
P1 1 0 port = 1
P2 12 0 port= 2 z0=100
```

NOTE: Each port requires a unique port number. If a circuit has N ports, the netlist must contain the sequential set of port numbers, 1 to N.

7.11.3. Example

An example S-parameter analysis netlist is given in figure 7-35. This example uses 2 ports. The S parameters are output to a file in Touchstone 1 format [31]. Touchstone 2 format is also supported. Note that `SPARCALC=1` is specified on the `.LIN` line. If `SPARCALC=0` is specified instead then a `.AC` analysis will be done.

```
* S-parameter Analysis example
P1 1 0 port= 1
~
C1 2 0 1e-2
Rgs 1 2 0.02

.subckt RCBlock IN OUT GND
R1 IN OUT 20
C1 IN OUT 1p
Cg1 OUT GND 1p
.ends

X1 2 3 0 RCBlock
X2 3 4 0 RCBlock
X3 4 5 0 RCBlock
X4 5 6 0 RCBlock
X5 6 7 0 RCBlock
X6 7 8 0 RCBlock
X7 8 9 0 RCBlock
X8 9 10 0 RCBlock
X9 10 11 0 RCBlock
X10 11 12 0 RCBlock

.AC DEC 10 10  1e5

.LIN FORMAT=TOUCHSTONE  sparcalc=1

P2 12 0  port=2 z0=100

.END
```

Figure 7-35. S-parameter Example Netlist.

7.11.4. Output

For S-parameter analysis, the output is controlled by the .LIN command when SPARCALC=1 on that line. Any information on .PRINT AC lines will be ignored in that case. Table 7-11 lists the format options and files created. If SPARCALC=0 then a .AC analysis is done instead.

All three data formats defined in the Touchstone standard, which are real-imaginary (RI), magnitude-angle(MA) and magnitude(db)-angle (DB), are supported. The default is RI. Other options can be selected via the inclusion of the DATAFORMAT=<val> argument on the .LIN line. Per the Touchstone standard, all angle values are output in degrees.

The default filename for both Touchstone formats is <netlistName>.sNp where N is the number of "ports" (P devices) specified in the netlist. The output can be redirected to another file with the -o command line option or by using a FILE=<filename> argument on the .LIN line.

The default output is S-parameters. The LINTYPE=<S|Y|Z> argument can be used on the .LIN line to select Y-parameter or Z-parameter output instead. (Note: The default filename is still <netlistName>.sNp even if the output file contains Y-parameter or Z-parameter output rather than S-parameter output.) The Xyce Reference Guide [3] has a complete listing of which arguments, typically used on a .PRINT line also work on a .LIN line.

When a .LIN analysis is done then additional output variable formats are available via the .PRINT AC line, where <index1> and <index2> must both be greater than 0 and also both less than or equal to the number of ports in the netlist:

- SR(<index1>,<index2>) to output the real component of an S-parameter
- SI(<index1>,<index2>) to output the imaginary component of an S-parameter
- SM(<index1>,<index2>) to output the magnitude of an S-parameter
- SP(<index1>,<index2>) to output the phase of an S-parameter in degrees
- SDB(<index1>,<index2>) to output the magnitude of an S-parameter in decibels.
- YR(<index1>,<index2>) to output the real component of a Y-parameter
- YI(<index1>,<index2>) to output the imaginary component of a Y-parameter
- YM(<index1>,<index2>) to output the magnitude of a Y-parameter
- YP(<index1>,<index2>) to output the phase of a Y-parameter in degrees
- YDB(<index1>,<index2>) to output the magnitude of a Y-parameter in decibels.
- ZR(<index1>,<index2>) to output the real component of a Z-parameter
- ZI(<index1>,<index2>) to output the imaginary component of a Z-parameter
- ZM(<index1>,<index2>) to output the magnitude of a Z-parameter
- ZP(<index1>,<index2>) to output the phase of a Z-parameter in degrees
- ZDB(<index1>,<index2>) to output the magnitude of a Z-parameter in decibels.

Example:

```
.print AC SR(1,1) YI(1,2) ZM(2,1)
```

Table 7-11. Output generated for .LIN analysis

Command	Files	Format
`.LIN`	*circuit-file*.sNp	S-parameter data in Touchstone 2 format. Data format is RI.
`.LIN FORMAT=TOUCHSTONE`	*circuit-file*.sNp	S-parameter data in Touchstone 1 format. Data format is RI.
`.LIN FORMAT=TOUCHSTONE2`	*circuit-file*.sNp	S-parameter data in Touchstone 2 format. Data format is RI.
`.LIN DATAFORMAT=MA`	*circuit-file*.sNp	S-parameter data in Touchstone 2 format. Data format is MA.
`.LIN DATAFORMAT=DB`	*circuit-file*.sNp	S-parameter data in Touchstone 2 format. Data format is DB.
`.LIN LINTYPE=Y`	*circuit-file*.sNp	Y-parameter data in Touchstone 2 format. Data format is RI.
`.LIN LINTYPE=Z`	*circuit-file*.sNp	Z-parameter data in Touchstone 2 format. Data format is RI.
Xyce `-r raw-file-name`	NA	This will produce a parsing error.
Xyce `-r raw-file-name -a`	NA	This will produce a parsing error.

Chapter Overview

This chapter includes the following sections:

- Section 8.1, *Continuation Algorithms Overview*
- Section 8.2, *Natural Parameter Continuation*
- Section 8.3, *Natural Multiparameter Continuation*
- Section 8.4, *GMIN Stepping Continuation*
- Section 8.5, *Source Stepping Continuation*
- Section 8.6, *MOSFET Continuation*
- Section 8.7, *Pseudo Transient*
- Section 8.8, *Arc Length Continuation*

8.1. Continuation Algorithms Overview

Often, circuit convergence problems are most prominent during the DC operating point calculation. Unlike transient solves, DC operating point analysis cannot rely on a good initial guess from a previous step, and cannot simply reduce the step size when the solver fails. Additionally, operating points often have multiple solutions, with no capability to interpret the user's intent. Multiple solutions can, even for converged circuit problems, result in a standard Newton solve being unreliable. For example, the operating point solution to a Schmidt trigger circuit has been observed to change with the computational platform.

Continuation methods can often provide solutions to difficult nonlinear problems, including circuit analysis, even when conventional methods (e.g., Newton's method) fail [32] [33]. This chapter gives an introduction to using continuation algorithms (sometimes called homotopy algorithms) in Xyce. The Xyce Reference Guide [3] provides a more complete description of solver options.

The underlying numerical library used by Xyce to support continuation methods is LOCA (Library of Continuation Algorithms) [34, 35]. For a description of the numerical details see the LOCA theory and implementation manual [36].

8.1.1. Continuation Algorithms Available in Xyce

Xyce invokes a well-known SPICE method, "GMIN stepping," automatically when a DC operating point fails to converge without it. It is a special case where the parameter being swept is artificial. GMIN stepping can also be invoked with `.options nonlin continuation=3` or `.options nonlin continuation=gmin`. See Section 8.4 for more information and an example of GMIN stepping.

Xyce also invokes the SPICE "source stepping" method automatically when a DC operating point calculation fails both normal Newton's method and the automatic GMIN stepping attempt. This is an example of natural parameter continuation that steps through a scale factor that is applied to all DC voltage sources in the circuit. It is described in Section 8.5.

Another basic type of continuation in Xyce is accessed by setting `.options nonlin continuation=1`. This allows the user to sweep existing device parameters (models and instances), as well as a few reserved artificial parameter cases. The most obvious natural parameter to use is the magnitude(s) of independent voltage or current sources, the choice of which is equivalent to "source stepping" in SPICE. Section 8.2 provides a Xyce source-stepping example. For some circuits (as in the aforementioned Schmidt trigger), source stepping leads to turning points in the continuation.

A special type of continuation, an algorithm designed specifically for MOSFET circuits [37], involves two internal MOSFET model parameters—one for the MOSFET gain, and the other for the nonlinearity of the current-voltage relationship. This algorithm is invoked with `.options nonlin continuation=2`, and has proven to be effective in some large MOSFET circuits. Section 8.6 provides a detailed example.

8.2. Natural Parameter Continuation

Figure 8-1 shows a natural parameter continuation netlist with parameters pertinent to the continuation algorithm highlighted in red. For this example, the parameter being swept is the DC value of the voltage source `VDDdev`. This example demonstrates a version of "source stepping" that is similar to SPICE, but only applied to the single voltage source `VDDdev` rather than to all voltage sources. For an example of source stepping in which every source in simultaneously swept, see sections 8.2.2 and 8.5.

Using continuation on the magnitudes of independent voltage and current sources is a fairly obvious technique when a DC operating point calculation fails to converge. However, a naïve application of natural parameter continuation to single voltage and current sources does not often enable convergence in practice. Xyce will apply source stepping automatically during the DC operating point calculation if both the standard Newton's method and GMIN stepping fail. So, normally, this technique of manually forcing stepping of a single source is unnecessary.

```
MOS level 1 model CMOS inverter
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3
* Continuation Options

.options nonlin continuation=1
.options loca stepper=0 predictor=0 stepcontrol=1
+ conparam=VDDdev
+ initialvalue=0.0 minvalue=-1.0 maxvalue=5.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=5.0 aggressiveness=1.0
+ maxsteps=100 maxnliters=200

VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 8-1. Example natural parameter continuation netlist. This example of source stepping shows a circuit that does not require continuation to run. Most circuits complex enough to require continuation would not fit on a single page.

8.2.1. Explanation of Parameters, Best Practice

Figure 8-1 also illustrates the following "best practice" rules:

- `.options nonlin continuation=1`. Sets the algorithm to use natural parameter continuation.
- `.options loca conparam=VDDdev`. If using natural parameter continuation, it is necessary include a setting for `conparam`. It sets which input parameter to use to perform continuation. The parameter name is subject to the same rules as parameter used by the `.STEP` capability. (Section 7.4.2). In this case, the parameter is the magnitude of the DC voltage source, VDDdev. For this type of voltage source, it was possible to use the default device parameter (Section 7.4.5)
- `.options loca initialvalue=0.0`. This is required.
- `.options loca maxvalue=5.0`. This is required.
- `.options loca stepcontrol=1` or `.options loca stepcontrol=adaptive`. This specifies the continuation steps to be adaptive, rather than constant. This is recommended.
- `.options loca maxsteps=100`. This sets the maximum number of continuation steps for each parameter.
- `.options loca maxnliters=200`. This is the maximum number of nonlinear iterations, and has precedence over the similar number that can be set on the `.options nonlin` line.
- `.options loca aggressiveness=1.0`. This refers to the step size control algorithm, and the value of this parameter can be anything from 0.0 to 1.0. 1.0 is the most aggressive. In practice, try starting with this set to 1.0. If the solver fails, then reset to a smaller number.

8.2.2. Source Scaling Continuation

Figure 8-2 shows a natural parameter continuation netlist with parameters pertinent to the continuation algorithm highlighted in red. For this example, a special parameter `VSRCSCALE` is being swept from zero to one. This parameter is applied as a scaling factor for all DC sources in the netlist. As a result, this example demonstrates an explicit invocation of “source stepping” like the one that both Xyce and SPICE apply automatically as part of their DC operating point solution strategies if Newton’s method and GMIN stepping both fail.

Note that while the special parameter has the name `VSRCSCALE`, it would be applied in the same manner to current sources if they were present in the circuit. The name is an artifact of an earlier implementation.

Note also that a more convenient method to accomplish source stepping is described in section 8.5.

```
*Simple netlist demonstrating source-stepping
*This source will be swept with .DC
V1 1 0 DC 1
R1 1 0 100
*This source will be not swept with .DC
V2 2 0 DC 8
R2 2 0 100
.DC V1 1 8 1
.print DC V(1) V(2)
.print HOMOTOPY V(1) V(2)
* Continuation Options
.options nonlin continuation=1

.options loca stepper=0 predictor=0 stepcontrol=0
+ conparam=vsrcscale
+ initialvalue=0.0 minvalue=-1.0 maxvalue=1.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=0.2
+ aggressiveness=1.0
+ maxsteps=5000 maxnliters=200

.END
```

Figure 8-2. Example natural parameter continuation netlist implementing source stepping over all DC vsources. This particular example of source stepping shows a circuit that does not require continuation to run.

8.3. Natural Multiparameter Continuation

It is possible to use the natural parameter continuation specification to have Xyce sweep multiple parameters in sequential order. This requires specifying many of the parameters in the `.options loca` statement as vectors, delineated by commas, rather than as single parameters.

This is a usage example — the circuit itself does not require continuation to run. Most circuits complex enough to require continuation would not fit on a single page.

8.3.1. Explanation of Parameters, Best Practice

The solver parameters set in figure 8-3 are the same as those from figure 8-1, but many of them are in vector form. Specify any parameters specific to the continuation variable as a vector, including `conparam`, `initialvalue`, `minvalue`, `maxvalue`, `initialstepsize`, `minstepsize`, `maxstepsize`, and `aggressiveness`. Otherwise, the specification is identical.

```
MOS level 1 model CMOS inverter
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3
* Continuation Options
.options nonlin continuation=1
.options loca stepper=0 predictor=0 stepcontrol=adaptive
+ conparam=mosfet:gainscale,mosfet:nltermscale
+ initialvalue=0.0,0.0
+ minvalue=-1.0,-1.0
+ maxvalue=1.0,1.0
+ initialstepsize=0.2,0.2
+ minstepsize=1.0e-4,1.0e-4
+ maxstepsize=5.0,5.0
+ aggressiveness=1.0,1.0

VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 8-3. Example multiparameter continuation netlist. This netlist reproduces MOSFET continuation with a manual specification.

8.4. GMIN Stepping

GMIN stepping is a type of continuation commonly available in circuit simulators. Like SPICE, Xyce automatically attempts GMIN stepping if the initial operating point fails, and if the attempt at GMIN stepping fails, it will subsequently attempt source stepping 8.5. As such, it is not typically necessary to manually specify GMIN stepping in a Xyce netlist.

However, GMIN stepping may be manually specified by setting `continuation=3` or more conveniently, `continuation=gmin`. If it is manually specified, Xyce will *not* attempt to find a DC operating point using any other method; it will attempt GMIN stepping, and if that fails it will exit with error, not attempting any other method. Figure 8-4 provides a netlist example of GMIN stepping, in which the method has been explicitly requested.

```
Simple GMIN stepping example.
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3

* Continuation Options
.options nonlin continuation=gmin


VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 8-4. Example GMIN stepping netlist. The continuation parameter is gmin. It can also be specified using continuation=3.

The name ”GMIN stepping” can be somewhat confusing, as ”GMIN” is also a user-specified device package parameter (unrelated to this algorithm) that one may set. In the device context, “GMIN” refers to a minimum conductance applied to many device models to enhance convergence. In the continuation context, it refers to the conductance of resistors attached from every circuit node to ground.

The conductance, which is the continuation parameter, is initially very large, and is iteratively reduced until the artificial resistors have a very high resistance. At the end of the continuation, the resistors are removed from the problem. At this point, assuming the continuation has been successful, the original user-specified problem has been solved.

8.5. Source Stepping

Source stepping is a type of continuation commonly available in circuit simulators. Like SPICE, Xyce automatically attempts source stepping if the initial operating point fails, and if the subsequent attempt at GMIN stepping 8.4 also fails. As such, it is not typically necessary to manually specify source stepping in a Xyce netlist.

However, source stepping may be manually specified by setting `continuation=34` or more conveniently, `continuation=sourcestep`. If it is manually specified,Xyce will *not* attempt to find a DC operating point using any other method; it will attempt source stepping, and if that fails it will exit with error, not attempting any other method. Figure 8-5 provides a netlist example of source stepping.

Note that a less convenient method to accomplish source stepping is described in section 8.2.2.

```
Simple test of explicit source stepping
****************************************
R1 1 0 1K
V1 1 0 5V

.OP
.PRINT DC V(1) I(V1)
.PRINT HOMOTOPY V(1) I(V1)
* Continuation Options
.options nonlin continuation=sourcestep

.END
```

Figure 8-5. Example source stepping netlist.

8.6. MOSFET Continuation

Figure 8-6 contains a MOSFET continuation example netlist, and is the same circuit as was used in figure 8-6, except some of the parameters are different. As before, the lines pertinent to the continuation algorithm are highlighted in red.

```
THIS CIRCUIT IS A MOS LEVEL 1 MODEL CMOS INVERTER
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3


* Continuation Options
.options nonlin continuation=mos


VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 8-6. MOSFET continuation netlist example. This is a usage example — the circuit itself does not require continuation to run. Most circuits complex enough to require continuation would not fit on a single page.

8.6.1. Explanation of Parameters, Best Practice

There are a few differences between the netlist in figures 8-1 and 8-6. This example shows one set of options, but there are numerous options of working combinations.

MOSFET continuation requires only `.options nonlin continuation=2` or `.options nonlin continuation=mos` parameters, which specifies use of the special MOSFET continuation. This is a two-pass continuation, in which first a parameter concerning gain is swept from 0 to 1, and then a parameter relating to the nonlinearity of the transfer curve is swept from 0 to 1. The default parameters will work for a variety of MOSFET circuits, so it often will be unneccessary to override them using an `.options loca` line. However, it is possible to override the default parameters using the same `.options loca` parameters described in Section 8.2.1.

8.7. Pseudo Transient

Pseudo transient continuation is very similar to GMIN stepping, in that both algorithms involve placing large artificial terms on the Jacobian matrix diagonal, and progressively making these terms smaller until the original circuit problem is recovered. One difference is, rather than doing a series on Newton solves, pseudo transient does a single nonlinear solve while progressively modifying the pseudo transient parameter. Figure 8-7 provides an example of pseudo transient continuation options.

```
* Continuation Options
.options nonlin continuation=9

.options loca
+ stepper=natural
+ predictor=constant
+ stepcontrol=adaptive
+ initialvalue=0.0
+ minvalue=0.0
+ maxvalue=1.0e12
+ initialstepsize=1.0e-6
+ minstepsize=1.0e-6
+ maxstepsize=1.0e6
+ aggressiveness=0.1
+ maxsteps=200
+ maxnliters=200
+ voltagescalefactor=1.0
```

Figure 8-7. Pseudo transient solver options example. The continuation parameter is set to 9.

8.7.1. Explanation of Parameters, Best Practice

Pseudo transient has not been observed to be as successful as MOSFET-continuation for large MOSFET circuits. However, it may be a good candidate for difficult non-MOSFET circuits as it tends to be faster because the total number of matrix solves is smaller.

8.8. Arc Length Continuation

Most of the forms of continuation described in this chapter are low-order, and thus cannot be used to track turning points in the solution. However, the ability to track turning points is a capability that can be enabled in Xyce. A simple example netlist, which is based on a third order polynomial (which thus has multiple DCOP solutions) is given in figure 8-8. The solution, which exhibits this turning point behavior is given in figure 8-9.

```
Test for turning points using arclength continuation
* polynomial coefficients:
.param A=3.0
.param B=-2.0
.param C=1.0
.param I=1.0

Vtest 1 0 5.0
Btest 1 0 V=A*(I(Vtest)-I)**3 + B*(I(Vtest)-I) + C
.DC Vtest 1 1 1

* natural parameter continuation options (via loca)
.options nonlin continuation=1

* stepper sets what order of continuation this is.
*   stepper=0 or stepper=NAT  is natural continuation
*   stepper=1 or stepper=ARC  is arclength continuation
*
* predictor must be set to secant to see turning points
*   predictor=0  tangent
*   predictor=1  secant
*   predictor=2  random
*   predictor=3  constant
.options loca stepper=1
+ predictor=1 stepcontrol=1
+ conparam=Vtest
+ initialvalue=0.0 minvalue=0.0 maxvalue=2.0
+ initialstepsize=0.01 minstepsize=1.0e-8 maxstepsize=0.1
+ aggressiveness=0.1
.print homotopy I(Vtest)
```

Figure 8-8. Arclength continuation example. An explanation of some of the important parameters is given in the comments.

8.8.1. Explanation of Parameters, Best Practice

Most of the parameters specified in the netlist 8-8 are the same as the ones described in the natural parameter section 8.2.1. However, two of them must be specified to non-default values to enable an

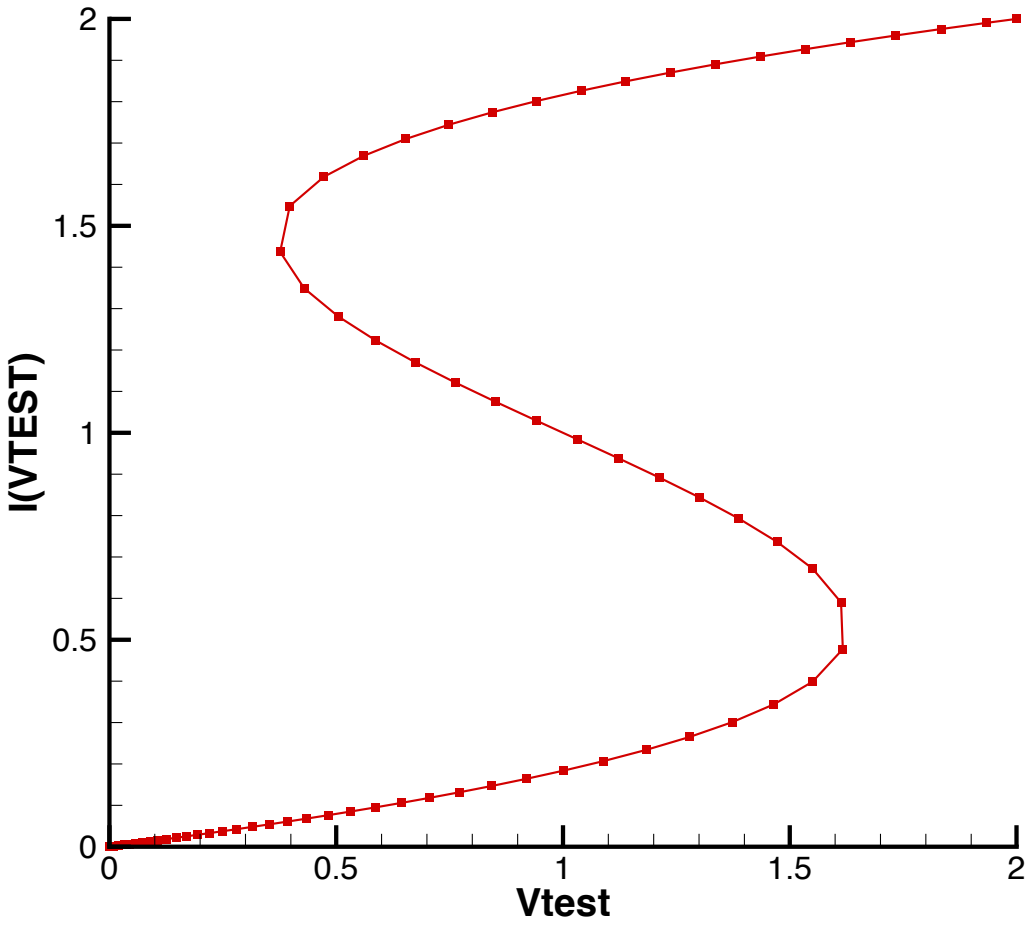

Figure 8-9. Arclength continuation example result. This result was generated using Xyce with the netlist in figure 8-8.

arclength calculation. Specifically, the `stepper` parameter must be set to `1` or `ARC`, and the `predictor` parameter must be set to `2` or `SECANT`.

Chapter Overview

This chapter illustrates how to output simulation results to data or output files and includes the following sections:

- Section 9.1, Control of Results Output
- Section 9.1.2, Additional Output Options
- Section 9.2, Output Analysis
- Section 9.3, Graphical Display of Solution Results

9.1. Control of Results Output

Xyce supports one solution output command, `.PRINT`, which is quite flexible, and supports several output formats.

9.1.1. `.PRINT` Command

The `.PRINT` command sends the analysis results to an output file. Xyce supports several options on the `.PRINT` line of netlists that control the format of the output.

Multiple .PRINT lines may be present in the netlist. Only .PRINT lines appropriate for the analysis being executed are activated. Each analysis type has a set of analysis print types. These analysis print types are used to specify the variables desired for each of the different output files which may be generated by an analysis type. If an additional analysis print type is activated and no .PRINT for that analysis print type is present, the variable list and options fall back to the .PRINT of that analysis type.

General Format:

```
.PRINT <print type> [options] <output variable> [<output variable>]*
```

Table 9-1 lists the available print types for the analyses.

Table 9-2 gives the various options available to the `.PRINT` command.

Table 9-3 gives the various output formats available to the `.PRINT` command.

The `<output variable>` parameter can be a variety of requested outputs, including nodal voltages, `V(...)`, or device currents, `I(...)`, and power, `P(...)` or `W(...)`, as given by

- `V(<node name>)`
- `V(<node name>,<node name>)` (the voltage difference between the first and second nodes)
- `I(<two-terminal device>)`
- `Ik(<three-or-more-terminal device>)` (the `k` indicates the device node from which to acquire the value, which is device specific; see the Xyce Reference Guide [3] for details)
- `P(two-terminal device)`
- `W(two-terminal device)`

At this time, power calculations are only supported for `.DC` and `.TRAN` analysis types. Power calculations may also not be supported for all Xyce devices yet. Consult the Xyce Reference Guide [3] for more details. As an example, the power supplied or dissipated by the voltage source `V` is calculated as $I \cdot \Delta V$ where the voltage drop is calculated as $(V_+ - V_-)$ and positive current flows from V_+ to V_-. Dissipated power has a positive sign, while supplied power has a negative sign. An important note is that the power calculations are a post-processing step, which places a limit on the accuracy of circuit-wide "energy conservation" calculations (e.g., total power supplied by sources - total power dissipated in non-source devices) in Xyce. The accuracy of the inputs (`V` and `I`) to the power calculations is limited by the nonlinear solver tolerances, and the error in the power calculations is upper-bounded by the sum of the product-terms of `V*(error in I)` and `I*(error in V)`.

Table 9-1. .PRINT Print Types

Analysis	Print Type	Description
.AC	AC	Sets default variable list and formats for print subtypes
.AC	AC_IC	Overrides variable list and format for AC initial conditions
.DC	DC	
.EMBDEDDEDSAMPLING	ES	
.HB	HB	
.HB	HB_FD	Overrides variable list and format for HB frequency domain
.HB	HB_IC	Overrides variable list and format for HB initial conditions
.HB	HB_STARTUP	Overrides variable list and format for HB start up
.HB	HB_TD	Overrides variable list and format for HB time domain
.NOISE	Noise	Outputs Noise spectral density curves
.TRAN	TRAN	
Specialized Output Commands		
Homotopy	HOMOTOPY	Sets variable list and format for homotopy
.SENS	SENS	Sets variable list and format for sensitivity

Voltage variables specified in the frequency domain have special processing to handle complex results. For file formats which have a complex output capability, the complex value is written. However, for file formats, such as STD and CSV, the complex value is written as two columns of data, the real part followed by the imaginary part. Pseudo names may also be used to compute scalar values from a complex voltage variable. These are given in Table 9-4.

For .AC and .NOISE analyses, current variables (see Table 9-4) are only supported for devices that have "branch currents" that are part of the solution vector. This includes the V, E, H and L devices. It also includes the voltage-form of the B device.

In addition to the above, internal device variables can be specified as an <output variable>. These take the form, N(device variable). The format of the device variable called by N is device-specific, and exact forms can be found in the Xyce Reference Guide [3].

It is also possible to output a device or model parameter directly, such as the resistance of a resistor R5, specified as R5:R on the .PRINT line. It is necessary to specify both the device name and parameter name, separated by a colon.

Table 9-2. .PRINT command options.

Option...	Action...
FORMAT= <STD|NOINDEX|PROBE|TECPLOT|RAW|CSV>	Controls the output format. See Table 9-3. The *default* is STD.
FILE=<filename>	Output filename. The *default* is the netlist filename with ".prn" appended. foo.cir.prn, where foo.cir is the input netlist filename. The suffix depends on the format. The *default* suffixes for FORMAT=STD|PROBE|TECPLOT|RAW|CSV are ".prn", ".csd", ".dat", ".raw", and ".csv", respectively.
WIDTH=<field-width>	Column width for the output data
PRECISION=<floating-point-precision>	Number of significant digits past the decimal point
FILTER=<floor-value>	Absolute value below which output variables will be printed as 0.0
DELIMITER=<TAB|COMMA>	Alternate delimiter between columns of output in the STD output format. The default is spaces.

Table 9-3. .PRINT FORMAT options.

Format...	Action...
STD	Outputs data in standard columns
NOINDEX	Outputs the same as the STD except the index column is omitted.
PROBE	Output is formatted to be compatible with the PSpice Probe plotting utility.
RAW	Output conforms to the Spice binary rawfile. Use the **-a** command line option to produce an ASCII rawfile.
TECPLOT	Output for use in the TecPlot graphics package.
CSV	Produces a comma-separated value format.
GNUPLOT	Output data in standard columns, with improved Gnuplot compatibility for .STEP data.

Finally, an expression may also be specified as an <output variable>. To do so, enclose the expression within curly braces ({}). See Section 4.3 for a description of expressions.

Example:

```
.PRINT TRAN FILE=Output.prn V(3) I(R3) ID(M5) V(4)
.PRINT DC FORMAT=TECPLOT FILE=Output.dat V(2) {I(C3)+abs(V(4))*5.0}
```

Table 9-4. Pseudo Variables for Complex Output

Variable	Definitions
VR(*node*)	Voltage Real Component
VI(*node*)	Voltage Imaginary Component
VM(*node*)	Voltage Magnitude
VP(*node*)	Voltage Phase, Degrees
VDB(*node*)	Voltage Magnitude, Decibels
VR($node_1$,$node_2$)	Difference of Voltage Real Component
VI($node_1$,$node_2$)	Difference of Voltage Imaginary Component
VM($node_1$,$node_2$1)	Difference of Voltage Magnitude
VP($node_1$,$node_2$)	Difference of Voltage Phase, Degrees
VDB($node_1$,$node_2$)	Difference of Voltage Magnitude, Decibels
IR(*device*)	Current Real Component
II(*device*)	Current Imaginary Component
IM(*device*)	Current Magnitude
IP(*device*)	Current Phase, Degrees
IDB(*device*)	Current Magnitude, Decibels
SR($index_1$,$index_2$)	S-Parameter Real Component
SI($index_1$,$index_2$)	S-Parameter Imaginary Component
SM($index_1$,$index_2$)	S-Parameter Magnitude
SP($index_1$,$index_2$)	S-Parameter Phase, Degrees
SDB($index_1$,$index_2$)	S-Parameter Magnitude, Decibels
YR($index_1$,$index_2$)	Y-Parameter Real Component
YI($index_1$,$index_2$)	Y-Parameter Imaginary Component
YM($index_1$,$index_2$)	Y-Parameter Magnitude
YP($index_1$,$index_2$)	Y-Parameter Phase, Degrees
YDB($index_1$,$index_2$)	Y-Parameter Magnitude, Decibels
ZR($index_1$,$index_2$)	Z-Parameter Real Component
ZI($index_1$,$index_2$)	Z-Parameter Imaginary Component
ZM($index_1$,$index_2$)	Z-Parameter Magnitude
ZP($index_1$,$index_2$)	Z-Parameter Phase, Degrees
ZDB($index_1$,$index_2$)	Z-Parameter Magnitude, Decibels

```
.PRINT HB RA:R V(Vsrc) VM(Vsrc)
.PRINT HB_TD RA:R V(Vsrc)
.PRINT TRAN FORMAT=CSV R1:R R1:TEMP I(R1) R2:R R2:TEMP
.PRINT AC v(3) .PRINT AC FILE=AUX.FD.prn v(1) .OP
.PRINT AC_IC v(1) v(2)
```

9.1.2. Additional Output Options

Additional control of .PRINT line output can be set using the .OPTIONS OUTPUT specification.

9.1.2.1. Output Intervals

An important feature of the .OPTIONS OUTPUT command is to provide control of the interval at which transient data is written to files specified by .PRINT TRAN commands. This can be especially useful in controlling the size of the results file for simulations that require a large number of time steps. The default behavior is for Xyce to output results at every time step, so using this feature to reduce output frequency will result in improved performance.

General Format:

```
.OPTIONS OUTPUT INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1>
...]]
```

INITIAL_INTERVAL=<interval> specifies the starting interval time for output and <tx ix> specifies later simulation times (tx) where the output interval will change to (ix).

The following example shows the output being requested (via the netlist .OPTIONS OUTPUT command) every $.1\mu s$ for the first $10\mu s$, every $1\mu s$ for the next $10\mu s$, and every $5\mu s$ for the remainder of the simulation:

Example: .OPTIONS OUTPUT INITIAL_INTERVAL=.1us 10us 1us 20us 5us

9.1.2.2. Suppressing output file header and footer

It can be convenient to have an output file that contains only numerical data with minimal formatting. There are two options, PRINTHEADER and PRINTFOOTER available on the .OPTIONS OUTPUT line to facilitate this. The format is given by:

General Format:

```
.OPTIONS OUTPUT PRINTHEADER=<boolean> PRINTFOOTER=<boolean>
```

So, to produce an output file that has no header and no footer, simply set:

Example: .OPTIONS OUTPUT PRINTHEADER=false PRINTFOOTER=false

Note that PRINTHEADER is only supported for .PRINT FORMAT=<STD|GNUPLOT|SPLOT>, while PRINTFOOTER is only supported for .PRINT FORMAT=<STD|GNUPLOT|SPLOT|TECPLOT>.

9.1.2.3. Outputting all solution variables to file

It can be convenient to have all the solution variables output to file during a transient run without specifying all of them on a .PRINT TRAN line. This can be accomplished with the SNAPSHOTS option available on the .OPTIONS OUTPUT line. The format is given by:

General Format:

```
.OPTIONS OUTPUT SNAPSHOTS=<boolean>
```

So, to produce an output file that has all the solution variables, simply set:

Example:

```
.OPTIONS OUTPUT SNAPSHOTS=true
```

Note that setting SNAPSHOTS will result in the parser ignoring any solution nodes specified in the .PRINT TRAN line.

9.1.3. Output File Redirection

There are three ways to re-direct the output of a Xyce simulation run to a file other than the default file. They are the -r and -o command line options and the FILE= qualifier on individual .PRINT lines. This subsection provides an overview of them, and how they work with each other.

9.1.3.1. -r Output

The -r <fileName> command line option will output all of the variables in the solution vector to the binary rawfile <fileName>. The command line -r <fileName> -a will output a human-readable (ASCII) rawfile instead. The -r command line option is supported for the .AC, .DC, .NOISE and .TRAN analysis types.

If the -r and -o command line options are both specified then the -r command line option only applies to the output to the binary rawfile <fileName>. The -o command line option still applies to the other outputs, such as from .MEASURE lines, as described below. Finally, Xyce does not permit the -r output to overwrite the netlist file (<netlistName>). Instead, the output will be placed into the file <netlistName>.raw.

If -r output is requested for a netlist that has .PRINT SENS or .PRINT HOMOTOPY lines in it then the sensitivity and homotopy output will use the instructions given by those .PRINT lines (or by the -o command line). This approach was taken because those two analysis types do not support rawfile output. However, an attempt to re-direct the sensitivity and homotopy output, with a FILE= qualifier on their .PRINT lines, to the same file as requested with the -r <fileName> command line option will be a fatal parsing error.

9.1.3.2. -o Output

The -o command line option is supported for compatibility with other circuit simulators such as Spice3f5. As mentioned above, if the -r and -o command line options are both specified then the -r command line option only applies to the binary rawfile <fileName>. Xyce does not permit the -o output, from any of the .PRINT lines in the netlist, to overwrite the netlist file (<netlistName>).

The -o <fileName> command line option is supported for all analysis types. The output file(s) from the netlist's .PRINT lines will be in FORMAT=STD with spaces as the delimiter between columns. So, the -delim command line option and any DELIMITER=, FILE= and FORMAT= qualifiers on the relevant .PRINT lines will be silently ignored. Those output(s) will be placed into a set of files with the "basename" <fileName>. So, for a .TRAN analysis, this command line would produce the output file <results>.prn:

```
Xyce -o results <netlistName>
```

The results from any .MEASURE TRAN, .MEASURE TRAN_CONT, .FFT and/or .FOUR lines in the netlist would be placed into the files <results>.mt0, <results>_measurename.mt0, <results>.fft0, and <results>.four0, respectively. For a .STEP analysis, the results file <results>.res would also be made.

For a .AC analysis, that also has a .OP statement in the netlist, the previous command line would produce output files with the correct default extensions, which are <results>.FD.prn and <results>.TD.prn in this case. Similar rules apply for the .PRINT output for .ES, .HB, .NOISE and .PCE analyses.

For backward compatibility with Xyce 7.3, and earlier, these two command lines are equivalent. The "default print extension" of .prn on the file name requested with -o is ignored.

```
Xyce -o results.prn <netlistName>
Xyce -o results <netlistName>
```

There are two limitations on -o output. First, the dashoFilename can not be a directory name. Second, if the dashoFilename contains a directory path (e.g., <path>/filename) then <path> must already exist. Xyce will not create new directories. (Note: These two limitations also exist for the -r, -l, -namesfile and -rsf command line options.)

9.1.3.3. FILE= Qualifier on .PRINT Lines

The FILE=<fileName> qualifier on individual .PRINT lines is often a preferred option to the -o command line option for output file redirection, since it is more flexible. However, as noted above, the -r and -o command line options do take precedence if either of them are specified.

The combination of FILE= and FORMAT= qualifiers on the .PRINT lines in a netlist allows fine-grained control of the output formats, output file names and variable lists in each file. As a simple example, this combination of .PRINT lines would produce output files in both PROBE (<netlistName.csd>) and CSV (<netlistName.csv>) formats without having to do multiple simulation runs.

```
.PRINT TRAN FORMAT=CSV V(1) V(2)
.PRINT TRAN FORMAT=PROBE V(1) V(2)
```

Finally, Xyce does not permit the FILE= qualifier to overwrite the netlist file (<netlistName>). Instead, the output will be placed into the file <netlistName>.<defaultExtension>, where the default extension is set by the FORMAT= qualifier on that .PRINT line and the analysis type specified in the netlist.

9.1.3.4. Multi-File Output for AC and HB Analysis

The .AC and .HB analysis types can produce multiple output files. This can occur in two ways. For example, a netlist may have explicit .PRINT HB_FD and .PRINT HB_TD lines. In that case, the frequency domain and time domain outputs for that netlist's .HB analysis will use the explicit variable lists, FILE= qualifiers, and FORMAT= qualifiers from each line. As a convenience, Xyce can also generate "fallback print lines". For example, a .PRINT HB will automatically generate the output from equivalent .PRINT HB_FD and .PRINT HB_TD lines using its variable list, FILE= qualifier, and FORMAT= qualifier.

The interactions and precedence between these types of output are as follows. First, a more explicit line (e.g., .PRINT HB_FD or .PRINT AC_IC) takes precedence over a less explicit line (e.g., .PRINT HB or .PRINT AC), if both appear in the netlist. Second, if a FILE=hbFile qualifier appears on a .PRINT HB line, that is using FORMAT=STD, then its four possible output files will be hbFile.HB.FD.prn, hbFile.HB.TD.prn, hbFile.hc_ic.prn and hbFile.startup.prn. Finally, if a FILE=acFile qualifier appears on a .PRINT AC line, that is using FORMAT=STD, then its two possible output files will be acFile and acFile.TD.prn.

9.2. Output Analysis

9.2.1. .MEASURE

Xyce supports analysis of the data from a simulation through the .MEASURE command. Using .MEASURE one can locate extrema in a voltage or current node, calculate integrals, derivatives, Fourier transforms, and locate transient events. (Note: .MEAS is an allowed synonym for .MEASURE in Xyce.)

General Format:

```
.MEASURE <analysis type> <measure name> <measure type> <simulation
variable>
+ [qualifiers]
```

<analysis type> is the analysis under which the measure should be calculated. Currently, transient (.TRAN), dc (.DC), ac (.AC) and noise (.NOISE) analyses are supported, and .MEASURE can be used with .STEP in all four cases. The specified measure is referenced by <measure name>. That name is used in the summary output at the end of the simulation to report the value of that measure, and can also be used in .PRINT statements. The <measure type> is the type of calculation to be done. For .TRAN analyses, the supported measure types are:

- AVG: Computes the arithmetic mean.
- DERIV: Computes the derivative of a simulation variable, either at a user-specified time or when the simulation variable hits a user-specified value.

- DUTY: Fraction of time that a given simulation variable is greater than ON and does not fall below OFF (ON and OFF are defined later in this section).
- EQN: Calculates the value of a Xyce expression during the simulation.
- ERR, ERR1 and ERR2: Calculate the error between simulation variables.
- ERROR: Calculates the norm between the measured waveform and a "comparison waveform" specified in a file. The supported norms are L1, L2 and INFNORM. The default norm is the L2 norm. The descriptive output for each ERROR measure, that is printed to standard output, will explicitly state which norm was used for each ERROR measure.
- FIND-AT: Returns the value of a solution variable when the independent variable (time, frequency or DC sweep value) reaches the value specified in the AT clause. If the conditions specified in the AT clause are not found during the simulation then the measure will return the default value of 0 (or the user-specified default value).
- FIND-WHEN: Returns the value of a solution variable when the solution variable specified in the WHEN clause reaches a specified fixed value or is equal to another solution variable. If the conditions specified for finding the value specified in the WHEN clause are not found during the simulation then the measure will return the default value of 0 (or the user-specified default value).
- FOUR: Calculates the Fourier transform of the solution variable for the .TRAN analysis type using the fundamental frequency AT. By default, the DC component and first nine harmonics are computed; more can be reported by setting NUMFREQ to the desired value. More interpolation points can be used in the Fourier analysis by setting GRIDSIZE, which is 200 by default. For this measure, the phase data is output in degrees.
- FREQ: An estimate of the frequency of a solution variable found by cycle counting during the simulation. Thresholds are defined through the values of ON and OFF.
- INTEG: Calculates the integral of a solution variable through second order numerical integration.
- MAX: Returns the maximum value of a solution variable.
- MIN: Returns the minimum value of a solution variable.
- OFF_TIME: Returns the time that a solution variable is below OFF, and not greater than ON for the simulation, normalized by the number of cycles of the waveform during the simulation.
- ON_TIME: Returns the time that a solution variable is above ON, and not less than OFF for the simulation, normalized by the number of cycles of the waveform during the simulation.
- PP: Returns the difference between the maximum value and the minimum value of a solution variable during the simulation.
- RMS: Computes the root-mean-squared value of a solution variable.
- TRIG-TARG: Measures the time between a trigger event and a target event.
- WHEN: Returns the time when a solution variable reaches a specified fixed value or is equal to another solution variable. If the conditions specified for finding the value specified in the WHEN clause are not found during the simulation then the measure will return the default value of 0 (or a user-specified default value).

The AVG, DERIV, EQN, ERR, ERR1, ERR2, FIND-AT, FIND-WHEN, INTEG, MIN, MAX, PP, RMS, TRIG-TARG and WHEN measures are supported for all four "measure modes" (TRAN, DC, AC and NOISE).

The ERROR measure is Xyce-specific, and is supported for all four "measure modes" (TRAN, DC, AC and NOISE). The DUTY, FREQ, FOUR, OFF_TIME and ON_TIME measures are also Xyce-specific, and are only supported for TRAN measure mode.

Xyce also supports "continuous" measures, that can return more than one value, for .TRAN, .DC, .AC and .NOISE analyses. The corresponding measure modes are TRAN_CONT, DC_CONT, AC_CONT and NOISE_CONT. The supported measure types for continuous measures are DERIV-AT, DERIV-WHEN, FIND-AT, FIND-WHEN, WHEN and TRIG-TARG measures.

For transient analyses, that also have .FFT analysis statements, Xyce also supports FFT measure mode. In that case, the following additional measure types can be associated with those .FFT lines. In addition, FFT mode measures, such as FIND-AT measures, can be used in EQN measures to create more customized metrics from the .FFT results.

- ENOB: computes the Effective Number of Bits.
- FIND-AT: returns the requested FFT cofficient at the requested frequency, which is rounded to the nearest harmonic.
- SFDR: computes the Spurious Free Dynamic Range.
- SNDR: computes the Signal to Noise-plus-Distortion Ratio.
- SNR: computes the Signal to Noise Ratio.
- THD: computes the Total Harmonic Distortion.

The <simulation variable> specifies a voltage or current node that will be used in this measure, such as V(a). The measure WHEN is different from the other measures in that it can take one or two solution variables. For example, WHEN v(a)=5 returns the time when V(a) equals 5. Or, if WHEN V(a) = V(b) is specified, the time when V(a) equals V(b) is returned.

The .MEASURE command can also take optional [qualifiers] that limit the time window (or frequency window or DC sweep values) when .MEASURE is applied. The [qualifiers] also place numeric limits on what state a value is considered to be in (e.g., ON and OFF), and provide numeric qualification on comparisons of values (e.g., MINVAL). A partial list of the supported qualifiers is as follows. See the Xyce Reference Guide [3] for more details on these qualifiers, and for the complete list of supported qualifiers for each combination of analysis type and measure type.

- FROM=value A time (or frequency or DC sweep value) after which the measurement calculation will start.
- TO=value A time (or frequency or DC sweep value) after which the measurement calculation will end.
- TD=value A time delay before which the measurement should be taken or checked.
- RISE=r|LAST The number of rises after which the measurement should be checked. If LAST is specified, then the last rise found in the simulation will be used.

- FALL=f|LAST The number of falls after which the measurement should be checked. If LAST is specified, then the last fall found in the simulation will be used.
- CROSS=c|LAST The number of zero crossings after which the measurement should be checked. If LAST is specified, then the last zero crossing found in the simulation will be used.
- MINVALUE=value An allowed absolute difference between the simulation variable and the variable to which it is being compared. This has a default value of 1.0e-12. One may need to specify a larger value to avoid missing the test condition in a transient run.
- ON=value The value at which a signal is considered to be on for frequency, duty and on time calculations
- OFF=value The value at which a signal is considered to be off for frequency, duty and off time calculations.

An example of using .measure during a transient analysis is shown in the following netlist:

```
VS  1  0  SIN(0 1.0 1KHZ 0 0)
VP  2  0  PULSE( 0 1 0.2ms 0.2ms 0.2ms 1ms 2ms )

R1  1  0  100
R2  2  0  100

.TRAN 0  10ms
.PRINT TRAN FORMAT=NOINDEX V(1) V(2)

.MEASURE TRAN avg1 AVG V(1)
.MEASURE TRAN avg2 AVG V(2)

.MEASURE TRAN duty1 DUTY V(1) ON=0.75 OFF=0.25
```

The measure avg1 returns the average of v(1), and avg2 returns the average of v(2). Additionally, duty1 computes the fraction of time that v(1) is above 0.75 V, without falling below 0.25 V.

The next netlist provides an example of using the when measure during a transient analysis:

```
VS  1  0  SIN(0 1.0 1KHZ 0 0)
VP  2  0  PULSE( 0 100 0.2ms 0.2ms 0.2ms 1ms 2ms )

R1  1  0  100
R2  2  0  100

.TRAN 0  10ms 0 1.0e-5
.PRINT TRAN FORMAT=NOINDEX V(1) V(2)

.MEASURE TRAN hit1_75 WHEN V(1)=0.75 MINVAL=0.02
.MEASURE TRAN hit2_75 WHEN V(1)=0.75 MINVAL=0.08 RISE=2
```

In the above netlist, the measure called hit1_75 will return the simulation time where v(1) reaches a value of 0.75, while hit2_75 returns the second time that v(1) reaches a value of 0.75. The MINVAL option acts at an absolute tolerance in this case. So, the above measure statements are more exactly interpreted as hit1_75 is the simulation time when v(1) reaches a value of 0.75 ± 0.02 and hit2_75 is the simulation time when v(1) reaches a value of 0.75 ± 0.08 on its second rise.

Measured results are reported to the output and log file. Additionally, for TRAN measures, the results are stored in files called circuitFileName.mt#, where the suffixed number (#) starts at 0 and increases for multiple iterations (.STEP iterations) of a given simulation. Each line of this file will contain the measurement name, followed by its value for that run and/or step iteration. For DC measures, the results are stored in the files circuitFileName.ms#, while AC and NOISE measures use the files circuitFileName.ma#.

Xyce also supports an ability to *re-measure* results from a prior run of Xyce. In this mode, one can modify .MEASURE and/or .FFT statements and have Xyce recalculate the results based on existing simulation data. This can be useful if the original simulation takes some time to complete. (Note: remeasure works for both transient and dc analyses. However, it may be most useful for transient analyses. It is not currently supported for ac or noise analyses.) The syntax in the netlist is the same and the only thing that changes is how one invokes Xyce. For example, if the netlist is called myNetlist.cir and the existing output file is called myNetlist.cir.prn, then remeasure is accomplished with:

```
Xyce -remeasure myNetlist.cir.prn myNetlist.cir
```

There are some important limitations to remeasure. First, the data used by the .MEASURE and/or .FFT commands must be present in the output file because, in this mode, Xyce is not recalculating the full solution. Second, .prn, .csv and .csd files are supported, but remeasure may only work for .csv and .csd files generated by Xyce.

9.2.2. .FOUR

Fourier analysis can be performed as a part of the transient analysis using the .FOUR command.

General Format:

```
.FOUR freq ov1 <ovn>*
```

freq is the fundamental frequency used for Fourier analysis. The ov1 parameter is the desired solution output to be analyzed, specifically:

- V(<node name>)
- V(<node name>,<node name>) (the voltage difference between the first and second nodes)
- I(<two-terminal device>)
- Ik(<three-or-more-terminal device>) (see the .PRINT section, 9.1.1, for more detail)
- N(device variable) (see the .PRINT section, 9.1.1, for more detail)
- SENS Transient direct sensitivities specified by the .SENS command (see the sensitivity analysis section, 7.10 for more detail)

- The pseudo-variables given in Table 9-4 for complex output handling

At least one solution variable must be specified, but Fourier analysis can be performed on several solution variables for each fundamental frequency, `freq`. Multiple `.FOUR` lines may be used in a netlist. All results from Fourier analysis will be returned to the user in a file with the same name as the netlist file suffixed with `.four#`, where the suffixed number (`#`) starts at `0` and increases for multiple iterations (`.STEP` iterations) of a given simulation. For this analysis, the phase data is output in degrees.

If `SENS` is specified as the output, then all the direct sensitivities specified by the `.SENS` command will be processed, as it is difficult to manually specify the full name of a sensitivity variable. So, if one wants to use sensitivities as the output, do the following:

```
.FOUR 20MEG SENS
```

Fourier analysis is performed over the last period (`1/freq`) of the transient simulation. The dc component and the first nine harmonics are calculated. The number of harmonics computed by `.FOUR` is static. So, the main difference between `.FOUR` and the Fourier analysis in `.MEASURE` is that the latter allows the user to select the number of harmonics. The default options for the Fourier analysis in `.MEASURE` is the same as `.FOUR`. For instance, these two lines will result in the same Fourier analysis:

```
.FOUR 20MEG V(2)
.MEASURE TRAN FOURV2 FOUR V(2) AT=20MEG
```

The Fourier analysis in `.MEASURE` will allow for more harmonics to be computed using the `NUMFREQ` option and more interpolation points to be used in the Fourier analysis with `GRIDSIZE`. For instance, to compute twenty harmonics (including the dc component), the previous `.MEASURE` line can be amended to:

```
.MEASURE TRAN FOURV2 FOUR V(2) AT=20MEG NUMFREQ=20
```

To increase the number of interpolation points from 200, which is the default, to 500, the line can be amended to:

```
.MEASURE TRAN FOURV2 FOUR V(2) AT=20MEG NUMFREQ=20 GRIDSIZE=500
```

For maximum accuracy of the Fourier analysis, it is recommended that the time integration option `DELMAX` should be set to `period/100`. This is the preferred approach to improving the accuracy of the Fourier analysis versus increasing the number of interpolation points.

9.2.3. .FFT

A Fast Fourier Transform (FFT) analysis can be performed as a part of the transient analysis using the .FFT command. The simplest invocation would be:

General Format:

.FFT <ov>

where the ov1 parameter is the desired solution output to be analyzed, specifically:

- V(<node name>)
- V(<node name>,<node name>) (the voltage difference between the first and second nodes)
- I(<two-terminal device>)
- Ik(<three-or-more-terminal device>) (see the .PRINT section, 9.1.1, for more detail)
- N(device variable) (see the .PRINT section, 9.1.1, for more detail)

Other useful optional parameters include the following:

- NP=value The number of points in the FFT. This value must be a power of 2. The default value is 1024.
- WINDOW=value The windowing function that will be applied to the sampled waveform values. The allowed values are as follows:
  - RECT = rectangular window (default)
  - BART = Bartlett (triangular) window
  - HANN = Hanning window
  - HAMM = Hamming window
  - BLACK = Blackman window
  - HARRIS = Blackman-Harris window
- START=value The start time for the FFT analysis. The default value is the start time for the transient analysis.
- STOP=value The end time for the FFT analysis. The default value is the end time for the transient analysis.

The Xyce Reference Guide [3] provides more details on these optional parameters, as well as additional ones.

For maximum accuracy of the FFT analysis, it is recommended that the default setting of .OPTIONS FFT FFT_ACCURATE=1 be used. This setting instructs Xyce to insert breakpoints at the sample times requested by the collection of .FFT statements in the netlist.

By default, the results from each of the .FFT lines in the netlist are sent to the <netlistName>.fft0 file. In addition, metrics for each .FFT line can be sent to both stdout and the <netlistName>.fft0 file by setting .OPTIONS FFT FFTOUT=1. In that case, a sorted list of the 30 largest harmonics for each .FFT line will be also sent to the stdout. Those additional metrics include the Effective Number of

Bits (ENOB), Spurious Free Dynamic Range (SFDR), Signal to Noise Ratio (SNR), Signal to Noise-and-Distortion Ratio (SNDR) and Total Harmonic Distorion (THD). The Xyce Reference Guide [3] provides detailed definitions for these metrics. It also discusses the compatibility of various `.OPTIONS OUTPUT` and `.OPTIONS FFT` settings.

Xyce also supports an ability to *re-measure* results from a prior run of Xyce. In this mode, one can modify `.MEASURE` and/or `.FFT` statements and have Xyce recalculate the results based on existing simulation data. Section 9.2.1 and the Xyce Reference Guide [3] provides more details on this capability.

9.3. Graphical Display of Solution Results

Although Xyce does not provide integrated graphical display options, it produces output in a form that may readily be used with commonly available graphical tools, including TecPlot, gnuplot, and MS Excel (see figure 9-1 for an example plot using TecPlot, `http://www.amtec.com`). The standard Xyce print format (`FORMAT=STD` or `FORMAT=NOINDEX`) is well suited for use with gnuplot. However, `FORMAT=GNUPLOT` may be useful when plotting `.STEP` data in gnuplot, as it automatically inserts two blank lines between the blocks of step-data that are otherwise in standard format. Comma separated variable (`FORMAT=CSV`) is the best choice for import into Excel. `FORMAT=TECPLOT` produces output specifically targeted at the TecPlot tool. The `FORMAT=PROBE` option to the `.PRINT` command produces output `.csd` files that can be read by the PSpice Probe utility. See the PSpice Users Guide [4] for instructions on using the Probe tool, and the Xyce Reference Guide [3] for more details on the `.PRINT` command options.

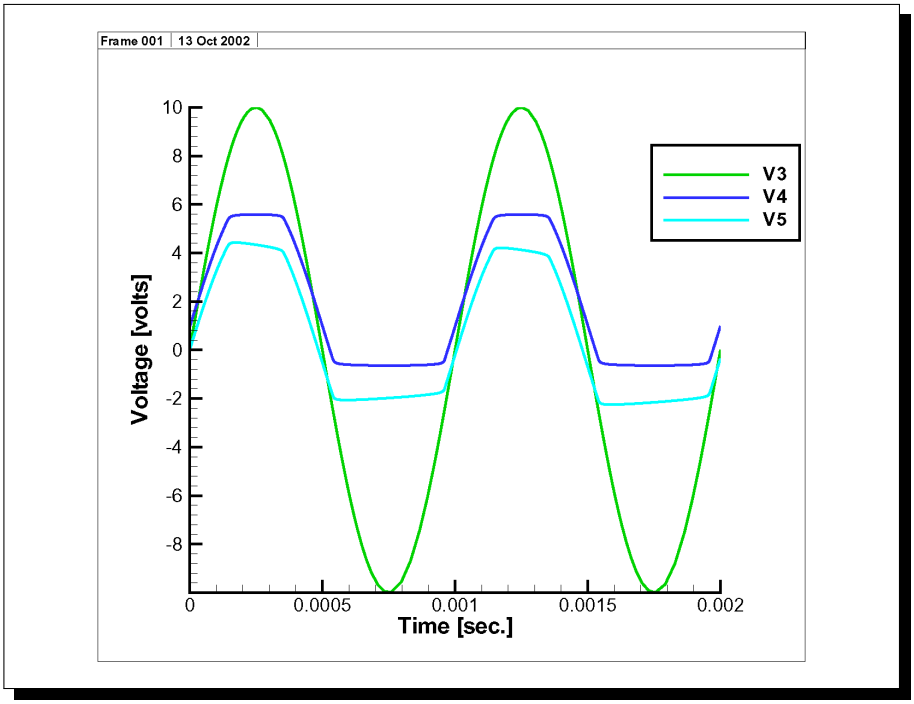

Figure 9-1. TecPlot plot of diode clipper circuit transient response from Xyce `.prn` file.

This page intentionally left blank.

10. CIRCUIT DIAGNOSTICS AND TROUBLESHOOTING

Chapter Overview

This chapter provides guidance on using Xyce's diagnostic output to understand simulation performance and aid in correcting circuit problems.

- Section 10.1, *Diagnostic Output*

10.1. Diagnostic Output

10.1.1. Introduction

To aid in understanding circuit simulation performance, Xyce has a diagnostic output mode where additional data can be sent to an external file for the user to study. The data can include information about GMIN stepping and source stepping during the DC operating point calculation as well as times and values of currents and voltages that are either extreme or over some user defined limit. Additionally, there is a capability to output solution nodes that appear to be discontinuous within a user specified limit. The data is sent to a separate file to avoid it being lost in the normal simulation output. The following sections will present examples of using diagnostic output in a circuit.

10.1.2. Enabling Diagnostic Output

To enable diagnostic output in a simulation add the line

```
.options diagnostic <diagnostic mode>
+ diagfilename=<output file name>
```

Figure 10-1. Basic diagnostic option line

where `<diagnostic mode>` can be any of the following modes

- `EXTREMALIMIT=<value>` Output the single most extreme solution value whose absolute value is greater than `value`.
- `CURRENTLIMIT=<value>` Output all of the branch currents and lead currents whose absolute value is greater than `value`.
- `VOLTAGELIMIT=<value>` Output all of the solution voltages whose absolute value is greater than `value`
- `DISCLIMIT=<value>` Output all solution variables where the absolute value of the difference between the predictor and corrector is greater than `value`

Multiple modes can be specified and the results will be grouped by diagnostic mode for each step within the diagnostic output file.

Finally, the `.options diagnostic diagfilename` is optional. If it is not given then the diagnostic output will be saved in a file called `[input file name].dia`. No matter whether it is set, or left at its default value, Xyce will overwrite any existing file with new output each time Xyce is run.

The next sections will give examples for each of the diagnostic modes listed above.

10.1.3. DC Operating Point Information

When Xyce attempts to calculate the DC operating point of a circuit, if the initial Newton solver approach doesn't converge then Xyce will automatically attempt GMIN stepping and then source stepping to find a solution. Information about the DC Operating point calculation and if GMIN or source stepping are used will be sent to the diagnostic output file. For example this is the output one might see if both GMIN and source stepping are needed:

```
Xyce Dagnostic Output
Analysis event INITIALIZE DC
Analysis event STEP_STARTED DC
Analysis event DC_OP_STARTED DC
Analysis event STEP_FAILED DC
Analysis event DC_OP_GMIN_STEPPING DC
Analysis event  DC_OP_GMIN_STEPPING_FAILED DC smallest gmin = 1.057478e-03
Analysis event DC_OP_SOURCE_STEPPING DC
Analysis event STEP_SUCCESSFUL DC
```

In this example the default DC OP calculation is started and fails as reported with:

```
Analysis event DC_OP_STARTED DC
Analysis event STEP_FAILED DC
```

Then GMIN stepping is tried and it fails at a GMIN of around 1e-3.

```
Analysis event DC_OP_GMIN_STEPPING DC
Analysis event  DC_OP_GMIN_STEPPING_FAILED DC smallest gmin = 1.057478e-03
```

Then source stepping is attempted and that is successful.

```
Analysis event DC_OP_SOURCE_STEPPING DC
Analysis event STEP_SUCCESSFUL DC
```

Understanding how Xyce obtained or failed to obtain a DC Operating point can be helpful in diagnosing a circuit topology or initial condition problem.

10.1.4. Extrema Output

To obtain the solution node name and value that has the largest absolute value over some limit, use the `EXTREMALIMIT` option.

In this simple Xyce input netlist, `EXTREMALIMIT` is used to output values over 6.0 Volts

```
* 4049 Circuit

X2 node3 node8 node2 0 CD4049UB
V1 node2 0 5
C1 node8 node5 1.088N
R1 node3 node5 46.5K
R2 node1 node5 98.6K
X1 node1 node3 node2 0 CD4049UB

.tran .05U 500U NOOP

.options timeint method=gear abstol=1.0e-7
+ reltol=1.0e-6

.options diagnostic EXTREMALIMIT=6.0
+ diagfilename=ExtremaCheck1.dia

.print tran V(node8) v(node5) V(node1) V(node3)

* Subcircuit and Model Definitions
.SUBCKT CD4049UB 1  2   3   4
*HEX BUFFER    IN OUT VCC VSS
M1 2 1 3 3 CD4049P
M2 2 1 4 4 CD4049N
.MODEL CD4049P PMOS (LEVEL=1 VTO=-2.9 KP=2M
+ GAMMA=3.97U PHI=.75 LAMBDA=1.87M RD=28.2
+ RS=45.2 IS=31.2F PB=.8 MJ=.46 CBD=148P
+ CBS=177P CGSO=218N CGDO=182N CGBO=299N)
.MODEL CD4049N NMOS (LEVEL=1 VTO=2.1 KP=5M
+ GAMMA=3.97U PHI=.75 LAMBDA=1.87M RD=4.2
+ RS=4.2 IS=31.2F PB=.8 MJ=.46 CBD=105P
+ CBS=127P CGSO=156N CGDO=130N CGBO=214N)
.ENDS

.END
```

Figure 10-2. Sample use of the diagnostic EXTREMALIMIT

Running this netlist in Xyce will produce a diagnostic output file with

```
Xyce Diagnostic Output
 Solution Extrema output requested with extremalimit = 6
 Extreme value found in TRAN analysis at STEP_STARTED time=0.000174793
     V(NODE5)=6.0347
 Extreme value found in TRAN analysis at STEP_SUCCESSFUL time=0.000174793
     V(NODE5)=6.0347
```

The output shows what type of diagnostic was requested, the diagnostic limit imposed and then points at when the limit was reached in the solution. In this case, the voltage node NODE5 exceeded the limit of 6 with a value of 6.0347 at the simulation time of 0.000174793 seconds.

The extrema output diagnostic can be useful to determine the largest values encountered during a simulation. Once the extrema are known it can be useful to focus on when voltages or currents exceed a set value. That will be examined in the next section.

10.1.5. Voltage and Current Output

Similar to EXTREMALIMIT, the diagnostic output options, VOLTAGELIMIT and CURRENTLIMIT allow reporting on voltages and currents at different limits. Additionally, unlike EXTREMALIMIT, VOLTAGELIMIT and CURRENTLIMIT report **all** voltages and currents that are over a given limit at each step.

Thus, the line

```
.options diagnostic VOLTAGELIMIT=5.0 CURRENTLIMIT=3e-2 diagfilename=VICheck1.dia
```

will report voltage variables that have absolute values greater than 5.0 and currents with absolute values greater than 3.0e-2. If used in the oscillator circuit from Figure 10-2, the diagnostic output will look like:

```
Xyce Diagnostic Output
 Node voltage output requested with voltagelimit = 5
 Lead current output requested with currentlimit = 0.03
...
Voltage over limit value found in TRAN analysis at STEP_STARTED time=7.18202e-05
     V(M:X2:1_drainprime)=5.00228
     V(M:X2:1_sourceprime)=5.00023
     V(NODE8)=5.00232
     V(M:X2:2_drainprime)=5.00233
...
Current over limit value found in TRAN analysis at STEP_STARTED time=0.000102911
     solution I(L3_branch)=0.0300498
     solution I(V2_branch)=-0.0300498
     lead I(L3:BRANCH_D)=0.0300498
     lead I(R5:BRANCH_D)=0.0300469
     lead I(V2:BRANCH_D)=-0.0300498
```

Note that unlike the extrema diagnostic, the voltage and current diagnostic report all solution values and branch currents whose absolute values exceed the limits. This can be useful in understanding the operational ranges of currents and voltages within a circuit.

10.1.6. Discontinuity Output

Discontinuous behavior in a solution variable can cause problems for step progression in transient analysis as the time integrator will try to minimize dramatic changes in the solution. The time integrator does this by comparing the predicted solution that is based on the trajectory of the prior solution points with the actual solution, i.e. corrector, found by the nonlinear solver. If the difference between the predicted solution and the corrector is greater than some tolerance, then the time integrator will reject that step and try a smaller time step. When the circuit has a true discontinuity in it, smaller time steps will not make the discontinuity smaller and time integration may ultimately fail.

The following example circuit has discontinuous behavior in the output from the device `bsrc` which changes from 0 to 1 when the voltage on node 1 goes above 1. Normally this would cause the transient simulation to fail, but this netlist also instructs the time integrator to ignore predictor-corrector error and just accept steps where the nonlinear solver converges. This option is **not recommended**, but used just for illustrative purposes.

```
*
* circuit with an intentional discontinuity
*

Vin 1 0 pulse 0 5 1e-7 1e-6 1e-6 1e-3 2e-3

rload 1 0 10


bsrc 3 0 v = {if(v(1)>1, 1,0)}
rload3 3 0 100


.options timeint erroption=1

.options diagnostic DISCLIMIT=1e-2
+ diagfilename=DisconCheck2.dia

.print tran v(1) v(3)
.tran 0 1e-2

.end
```

Figure 10-3. Example circuit with an intentional discontinuity

When this circuit is run in Xyce, the diagnostic output will look like:

```
Xyce Diagnostic Output
 Discontinuity output requested with disclimit = 0.01
 ...
 Discontinuity over limit value found in TRAN analysis at STEP_STARTED time=4.27e-0
```

```
    V(3)=1 xPredictor  = 0 qPredictor  = 0 |diff| = 1
```

The output shows what type of diagnostic was requested, the diagnostic limit imposed and then points at when node 3 had a discontinuity of 1 at the simulation time where node 1 triggered the step change in output. If we used Xyce's standard time integration options and removed the `.options timeint erroption=1` line from the above netlist then Xyce would fail with a time-step-too-small error. The diagnostic output would show:

```
Xyce Diagnostic Output
 Discontinuity output requested with disclimit = 0.01
 ...
 Discontinuity over limit value found in TRAN analysis at STEP_STARTED time=3.01602
     V(1)=0.990008 xPredictor  = 1.06201 qPredictor  = 0 |diff| = 0.072
 Discontinuity over limit value found in TRAN analysis at STEP_FAILED time=3.01602e
     V(1)=0.990008 xPredictor  = 1.00801 qPredictor  = 0 |diff| = 0.018
```

At the start of a transient step `V(1)=0.990008` and the predicted value for `V(1)` the end of that step would be 1.06201. That would trigger `V(3)` to jump from 0 to 1 and thus the step fails. When running the time integration with the default step-error control, the actual discontinuity at node 3 is harder to see because its effect is limiting the change on node 1. However, given that the simulation is having problems when node 1 is trying to cross the voltage threshold 1.0, it provides useful insight into where the problem is. Thus, for best use of discontinuity output, it may be necessary to try turning step error control on and off in a set of simulations.

Finally, it is recommended to not set `DISCLIMIT` too small, i.e. below the time integrator `ABSTOL`, as this will cause most of the solution variables to be reported as discontinuous.

This page intentionally left blank.

11. GUIDANCE FOR RUNNING XYCE IN PARALLEL

Chapter Overview

This chapter provides guidance for running a parallel version of Xyce, and includes the following sections:

- Section 11.1, *Introduction*
- Section 11.2, *Processor Affinity on Linux systems*
- Section 11.3, *Problem Size*
- Section 11.4, *Linear Solver Options*
- Section 11.5, *Transformation Options*
- Section 11.6, *Device Distribution Options*

11.1. Introduction

Xyce is designed from the ground up to be distributed-memory parallel, supported by the message-passing interface (MPI) standard. Although many of the issues pertinent to running in parallel are still being researched, Xyce is mature enough that some general principles have emerged for efficiently running problems in a parallel environment. In addition to the information in this chapter, reference [38] provides supplemental information about Xyce parallel performance.

Parallel simulations must be run from the command line. Section 2.2.1 provides information about the parallel execution syntax for Xyce.

11.2. Processor Affinity on Linux systems

Beginning with release 1.8 of OpenMPI, the default behavior on Linux systems is for OpenMPI to apply "processor affinity" to runs invoked with mpirun. This is intended to improve performance of parallel runs by preventing the processes from moving around the system, possibly degrading memory access efficiency. Since most users of OpenMPI are trying to maximize the performance of their systems, this is generally a good change.

Prior to that release, the default behavior was to allow MPI processes to be moved from processor to processor by the Linux system as needed.

If you are the only person running MPI jobs on a system, and you are only running one job, this change will have no impact on you, but if you are running more than one MPI job at a time, or more than one user on your system is running MPI jobs and the system is not using a resource manager such as slurm, then this change has a profound impact that must be understood and adapted to with mpirun options.

11.2.1. Default OpenMPI Behavior with Processor Affinity Support

At the time of this writing, OpenMPI's mpirun will, by default, bind processes to a core if the number of processors requested is 2 or less. If the number of processes requested is greater than 2, it will bind processes to a socket.

What this means is that once your run starts, each process of your parallel job will be locked onto the processor (core or socket) on which it started.

11.2.2. Why You Have to Know About This

Unfortunately, OpenMPI by default allocates processors to processes in a round-robin fashion starting from the lowest numbered processor on the system, and each mpirun makes this determination without any access to what other mpirun invocations have done.

The effect of this is that if you start five identical 2-processor parallel runs of Xyce simultaneously on a 16-core system and don't add extra mpirun options, each of these five jobs will be locked to the *same* two processors (processors 0 and 1) on the system.

Rather than getting five jobs run in the time it takes to run one, each will only get 20% of the two processors and take at least five times longer to run than a single job would have.

The same thing would be true if five different users each ran one 2-processor Xyce job. All jobs would be running on processors 0 and 1 of this 16-processor system.

Therefore, unless you are the only person on your system and you are only running one job, you must be aware if OpenMPI is using processor affinity on your system, and take appropriate steps to avoid oversubscribing cores.

11.2.3. Affected Systems

At the time of this writing, OpenMPI supports processor affinity only on Linux systems, and so the issues of this section apply only to Linux. The OS X kernel does not support jobs setting their own processor affinity. Some other systems have processor affinity support in their kernels, but it is not yet supported by OpenMPI on those systems.

Systems that are running a resource manager such as slurm (which includes all of Sandia's high performance computing systems) are not impacted by this issue, because slurm will allocate a specific set of CPUs to be exclusive to your job, and no other jobs of yours or other users can run on these CPUs.

11.2.4. mpirun Command Line Options to Change Default Behavior

If squeezing maximum performance out of the hardware is not important, then the simplest way to avoid the oversubscription issue described above is to use the `-bind-to none` option to mpirun. This was the default behavior of mpirun prior to release 1.8 of OpenMPI and the default behavior of OpenMPI on every system other than Linux. While the `-bind-to none` option will potentially allow your mpi processes to move around the system and have degraded memory performance, it will *NOT* accidentally stack multiple jobs onto the same small set of processors. Any number of mpirun jobs may be run with this option, and they will not compete for resources unless the Linux task scheduler makes them do so.

If you do not want to do without the benefits that processor affinity can bring, you can manually specify the set of CPUs that OpenMPI will use for your job by using the `-cpu-set` option to mpirun, chosing as your CPU set some numbered processors that you know are not being used by other MPI jobs. For example, `mpirun -np 2 -cpu-set 2-3` will run your 2 processor job with the jobs locked to processors 2 and 3 instead of to processors 0 and 1. You will have to make sure to use a different CPU set for each job, and make sure that you are not using the same CPU set as some other user on the system.

There are other options for modifying OpenMPI's processor binding behavior, so consult OpenMPI documentation if you wish to understand them better.

Finally, if you really want to solve the problem correctly, reaping both the benefits of processor affinity and the simplicity of using mpirun's defaults, you can install and configure a resource manager on your system. This is a topic that is far outside the scope of Xyce documentation. See `http://slurm.schedmd.com/` for documentation on one such resource manager.

11.3. Problem Size

Running Xyce in parallel is often useful for circuits with thousands of devices or more. However, due to the overhead of interprocessor communication, there is an optimal number of processors that will achieve the best performance. This number is dependent upon many factors, including the number and type of devices, the topology of the circuit, and the characteristics of the computing architecture. It is difficult to know a priori what this optimal number of processors is. However, it is apparent when that optimal number is exceeded because, as the number of processors is increased, the total simulation time will also increase. This is due to the increasing amount of required communication and decreasing amount of work per processor. In other words, the benefit of distributing the problem is outweighed by the communication overhead, so increasing the processor count beyond this optimal point is counterproductive.

11.3.1. Ideal Problem Size

In general, a circuit needs to be relatively large to take full advantage of the parallel capability of Xyce. However, parallelism is achieved in two distinct phases of the code: the device evaluation and the linear solve. The device evaluation is, as the name implies, the evaluation of all the device equations in order to compute the residual vector and Jacobian entries for Newton's method. Xyce distributes the number of devices over the number of processors in parallel, so their evaluation enables speedups in the total simulation time even for thousands of devices.

The linear solve phase is more computationally complex. The Jacobian matrix generated by most circuits is sparse and has heterogeneous structure, in that there is not a regular sparsity pattern in the matrix nonzeros. Sparse, direct linear solvers have proven to be efficient on these types of linear systems up into the tens to hundreds of thousands of unknowns. They become less efficient for linear systems in the hundreds of thousands of unknowns. This is where iterative linear solvers can provide scalable performance because of their inherent parallelism. Unfortunately, the effectiveness of iterative linear solvers is dependent upon preconditioning the linear system (see Section 11.4.6). The benefit of direct over iterative linear solvers is that they rarely fail to compute a solution, so direct linear solvers are the more robust option for enabling simulations to complete.

In general, there are three modes in which Xyce can be executed: "Serial load, serial solve", "Parallel load, serial solve", and "Parallel load, parallel solve". Each of these modes optimizes the amount of available parallelism for a given linear system size, as summarized in Table 11-1. The "load" refers to the device evaluation phase combined with the assembly of the Jacobian matrix and residual vector, while the "solve" refers to the linear solve phase. "Serial load, serial solve" is the only mode of computation that a serial version of Xyce will perform, but it can also be obtained in a parallel version of Xyce by using only one MPI processor. Both of the "Parallel load" simulation modes require a parallel build of Xyce, where the linear solver method can be a direct method ("serial solve") or an iterative method ("parallel solve") using the options discussed in Section 11.4. Hybrid linear solvers, which combine the best attributes of both direct and iterative methods, provide a robust and scalable option. They are not reflected in Table 11-1, but more information about these types of linear solvers will be discussed in Section 11.4.7.

11.3.2. Smallest Possible Problem Size

Circuits consist of a discrete set of components (voltage nodes, devices, etc.). For parallel simulation, it is preferable that Xyce be able to put at least one discrete component of the problem on each processor. In

Table 11-1. Xyce simulation modes.

Mode	Linear System Size	Reason
"Serial load, serial solve"	10^0 - 10^3	MPI overhead cannot speed up device evaluation or linear solve.
"Parallel load, serial solve"	10^3 - 10^4	Distributed device evaluations can speed up the simulation, but iterative linear solvers are not more efficient than direct methods.
"Parallel load, parallel solve"	10^4 or more	Distributed device evaluations can speed up the simulation and so can iterative linear solvers, if an efficient preconditioner is available.

practice, this means the circuit should be distributed across fewer processors than the number of nodes and devices it contains.

11.4. Linear Solver Options

The different linear solvers available in Xyce are:

- KLU
- KSparse
- SuperLU and SuperLU DIST (optional)
- The AztecOO iterative solver library
- The Belos iterative solver library
- The ShyLU hybrid solver library (optional)

AztecOO and Belos are the parallel iterative solvers and KLU, KSparse, and SuperLU (optional) are the serial direct solvers that are available for both serial and parallel builds of Xyce. If KLU, KSparse, or SuperLU is used with a parallel version of Xyce, the devices are evaluated and linear problem is assembled in parallel, but the linear system is solved in serial on one processor. This can be quite effective for circuits with tens of thousands of devices or fewer (see Table 11-1). The ShyLU hybrid linear solver, which combines the robustness of a direct solver with the scalability of an iterative solver, will be discussed in Section 11.4.7.

The user can specify the solver through the `.OPTIONS LINSOL` control line in the netlist. The default linear solver used by Xyce is described in Table 11-2. By default, a parallel version of Xyce uses AztecOO as the linear solver when the linear system is larger than ten thousand unknowns. For any linear system smaller than ten thousand unknowns, Xyce uses KLU as the linear solver. A serial version of Xyce uses KLU as its default linear solver. To use a solver other than the default the user needs to add the option "`TYPE=<solver>`" to the `.OPTIONS LINSOL` control line in the netlist, where `<solver>` is '`KLU`,' '`KSPARSE`,' '`SUPERLU`,' '`SUPERLUDIST`,' '`AZTECOO`,' '`BELOS`,' or '`SHYLU`.'

Table 11-2. Xyce default linear solver.

Solver	Version	Linear System Size
KLU	Serial	*all*
KLU	Parallel	$< 10^4$ unknowns
AztecOO	Parallel	$\geq 10^4$ unknowns

11.4.1. KLU

KLU is a serial, sparse direct solver native to the Amesos package in Trilinos [39] and is the default solver for serial builds of Xyce. KLU is the default solver for small circuits in parallel builds of Xyce as well, but this requires the linear system to be solved on one processor and the solution communicated back to all processors. As long as the linear system can fit on one processor, KLU is often a superior approach to using an iterative linear solver. So, if a parallel build of Xyce is run in serial on a circuit that generates a linear system larger than ten thousand unknowns and the simulation fails to converge, then specifying KLU as the linear solver may fix that problem.

Some of the solver parameters for KLU can be altered through the '`.OPTIONS LINSOL`' control line in the netlist. Table 11-3 lists solver parameters and their default values for KLU.

Table 11-3. KLU linear solver options.

Option	Description	Default Value
`KLU_repivot`	Recompute pivot order each solve	1 (true)
`output_ls`	Write out linear systems solved by KLU to file every # solves	0 (no output)
`output_base_ls`	Write out linear systems before any transformations to file every # solves	0 (no output)
`output_failed_ls`	Write out linear systems KLU failed to solve to file	0 (no output)

11.4.2. KSparse

KSparse is a serial, sparse direct solver based on Ken Kundert's sparse solver, Sparse 1.3. Kundert's sparse solver was developed as part of the SPICE circuit simulation code. KSparse is built, by default, in Xyce. Similar to KLU, KSparse can be used in a parallel version of Xyce, but the linear system is solved on one processor.

11.4.3. SuperLU and SuperLU DIST

SuperLU is a serial, sparse direct solver and SuperLU DIST is a parallel, sparse direct solver with an interface in the Amesos package. SuperLU and SuperLU DIST support are *optionally* built in Xyce, so they are not available by default in any Xyce build or provided binary. Furthermore, to enable SuperLU and SuperLU DIST support in Xyce, it is necessary to build SuperLU and SuperLU DIST support in Amesos/Trilinos. Similar to KLU, SuperLU can be used in a parallel version of Xyce, but the linear system is solved on one processor. SuperLU DIST can only be used in a parallel version of Xyce, the Amesos

interface handles the redistribution of the matrix into the format required by SuperLU DIST. Xyce does not allow modifications to SuperLU and SuperLU DIST solver parameters.

11.4.4. AztecOO

AztecOO is a package in Trilinos [39] that offers an assortment of iterative linear solver algorithms. Xyce uses the Generalized Minimal Residual (GMRES) method [40] from this suite of iterative solvers. Some of the solver parameters for GMRES can be altered through the '`.OPTIONS LINSOL`' control line in the netlist. Table 11-4 provides a list of solver parameters for AztecOO and their default values.

Table 11-4. AztecOO linear solver options.

Option	Description	Default Value
`AZ_max_iter`	Maximum allowed iterations	200
`AZ_tol`	Iterative solver (relative residual) tolerance	1.0e-9
`AZ_kspace`	Krylov subspace size	50
`output_ls`	Write out linear systems solved by AztecOO to file every # solves	0 (no output)
`output_base_ls`	Write out linear systems before any transformations to file every # solves	0 (no output)

11.4.4.1. Common AztecOO Warnings

If Xyce is built with the verbosity enabled for the linear algebra package, it is not uncommon to see warnings from AztecOO usually indicating the solver returned unconverged due to a numerical issue.

> **NOTE:** AztecOO warnings *do not* indicate the entire simulation has failed, Xyce uses a hierarchy of solvers so if the iterative linear solver fails, the nonlinear solver or time integrator will usually make adjustments and attempt the step again; so the warnings can often be ignored. If the entire simulation eventually fails (i.e., gets a "time-step-too-small" error), then the AztecOO warnings might contain clues as to what went wrong.

The simplest reason for AztecOO to return unconverged would be when the maximum number of iterations is reached, resulting in the following warning:

```
*********************************************************************
Warning: maximum number of iterations exceeded without convergence
*********************************************************************
```

Another reason AztecOO may return unconverged is when the GMRES Hessenberg matrix is ill-conditioned, which is usually a sign that the matrix and/or preconditioner is nearly singular, resulting in the following warning:

```
**************************************************************
Warning: the GMRES Hessenberg matrix is ill-conditioned.  This may
indicate that the application matrix is singular. In this case, GMRES
may have a least-squares solution.
**************************************************************
```

It is also common to lose accuracy when either the matrix or preconditioner, or both, are nearly singular. GMRES relies on an estimate of the residual norm, called the recursive residual, to determine convergence. Xyce uses the recursive residual instead of the actual residual for computational efficiency. However, numerical issues can cause the recursive residual to differ from the actual residual. When AztecOO detects but cannot rectify this situation, it outputs the following warning:

```
**************************************************************
Warning: recursive residual indicates convergence
though the true residual is too large.

Sometimes this occurs when storage is overwritten (e.g. the
solution vector was not dimensioned large enough to hold
external variables). Other times, this is due to roundoff. In
this case, the solution has either converged to the accuracy
of the machine or intermediate roundoff errors occurred
preventing full convergence. In the latter case, try solving
again using the new solution as an initial guess.
**************************************************************
```

11.4.5. Belos

Belos is a package in Trilinos [39] that offers an assortment of iterative linear solver algorithms. Many of the algorithms available in Belos can also be found in AztecOO. However, Belos offers a few computational advantages because its solvers are implemented using templated C++. In particular, AztecOO can solve linear systems only in double-precision arithmetic, while Belos can solve linear systems that are complex-valued or in extended-precision arithmetic. At this time, Xyce is using a subset of Belos capabilities, the default method is GMRES, and the interface to Belos will recognize most of the AztecOO linear solver options, as shown in Table 11-5.

Table 11-5. Belos linear solver options.

Option	Description	Default Value
AZ_max_iter	Maximum allowed iterations	200
AZ_tol	Iterative solver (relative residual) tolerance	1.0e-9
AZ_kspace	Krylov subspace size	50
output_ls	Write out linear systems solved by Belos to file every # solves	0 (no output)
output_base_ls	Write out linear systems before any transformations to file every # solves	0 (no output)

11.4.6. Preconditioning Options

Iterative linear solvers often require the assistance of a preconditioner to efficiently compute a solution of the linear system

$$Ax = b \tag{11.1}$$

to the requested accuracy. A preconditioner, M, is an approximation to the original matrix A that is inexpensive to solve. Then (11.1) can be rewritten to include this (right) preconditioner as

$$AM^{-1}y = b, \tag{11.2}$$

where $x = M^{-1}y$ is the solution to the original linear system. If $M = A$, then the solution to the linear system is found in one iteration. In practice, M is a good approximation to A, then it will take few iterations to compute the solution of the linear system to the requested accuracy. By default, Xyce uses a non-overlapped additive Schwarz preconditioner with an incomplete LU factorization on each subdomain [41]. The parameters of the incomplete LU factorization are found in Table 11-6. This is a simple preconditioner that always works, but is not always the most effective, so other preconditioning options will be presented in this section.

Xyce provides access to preconditioning packages in Trilinos [39], such as Ifpack, through an expanded preconditioning interface. If modifications to the preconditioner are necessary, the user may specify the preconditioner through the '`.OPTIONS LINSOL`' control line in the netlist. Table 11-6 provides a list of preconditioner parameters and their default values.

Table 11-6. Preconditioner options.

Option	Description	Default Value
`prec_type`	Preconditioner	Ifpack
`AZ_ilut_fill`	ILU fill level	2.0
`AZ_drop`	ILU drop tolerance	1.0e-3
`AZ_overlap`	ILU subdomain overlap	0
`AZ_athresh`	ILU absolute threshold	0.0001
`AZ_rthresh`	ILU relative threshold	1.0001
`use_aztec_precond`	Use native ILU from AztecOO package	0 (false)
`use_ifpack_factory`	Use Ifpack factory to create preconditioner	0 (false)
`ifpack_type`	Control which preconditioner Ifpack factory creates (ILU, ILUT, Amesos)	Amesos

In practice, the choice of an effective preconditioner is highly problem dependent. By default, Xyce provides a preconditioner that works for most circuits, but is not the best preconditioner for all circuits. One simple modification to the default preconditioner that often makes it more effective is the use of a sparse direct solver on each subdomain, instead of an inexact factorization:

```
.OPTIONS LINSOL USE_IFPACK_FACTORY=1
```

This preconditioner will fail if there is a singular subdomain matrix because the KLU solver on that subdomain will fail. If numerical difficulties are not encountered during the simulation, this preconditioner is superior to inexact factorizations. A more advanced preconditioner that has been effective for certain types of circuits uses the block triangular form (BTF) permutation of the original matrix before generating the additive Schwarz preconditioner. This preconditioner, which is published in [42], will be presented in Section 11.5.4.

11.4.7. ShyLU

ShyLU is a package in Trilinos [39] that provides a hybrid linear solver designed to be a black-box algebraic solver [43]. ShyLU support is *optionally* built in Xyce, so it is not available by default in any Xyce build or provided binary. Furthermore, to enable ShyLU support in Xyce, it is necessary to build the ShyLU package in Trilinos.

ShyLU is hybrid in both the parallel programming sense - using MPI and threads - and in the mathematical sense - using features from direct and iterative methods. Xyce uses ShyLU as a global Schur complement solver [41]. This solver can be expensive, but also has proven to be a robust and scalable approach for some circuit matrices [44].

ShyLU is under active development and testing in Xyce, so a minimum number of options are provided to the user for controlling this solver. The solution approach is static, the diagonal blocks of the partitioned matrix are solved using KLU, while the Schur complement is solved using an iterative method (AztecOO's GMRES specifically). The matrix partitioning is generated using a wide separator, which is a conventional vertex separator where all the vertices that are adjacent to the separator in one of the subgraphs are added in. The only options that can be modified are shown in Table 11-7. This includes the maximum number of iterations and solver tolerance used by GMRES and the dropping threshold that ShyLU uses to generate a preconditioner for GMRES.

Table 11-7. ShyLU linear solver options.

Option	Description	Default Value
`AZ_max_iter`	Maximum allowed iterations	30
`AZ_tol`	Iterative solver (relative residual) tolerance	1.0e-12
`ShyLU_rthresh`	Relative dropping threshold for Schur complement preconditioner	1.0e-3
`output_ls`	Write out linear systems solved by ShyLU to file every # solves	0 (no output)
`output_base_ls`	Write out linear systems before any transformations to file every # solves	0 (no output)

11.5. Transformation Options

Transformations are often used to permute the original linear system to one that is easier or more efficient for direct or iterative linear solvers. Xyce has many different permutations that can be applied to remove dense rows and columns from a matrix, reduce fill-in, find a block triangular form, or partition the linear system for improved parallel performance.

11.5.1. Removing Dense Rows and Columns

The transformation that reduces the linear system through removal of all rows and columns with single non-zero entries in the matrix is called singleton filtering. The values associated with these removed entries can be resolved in a pre- or post-processing phase with the linear solve. A by-product of this transformation is a more tractable and sparse linear system for the load balancing and linear solver algorithms. This

functionality can be turned on by adding 'TR_SINGLETON_FILTER=1' to the '.OPTIONS LINSOL' control line in the netlist. This option is enabled by default whenever iterative solvers are used in Xyce.

11.5.2. Reordering the Linear System

Approximate Minimum Degree (AMD) ordering is a symmetric permutation that reduces the fill-in for direct factorizations. If given a nonsymmetric matrix A, the transformation computes the AMD ordering of $A + A^T$. This functionality may be turned on by adding 'TR_AMD=1' to the '.OPTIONS LINSOL' control line in the netlist. For parallel builds of Xyce, AMD ordering is enabled by default whenever iterative solvers are used. In parallel, the AMD ordering is performed only on the local graph for each processor, not the global graph. This is to reduce the fill-in for the incomplete LU factorization used by the additive Schwarz preconditioner, see Section 11.4.6.

11.5.3. Partitioning the Linear System

Partitioning subdivides the linear system and then distributes it to the available processors. A good partition can have a dramatic effect on the parallel performance of a circuit simulation tool. There are two key components to a good partition:

- Effective load balance
- Minimizing communication overhead.

An effective load balance ensures the computational load of the calculation is equally distributed among available processors. Minimizing communication overhead seeks to distribute the problem in a way to reduce impacts of underlying message passing during the simulation run. For runs with a small number of devices per processor the communication overhead becomes the critical issue, while for runs with larger numbers of devices per processor the load balancing becomes more important.

Xyce provides hypergraph partitioning via the **Zoltan** library of parallel partitioning heuristics integrated into Xyce. The Isorropia package in Trilinos provides access to **Zoltan** and can be controlled through the '.OPTIONS LINSOL' control line in the netlist. Table 11-8 provides the partitioning options and their default parameters. For parallel builds of Xyce, when iterative solvers are used, **Isorropia** is enabled by default to use hypergraph partitioning. The linear system is statically load balanced at the beginning of the simulation based on the graph of the Jacobian matrix.

Table 11-8. Partitioning options.

Option	Description	Default Value
TR_PARTITION	Partitioning package	0 (none), serial, 1 (Isorropia), parallel
TR_PARTITION_TYPE	Isorropia partitioner type	HYPERGRAPH

Xyce includes an expanded partitioning interface to allow the user to access multiple partitioners through Isorropia. Users may change the partitioner provided by adding 'TR_PARTITION_TYPE' to the '.OPTIONS LINSOL' control line in the netlist. There are two options for partitioning: hypergraph ('TR_PARTITION_TYPE=HYPERGRAPH') and, optionally, graph ('TR_PARTITION_TYPE=GRAPH')

partitioning through ParMETIS. Occasionally it is desirable to turn off the partitioning option, even for parallel simulations. To do so, users can add the 'TR_PARTITION=0' to the '.OPTIONS LINSOL' control line.

These techniques can be very effective for improving the efficiency of the iterative linear solvers. See the **Zoltan User Guide** [45] for more details.

11.5.4. Permuting the Linear System to Block Triangular Form

The block triangular form (BTF) permutation is often useful for direct and iterative solvers, enabling a more efficient computation of the linear system solution. In particular, the BTF permutation has shown promise when it is combined with an additive Schwarz preconditioner (see Section 11.4.6) in the simulation of circuits with unidirectional flow.

The global BTF transformation computes the permutation of the linear system to block triangular form, and uses the block structure to partition the linear system. The partitioning can be a simple linear distribution of block rows, 'TR_GLOBAL_BTF=1', or a hypergraph partitioning of block rows, 'TR_GLOBAL_BTF=2.' As the global BTF transformation includes elements of other tranformations, it is imperative to turn off other linear solver options. To use the global BTF, the linear solver control line in the netlist should contain:

```
.OPTIONS LINSOL TR_GLOBAL_BTF=<1,2> TR_SINGLETON_FILTER=1
+ TR_AMD=0 TR_PARTITION=0
```

This transformation is only useful in parallel when using a preconditioned iterative solver. It is often more effective when combined with the exact factorization of each subdomain, given by the 'USE_IFPACK_FACTORY=1' option. In practice, the structure that this transformation takes advantage of is found in CMOS memory circuits [42].

11.6. Device Distribution Options

Xyce uses different parallel distributions for the objects that evaluate the device models and the linear system. As discussed in Section 11.5.3, a good parallel partition of the linear system can improve load balance and minimize communication. The same is true for the parallel distribution of devices, since they have widely varying computational cost and the circuit connectivity graph is often irregular. Furthermore, a more efficient device distribution strategy can overlap netlist processing with the simulation setup, enabling scalable parsing for larger netlists.

Table 11-9. Distribution options.

Option	Description	Default Value
STRATEGY	Distribution strategy	0

Xyce provides three different device distribution strategies that can be controlled through the .OPTIONS DIST control line in the netlist, see Table 11-9. The default strategy (STRATEGY=0) that Xyce uses for parallel device distribution is a first-come-first-served approach, in that the total number of devices in a circuit is divided evenly over the number of parallel processors in the order of parsing. This is the simplest distribution strategy, but it does not take into account the connectivity of a circuit or balance device model

computation. Therefore, this strategy can exhibit parallel imbalance for post-layout circuits that have a substantial portion of parasitic devices.

Xyce provides two other distribution strategies that enable scalable parsing for flattened circuits or distribute devices in a more balanced manner. The "flat round robin" strategy (`STRATEGY=1`) will generate the same device distribution as the default strategy, but every parallel processor will participate in reading their portion of the netlist. In that way, this strategy provides a more scalable setup than the default strategy, but can only be applied to flattened (non-hierarchical) netlists. The "device balanced" strategy (`STRATEGY=2`) will evenly divide each of the device types over the number of parallel processors, so each processor will have a balanced number of each model type. This allieviates the parallel imbalance in the device model computation that can be experienced with post-layout circuits. However, it does not take into account the circuit connectivity, so the communication will not be minimized by this strategy.

This page intentionally left blank.

Chapter Overview

This chapter includes the following sections:

- Section 12.1, *Power Node Parasitics*
- Section 12.2, *Two Level Algorithms Overview*
- Section 12.3, *Examples*
- Section 12.4, *Restart*

12.1. Power Node Parasitics

Parasitic elements (R, L, C) are frequently required for circuit simulations to capture important circuit behavior. Most parasitic elements (interconnects, etc.) can be added to netlists without causing any difficulties for the Xyce solvers. Small circuits in particular are very robust to the addition of parasitic elements. Larger circuits, however, that must be simulated in parallel will in general tend to have more solver difficulties with the addition of parasitic devices. Of particular note are parasitic elements attached to the power and/or ground nodes of large digital circuits. An example of this is shown in figure 12-1. As these nodes tend to be highly connected, they can potentially have a very high impact on solver difficulties.

One of the parallel algorithms used by Xyce is called *singleton removal* [42], which is applied at the linear solver level and is crucial for getting many large circuits to run in parallel. This algorithm takes advantage of the fact that, in circuit simulation, some solution values are available explicitly, rather than being a quantity that needs to be calculated as the solution to a particular equation. In circuit simulation, such quantities are usually the values of independent sources. For instance, the presence of an independent voltage source at a particular node in a circuit fixes the voltage at that node to be the value of the independent source; therefore, equations reflecting the value of the voltage at that particular node do not have to be added to the set of linear equations used (in part) to determine the voltages at all the nodes in the circuit. The technique of fixing such node voltages without including them in the rest of the linear solve can be handled in a preprocessing phase referred to as the singleton removal phase.

When simulating in parallel, singleton removal is crucial as some voltage sources (especially power supplies in digital circuits) are connected to hundreds or thousands of circuit nodes. This presents a big problem in parallel because having numerous connections can often mean a communication bottleneck during the linear solve. Using singleton removal eliminates that bottleneck.

While singleton removal can result in a great improvement for circuits with ideal power supplies, for circuits with nonideal power supplies, the communication bottleneck remains. Once parasitic elements are placed between the power supply and the rest of the circuit, it is only the voltage at the circuit node directly connected to the independent source that can be removed via singleton removal. Other nodes connected to this independent source through parasitic elements have voltages that must now be solved for directly.

12.2. Two Level Algorithms Overview

Fortunately, Xyce [38] provides a workaround that allows power node parasitics to be included in large circuits without breaking singleton removal. The workaround requires the use of a two-level Newton solve,

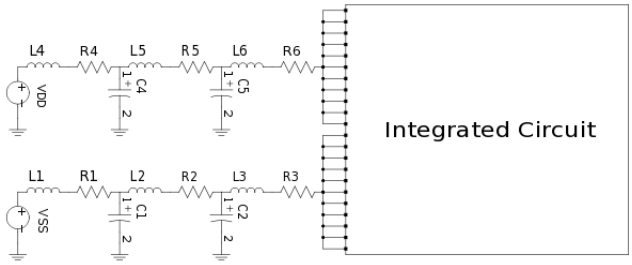

Figure 12-1. Power node parasitics example. NOTE: An RLC network sits between the VDD, VSS sources and the main circuit, so these highly connected nodes cannot be removed with a singleton filter.

in which the problem is divided into two very separate pieces, each for the most part treated as an entirely separate circuit with minimal coupling terms linking the pieces together.

For power-node problems, two-level users will typically split the netlist into ”top” and ”inner” netlists. The top netlist contains the power node parasitics and the ideal voltage sources, and very little else. The inner circuit should contain the rest of the circuit. Xyce couples the two circuits through an ”EXT” (external) device in the top circuit, and two or more independent voltage sources on the inner circuit. The values on the inner voltages are imposed from the top circuit, and the currents and conductances of the EXT device come from the inner circuit. An example is given below.

Xyce will construct a different linear system for each circuit. As such, the inner circuit will appear to have independent sources, allowing the singleton removal algorithm to work.

Since at least the 1980s, literature has included the two-level Newton algorithm, although mostly as it applied to circuit-device simulation. [46] and [47] provide a mathematical description, while [38] provides more information about the Xyce implementation.

12.3. Examples

```
THIS CIRCUIT IS THE TOP PART OF A TWO LEVEL EXAMPLE.
* compTop.cir - BSIM3 Transient Analysis

YEXT y1 DD1 SS1 externcode=xyce netlist=compInner.cir
Vdd DDorig 0 5.0
Vss SSorig 0 0.0

.options linsol type=klu
.options timeint abstol=1.0e-6  reltol=1.0e-3

* PARASITICS
l_Lwirevdd    DDorig Ny   .50n
l_Lwirevss    SSorig Nx   .50n
R_Rbw         Ny     DD1    50m
R_Rwi         Nx     SS1    50m

.tran 0.01ns 60ns
.print tran v(DD1) v(SS1) i(Vdd)

.END
```

Figure 12-2. Two-level top netlist example.

12.3.1. Explanation and Guidance

Figures 12-2 and 12-3 provide an example of a circuit that uses the two level algorithm. The top circuit (compTop.cir) (figure 12-2) invokes the inner circuit (compInner.cir) with the extern device, `y1`. To run this circuit, the user will only specify the top circuit on the command line:

```
THIS CIRCUIT IS THE INNER PART OF A TWO LEVEL EXAMPLE.
* compInner.cir - BSIM3 Transient Analysis
M1 Anot    A       DD1 DD1  PMOS w=3.6u l=1.2u
M2 Anot    A       SS1 SS1  NMOS w=1.8u l=1.2u
M3 Bnot    B       DD1 DD1  PMOS w=3.6u l=1.2u
M4 Bnot    B       SS1 SS1  NMOS w=1.8u l=1.2u
M5 AorBnot SS1     DD1 DD1  PMOS w=1.8u l=3.6u
M6 AorBnot B       1   SS1  NMOS w=1.8u l=1.2u
M7 1       Anot    SS1 SS1  NMOS w=1.8u l=1.2u
M8 Lnot    SS1     DD1 DD1  PMOS w=1.8u l=3.6u
M9 Lnot    Bnot    2   SS1  NMOS w=1.8u l=1.2u
M10 2      A       SS1 SS1  NMOS w=1.8u l=1.2u
M11 Qnot   SS1     DD1 DD1  PMOS w=3.6u l=3.6u
M12 Qnot   AorBnot 3   SS1  NMOS w=1.8u l=1.2u
M13 3      Lnot    SS1 SS1  NMOS w=1.8u l=1.2u
MQLO 8     Qnot    DD1 DD1  PMOS w=3.6u l=1.2u
MQL1 8     Qnot    SS1 SS1  NMOS w=1.8u l=1.2u
MLTO 9     Lnot    DD1 DD1  PMOS w=3.6u l=1.2u
MLT1 9     Lnot    SS1 SS1  NMOS w=1.8u l=1.2u
CQ Qnot 0 30f
CL Lnot 0 10f

Vconnect0000 DD1 0 0
Vconnect0001 SS1 0 0

Va A 0  pulse(0 5 10ns .1ns .1ns 15ns 30ns)
Vb B 0 0

.model nmos nmos (level=9)
.model pmos pmos (level=9)
.options linsol  type=klu
.options timeint abstol=1.0e-6 reltol=1.0e-3
.tran 0.01ns 60ns
.print tran v(a) v(b) 1.0+v(9) 1.0+v(8)

.END
```

Figure 12-3. Two-level inner netlist example.

```
Xyce compTop.cir <return>
```

The extern device (`YEXT y1` sits between the contents of compTop.cir and compInner.cir and is connected to two nodes in the top-level circuit, `DD1` and `SS1`. From the perspective of compTop.cir, the `YEXT y1` device looks like a nonlinear two-terminal resistor, which is the equivalent of the entire inner circuit.

In the inner circuit, Xyce applies nodes `DD1` and `SS1` though the independent sources `Vconnect0000` and `Vconnect0001`. By convention, the inner circuit must contain an independent voltage source for each node to which the EXT device is connected. The default naming convention requires that these sources be named `vconnectxxxx`, with xxxx being a four-digit integer starting at 0000.

NOTE: The `.tran` statement on the inner circuit must match the `.tran` statement on the top circuit. The same is true for the `.DC` analysis statements. Also, as both circuit files have their own `.print` statements, both will produce `*.prn` output files.

The coupling between the top and inner layers requires extra linear solves, so when using this algorithm the code will run more slowly. In general, one can expect a factor-of-two slowdown, for circuits that can be run either as conventional or two-level simulations. So, in practice this algorithm should only be applied when it is really needed (i.e., when conventional simulations fail).

Finally, when using this two-level method, one must take particular care with file names. In practice, a Xyce user may frequently change netlist file names to reflect new details about the run. When this happens, the name of the netlist invoked on the `YEXT y1` line must be changed. Failure to do so may result in using the wrong file for the inner simulation.

12.4. Restart

Restart works with the two-level algorithm. However, as the two-level algorithm involves two separate netlist input files, a two-level restart requires a separate restart file for each phase of the problem. So, the two files (e.g., `compTop.cir` and `compInner.cir`) require `.options restart` statements, and the statements in the two files must be consistent with each other. *The user must enforce this*, because the Xyce code does *not* check consistency between the top and inner file "`.options restart`" statements.

This page intentionally left blank.

Chapter Overview

This chapter includes the following sections:

13.1. Initial Conditions Overview

Xyce provides several different options for users to set an initial condition. Reasons for setting initial conditions include, but are not limited to:

- Improving the robustness of the DCOP solution
- Optimizing performance by reusing a DCOP solution of a previous run to start new transient runs
- Setting an initial state for a digital circuit
- Initiating an oscillator circuit.

As noted, setting initial conditions can be particularly useful for multistate digital circuits. Figure 13-1 provides an example result demonstrating how initial conditions can be used to set the state of a digital circuit. In this case, obtaining the state purely through transient simulation can be time-consuming and often is not practical..

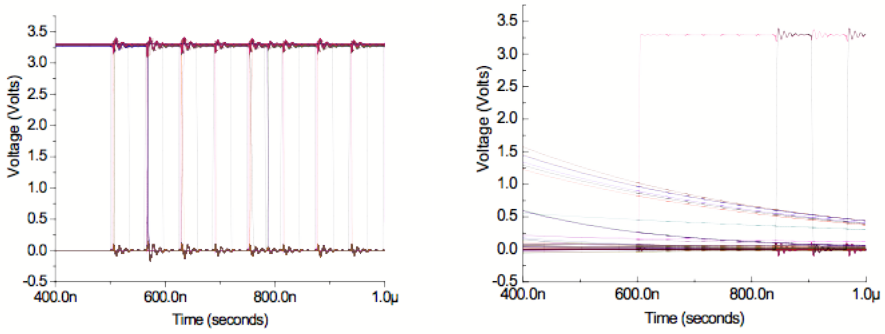

Figure 13-1. Example result with (left) and without (right) an IC on the output. NOTE: The preset example, with an IC, starts in the initial state directly out of the DCOP calculation, while the non-preset example requires a long transient to equilibrate.

13.2. Device Level IC= Specification

Many devices in Xyce support setting initial junction voltage conditions on the device instance line with the IC= keyword. This is frequently used to set the state of digital circuits. Figure 13-2 presents a simple inverter example demonstrating the use of IC= on a BSIMSOI device. In this example, the two initial conditions are specified as a vector, which is the preferred syntax. The initial conditions can also be specified separately (e.g., IC1=2 IC2=0).

While many circuit simulators have a similar IC= capability, Xyce implementation differs in some important respects. For any device with an IC= statement, Xyce enforces the junction drop during the operating point computation by inserting a voltage source in parallel with the device junction. Xyce then applies the parallel voltage source through the DCOP calculation, and then removes it prior to the beginning of the transient. This strongly enforces the requested junction drop, meaning that if the DCOP converges, the requested voltage drop will be in the solution and the entire circuit solution will be consistent with that voltage drop. Many other circuit codes apply IC= as a weaker constraint, with the intent of improving DCOP calculation robustness.

IC= can be applied to the following devices: BSIM3, BSIM4, BSIMSOI, Capacitor, Inductor and Digital Behavioral Devices (U and Y).

In the case of the capacitor, initial conditions specified via an IC= parameter are also applied as an initial condition for transients in the case that the user has used a NOOP or UIC option on a transient line, bypassing the operating point calculation. In this case only, the initial condition is not enforced by the addition of a voltage source, but rather by a weaker constraint similar to the method used by other SPICE-type simulators.

```
MOS LEVEL=10 INVERTER WITH IC=

.subckt INV IN OUT VDD GND
MN1 OUT IN GND GND GND NMOS w=4u  l=0.15u IC=2,0
MP1 OUT IN VDD GND VDD PMOS w=10u l=0.15u
.ends

.tran 20ns 30us
.print tran v(vout) v(in)+1.0 v(1)

VDDdev  VDD 0 2V
RIN IN  1 1K
VIN1  1 0  2V PULSE (2V 0V 1.5us 5ns 5ns 1.5us 3.01us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
XINV1 IN VOUT VDD 0 INV
.MODEL NMOS NMOS ( LEVEL = 10 )
.MODEL PMOS PMOS ( LEVEL = 10 )

.END
```

Figure 13-2. Example netlist with device-level IC=.

13.3. .IC and .DCVOLT Initial Condition Statements

.IC and .DCVOLT are equivalent methods for specifying initial conditions. How Xyce applies them, however, depends on whether the UIC parameter, discussed in a following subsection, is present on the .TRAN line. If UIC is not specified, then Xyce applies the conditions specified by .IC and .DCVOLT statements throughout the DCOP phase, ensuring the specified values will be the solved values at the end of the DCOP calculation. Xyce allows unspecified variables to find their computed values, consistent with the imposed voltages.

```
RC circuit
.ic v(1)=1.0
c1 1 0 1uF
R1 1 2 1K
v1 2 0 0V
.print tran v(1)
.tran 0 5ms
.options timeint reltol=1e-6 abstol=1e-6
.end
```

Figure 13-3. Example netlist with .IC. NOTE: Without the .IC statement, the capacitor is not given an initial charge, and the transient signals are flat. With the .IC statement, it has an initial charge, which then decays in transient. Without the .IC statement, the capacitor is not given an initial charge, and the signals in transient are all flat. With the .IC statement, it has an initial change which then decays in transient.

If UIC is specified on the .TRAN line, then Xyce skips the DCOP calculation altogether, and uses the values specified on .IC and .DCVOLT lines as the initial values for the transient calculation. Unspecified values are set to zero.

For the UIC and non-UIC cases, Xyce ignores specified values that do not correspond to existing circuit variables.

Finally, the .IC capability can only set voltage values, not current values.

13.3.1. Syntax

```
.IC V(node1) = val1 <V(node2) = val2> ...
.DCVOLT V(node1) = val1 <V(node2) = val2> ...
```

where: *val1, val2, ...* specify nodal voltages and *node1, node2, ...* specify node numbers.

13.3.2. Example

```
.IC V(1) = 2.0  V(A) = 4.5
.DCVOLT  1 2.0 A 4.5
```

Fig. 13-3 provides a more complete example (showing a full netlist).

13.4. .NODESET Initial Condition Statements

.NODESET is similar to .IC, except that Xyce enforces the specified conditions less strongly. For .NODESET simulations, Xyce performs *two* nonlinear solves for the DCOP condition. For the first solve, Xyce enforces the .NODESET values throughout the solve, similar to .IC. For the second solve, Xyce uses the result of the first solve as an initial guess, and allows all the values to float and eventually obtain their unconstrained, self-consistent values. As such, the computed values will not necessarily match the specified values.

If used with UIC or NOOP 13.6, .NODESET behaves the same as .IC and .DCVOLT.

13.4.1. Syntax

```
.NODESET V(node1) = val1 <V(node2) = val2> ...
.NODESET node1 val1 <node2 val2>
```

where: *val1, val2, ...* specify nodal voltages and *node1, node2, ...* specify node numbers.

13.4.2. Example

```
.NODESET V(1) = 2.0  V(A) = 4.5
.NODESET  1 2.0 A 4.5
```

13.5. .SAVE Statements

Xyce stores operating point information using .SAVE statements, and can then reuse that information to start subsequent transient simulations. Using .SAVE results in solution data being stored in a text file, comprised of .NODESET or .IC statements. This file can be applied to other simulations using .INCLUDE.

The .SAVE syntax is as follows:

```
.SAVE [TYPE=<IC|NODESET>] [FILE=<filename>] [LEVEL=<all|none>]
+ [TIME=<save_time>]
```

where:

The **TYPE** can be set to NODESET or IC. By default, it will be NODESET.

The **FILE** is the user-specified output file name for the output file. If this is not specified, Xyce uses *netlist.cir*.ic.

The **LEVEL** is an HSPICE compatibility parameter. Xyce supports ALL and NONE. If NONE is specified, then no save file is created. The default **LEVEL** is ALL.

TIME is an HSPICE compatibility parameter. This is unsupported in Xyce. Xyce outputs the save file only at time=0.0.

13.6. UIC and NOOP

As noted earlier, the UIC key word on the TRAN line will disable the DCOP calculation, and result in Xyce immediately going to transient. If the user specifies .IC or .NODESET, then the transient calculation will use the specified initial values as the initial starting point. The NOOP keyword works exactly the same way as UIC.

```
pierce oscillator
c1 1 0 100e-12
c2 3 0 100e-12
c3 2 3 99.5e-15
c4 1 3 25e-12
l1 2 4 2.55e-3
r1 1 3 1e5
r2 3 5 2.2e3
r3 1 4 6.4
v1 5 0 12
Q1 3 1 0 NBJT
.MODEL NBJT NPN (BF=100)
.print tran  v(2) v(3)

.tran 1ns 1us  UIC
.ic v(2)=-10000.0 v(5)=12.0
```

Figure 13-4. Example netlist with UIC. NOTE: This circuit is a pierce oscillator, which only oscillates if the operating point calculation is skipped. If the .IC statement is not included, the oscillator will take a long time to achieve its steady-state amplitude. By including the .IC statement, the amplitude of node 2 is preset to a value close to its final steady-state amplitude. The transient in this example only runs for 10 cycles as a demonstration. In general, the time scales for this oscillator are much longer and require millions of cycles.

13.6.1. Example

```
.tran 1ns 1us  UIC
.tran 1ns 1us  NOOP
```

Some circuits, particularly oscillator circuits, will only function properly if the operating point calculation is skipped, as they need an inconsistent initial state to oscillate. Figure 13-4 presents a Pierce oscillator example.

This page intentionally left blank.

14. WORKING WITH .PREPROCESS COMMANDS

Chapter Overview

This chapter includes the following sections:

- Section 14.1, *Introduction*
- Section 14.2, *Ground Synonym Replacement*
- Section 14.3, *Removal of Unused Components*
- Section 14.4, *Adding Resistors to Dangling Nodes*

14.1. Introduction

In an effort to make Xyce more compatible with other commercial circuit simulators (e.g., HSPICE), some optional tools have been added to increase the netlist processing capabilities of Xyce. These options, which occur toward the beginning of a simulation, have been incorporated not only to make Xyce more compatible with different (i.e. non-Xyce) netlist syntaxes, but also to help detect and remove certain singular netlist configurations that can often cause a Xyce simulation to fail. Because all of the commands described in this section occur as a precursory step to setting up a Xyce simulation, they are all invoked in a netlist file via the keyword `.PREPROCESS`. This chapter describes each of the different functionalities that can be invoked via a `.PREPROCESS` statement in detail and provides examples to illustrate their use.

14.2. Ground Synonym Replacement

In certain versions of SPICE, keywords such as `GROUND`, `GND`, and `GND!` can be used as node names in a netlist file to represent the ground node of a circuit. Xyce, however, only recognizes node `0` as an official name for ground. Hence, if any of the prior node names is encountered in a netlist file, Xyce will treat these as different nodes from ground. To illustrate this point, consider the netlist of figure 14-1. When the node `Gnd` is encountered in the definition of resistor `R3`, Xyce instantiates this as a new node. The schematic diagram corresponding to this netlist (figure 14-2) shows that the resistor `R3` is "floating" between node `2` and a node with only a single device connection, node `Gnd`. When Xyce executes the netlist of figure 14-1, the voltage `V(2)` will evaluate to 0.5V.

```
Circuit with "floating" resistor R3

V1 1 0 1
R1 1 2 1
R2 2 0 1
R3 2 Gnd 1

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14-1. Example netlist where `Gnd` is treated as being *different* from node `0`.

If one would rather treat `Gnd` the same as node `0` in the above example, use the figure 14-3 netlist instead. When the statement `.PREPROCESS REPLACEGROUND TRUE` is present in a netlist, Xyce will treat any nodes named `GND`, `GND!`, `GROUND`, or any capital/lowercase variant of these keywords (e.g., `gROunD`) as synonyms for node `0`. Hence, according to Xyce, the figure 14-3 netlist corresponds to figure 14-4 schematic diagram, and the voltage `V(2)` will evaluate to 0.33V.

NOTE: Only one `.PREPROCESS REPLACEGROUND` statement is allowed per netlist file. This constraint prevents the user from setting `REPLACEGROUND` to `TRUE` on one line and then to `FALSE` on another line. Also, there is no way to differentiate between different keywords. So, for example, it is not possible to treat `GROUND` as a synonym for node `0` while allowing `GND` to represent an independent node). If `REPLACEGROUND` is set to `TRUE`, Xyce will treat *both* of these keywords as node `0`.

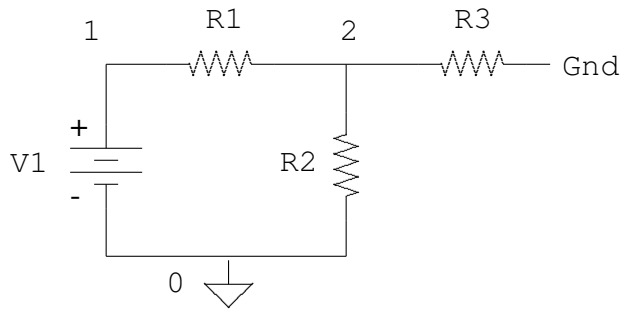

Figure 14-2. Circuit diagram corresponding to the netlist of figure 14-1 where node `Gnd` is treated as being *different* from node `0`.

```
Circuit where resistor R3 does *not* float

V1 1 0 1
R1 1 2 1
R2 2 0 1
R3 2 Gnd 1

.PREPROCESS REPLACEGROUND TRUE

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14-3. Example netlist where `Gnd` is treated as a synonym for node `0`.

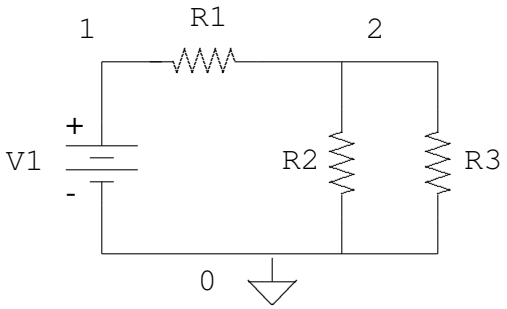

Figure 14-4. Circuit diagram corresponding to figure 14-3 where node `Gnd` is treated as a synonym for node `0`.

14.3. Removal of Unused Components

Consider a slight variant of the circuit in figure 14-3 with the netlist given in figure 14-5. Here, the resistor R3 is connected in a peculiar configuration: both terminals of the resistor are tied to the same circuit node, as is illustrated in figure 14-6. Clearly, the presence of this resistor has no effect on the other voltages and currents in the circuit since, by the very nature of its configuration, it has no voltage across it and, hence, does not draw any current. Therefore, in some sense, the component can be considered as "unused." The presence of a resistor such as R3 is rarely or never introduced by design, rather the presence of such components is the result of either human or automated error during netlist creation.

```
Circuit with an unused resistor R3

V1 1 0 1
R1 1 2 1
R2 2 0 1
R3 2 2 1

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14-5. Netlist with a resistor R3 whose device terminals are both the same node (node 2).

While the presence of the resistor R3 in figure 14-3 does not change the behavior of the circuit, it adds an additional component to the netlist Xyce must include when solving for the voltages and currents in the circuit. If the number of such components in a given netlist is large, it is potentially desirable to remove them from the netlist to ease the burden on Xyce's solver engines. This, in turn, can help to avoid possible convergence issues. For example, even though the netlist in figure 14-5 will run properly in Xyce, the netlist of figure 14-7 will abort. The voltage source V2 attempts to place a 1V difference between its two device terminals; however, as both nodes of the voltage source are the same, the voltage source is effectively shorted.

Xyce includes the following command to prevent similar situations:

```
.PREPROCESS REMOVEUNUSED <component list>
```

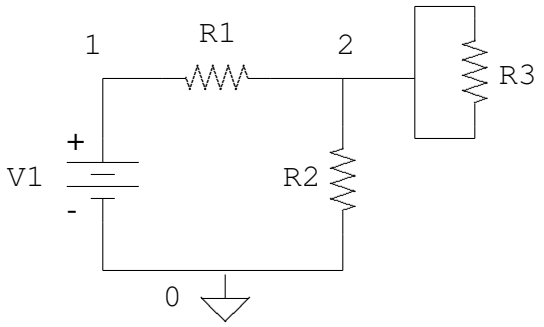

Figure 14-6. Circuit of figure 14-5 containing a resistor R3 whose terminals are tied to the same node (node 2).

```
Circuit with improperly connected voltage source V2

V1 1 0 1
R1 1 2 1
R2 2 0 1
V2 2 2 1

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14-7. Circuit with an improperly connected voltage source V2.

where <component list> is a list of device types separated by commas. For each device type specified in the list, Xyce checks for instances of that device type for which all of the device's terminals are connected to the same node. If such a device is found, Xyce removes that device from the netlist. For instance, when executing the netlist of figure 14-8, Xyce will seek out such devices and remove them from the netlist. This causes the resistor R3 to be removed from the netlist. Figure 14-9 presents the schematic of the resulting Xyce-simulated circuit. NOTE: The presence of "C" in the REMOVEUNUSED statement does not cause Xyce to abort even though there are no capacitors in the netlist. Also, as in the case of a REPLACEGROUND statement, only one .PREPROCESS REMOVEUNUSED line may be present in a netlist, or Xyce will abort.

Table 14-1 lists devices that can be removed via a REMOVEUNUSED statement. In the case of MOSFETs and BJTs, three device terminals must be the same (the gate, source, and drain in the case of a MOSFET; the base, collector, and emitter in the case of a BJT) to remove either device from the netlist.

```
Circuit with improperly connected voltage source V2

V1 1 0 1
R1 1 2 1
R2 2 0 1
R3 2 2 1

.PREPROCESS REMOVEUNUSED R,C

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14-8. Circuit with an "unused" resistor R3 removed from the netlist.

Table 14-1. List of keywords and device types which can be used in a .PREPROCESS REMOVEUNUSED statement.

Keyword	Device Type
C	Capacitor
D	Diode

Table 14-1. List of keywords and device types which can be used in a `.PREPROCESS REMOVEUNUSED` statement.

Keyword	Device Type
I	Independent Current Source
L	Inductor
M	MOSFET
Q	BJT
R	Resistor
V	Independent Voltage Source

14.4. Adding Resistors to Dangling Nodes

Consider the netlist of figure 14-10 and the corresponding schematic of figure 14-11. Nodes 3 and 4 of the netlist are what we will henceforth refer to as *dangling nodes*. We say that node 4 dangles because it is only connected to the terminal of a single device, while we say that node 3 dangles because it has no DC path to ground. The first of these situations—connection to a single device terminal only—can arise, for example, in a netlist which contains nodes representing output pins that are not connected to a load device. For instance, the resistance R2 in figure 14-10 could represent the resistance of an output pin of a package that is meant to drive resistive loads. Hence, an actual physical implementation of the circuit of figure 14-11 would normally include a resistor between node 4 and ground, but, in creating the netlist, the presence of such an output load has been (either intentionally or unintentionally) left out.

The second situation—where a node has no DC path to ground—is sometimes an effect that is purposely incorporated into a design (e.g., the design of switched capacitor integrators (e.g., see [48], chapter 10), but oftentimes it is also the result of some form of error in the process of creating the netlist. For instance, when graphical user interfaces (GUIs) are used to create circuit schematics that are then translated into netlists via software, one very common unintentional error is to fail to connect two nodes that are intended to be connected. To illustrate this point, consider the schematic of figure 14-12. The schematic seems to indicate that the lower terminal of resistor R2 should be connected to node 3. This is not the case as there is a small gap between node 3 and the line intended to connect node 3 to the resistor. Such an error can often go unnoticed when creating a schematic of the netlist in a GUI. Thus, when the schematic is translated into a netlist file, the resulting netlist would *not* connect the resistor to node 3 and would instead create a new node at the bottom of the resistor, resulting in the circuit depicted in figure 14-11.

While neither of the previous situations is necessarily threatening (Xyce will run the figure 14-10 netlist successfully to completion), there are times when it is desirable to somehow make a dangling node *not* dangle. For instance, returning to the example in which the resistor R2 represents the resistance of an output pin, one may want to simulate the circuit when a 1K load is attached between node 4 and ground in figure 14-11. In the case where a node has no DC path to ground, the situation is slightly more dangerous if, for instance, the node in question is also connected to a high-gain device such as the gate of a MOSFET. As the DC gate bias has a great impact on the DC current traveling through the drain and source of the transistor, not having a well-defined DC gate voltage can greatly degrade the simulated performance of the circuit.

In both prior examples, the only true way to “fix” each of these issues is to find all dangling nodes in a particular netlist file and augment the netlist at/near these nodes to obtain the desired behavior. If, however, the number of components in a circuit is very large (say on the order of hundreds of thousands of

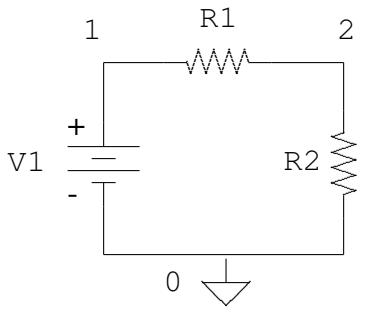

Figure 14-9. Circuit of figure 14-8 where resistor R3 has been removed via the `.PREPROCESS REMOVEUNUSED` statement.

```
Circuit with two dangling nodes, nodes 3 and 4

V1 1 0 1
R1 1 2 1
C1 2 3 1
C2 3 0 1
R2 2 4 1

.DC V1 0 1 0.1
.PRINT DC V(2)
.END
```

Figure 14-10. Netlist of circuit with two dangling nodes, nodes `3` and `4`.

components), manually augmenting the netlist file for each dangling node becomes a practical impossibility if the number of such nodes is large.

Hence, it is desirable for Xyce to be capabable of automatically augmenting netlist files so as to help remove dangling nodes from a given netlist. The command `.PREPROCESS ADDRESISTORS` is designed to do just this. Assuming the netlist of figure 14-13 is stored in the file `filename`, the `.PREPROCESS ADDRESISTORS` statements will cause Xyce to create a new netlist file called `filename_xyce.cir` (depicted in figure 14-14). The line `.PREPROCESS ADDRESISTORS NODCPATH 1G` instructs Xyce to create a copy of the netlist file containing a set of resistors of value 1 GΩ that are connected between ground and the nodes that did not have a DC path to ground. Similarly, the line `.PREPROCESS ADDRESISTORS ONETERMINAL 1M` instructs Xyce to add to the same netlist file a set of resistors of value 1 MΩ that are connected between ground and devices that are connected to only one terminal. The resistor `RNODCPATH1` in figure 14-14 achieves the first of these goals while `RONETERM1` achieves the second. Figure 14-15 shows a schematic of the resulting circuit represented by the netlist in figure 14-14.

Some general comments regarding the use of `.PREPROCESS ADDRESISTOR` statements include:

- Xyce does not terminate immediately after the netlist file is created. In other words, if Xyce is run on the `filename` of figure 14-13 netlist, it will attempt to execute this netlist as given (i.e., it tries to simulate the circuit of figure 14-11) and generates the file `filename_xyce.cir` as a byproduct. It is important to point out that the resistors that are added at the bottom of the netlist file `filename_xyce.cir` do **not** get added to the original netlist when Xyce is running on the file

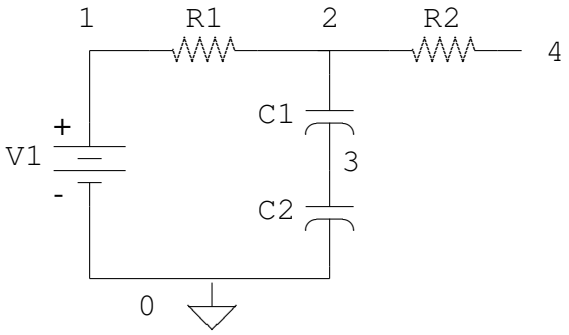

Figure 14-11. Schematic of netlist in figure 14-10.

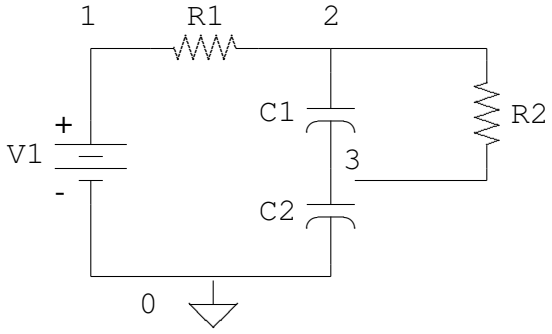

Figure 14-12. Schematic of a circuit with an incomplete connection between the resistor R2 and node 3.

`filename`. If one wishes to simulate Xyce with these resistors in place, one must run Xyce on `filename_xyce.cir` explicitly.

- The naming convention for resistors which connect to ground nodes which do not have a DC path to ground is `RNODCPATH<i>`, where `i` is an integer greater than 0; the naming convention is similar for nodes which are connected to only one device terminal (i.e., of the form `RONETERM<i>`). Xyce will not change this naming convention if a resistor with one of the above names already exists in the netlist.

  Hence, if a resistor named `RNODCPATH1` exists in netlist file `filename`, and Xyce detects there is a node in this netlist file that has no DC path to ground, Xyce will add *another* resistor with name `RNODCPATH1` to the netlist file `filename_xyce.cir` (assuming that either `.PREPROCESS ADDRESISTORS NODCPATH` or `.PREPROCESS ADDRESISTORS ONETERMINAL` are present in `filename`). If Xyce is subsequently run on `filename_xyce.cir`, it will exit in error due to the presence of two resistors with the same name.

- Commands `.PREPROCESS ADDRESISTORS NODCPATH` and `.PREPROCESS ADDRESISTORS ONETERMINAL` do **not** have to be simultaneously present in a netlist file. The presence of either command will generate a file `filename_xyce.cir`, and the presence of both will not generate two separate files. As with other `.PREPROCESS` commands, however, a netlist file is allowed to contain only one `NODCPATH` and one `ONETERMINAL` command each. If multiple `NODCPATH` and/or `ONETERMINAL` lines are found in a single netlist file, Xyce will exit in error.

- It is possible that a single node can have no DC path to ground *and* be connected to only one device terminal. If a `NODCPATH` and `ONETERMINAL` command are present in a given netlist file, **only** the

```
Circuit with two dangling nodes, nodes 3 and 4

V1 1 0 1
R1 1 2 1
C1 2 3 1
C2 3 0 1
R2 2 4 1

.PREPROCESS ADDRESISTORS NODCPATH 1G
.PREPROCESS ADDRESISTORS ONETERMINAL 1M

.DC V1 0 1 0.1
.PRINT DC V(2)
.END
```

Figure 14-13. Netlist of circuit with two dangling nodes, nodes 3 and 4, with .PREPROCESS ADDRESISTORS statements.

resistor corresponding to the ONETERMINAL command is added to the netlist file filename_xyce.cir and the resistor corresponding to the NODCPATH command is omitted. If a NODCPATH command is present but a ONETERMINAL command is not, then Xyce will add a resistor corresponding to the NODCPATH command to the netlist, as usual.

- In generating the file filename_xyce.cir, the original .PREPROCESS ADDRESISTOR statements are commented out with a warning message. This is to prevent Xyce from creating the file filename_xyce.cir_xyce.cir when the file filename_xyce.cir is run.

  NOTE: This feature avoids generating redundant files. While filename_xyce.cir_xyce.cir would be slightly different from filename_xyce.cir (e.g., a different date and time stamp), both files would functionally implement the same netlist.

```
XYCE-generated Netlist file copy:  TIME='07:32:31 AM'
* DATE='Dec 19, 2007'
*Original Netlist Title:

*Circuit with two dangling nodes, nodes 3 and 4.


V1 1 0 1
R1 1 2 1
C1 2 3 1
C2 3 0 1
R2 2 4 1

*.PREPROCESS ADDRESISTORS NODCPATH 1G
*Xyce:  ".PREPROCESS ADDRESISTORS" statement
* automatically commented out in netlist copy.
*.PREPROCESS ADDRESISTORS ONETERMINAL 1M
*Xyce:  ".PREPROCESS ADDRESISTORS" statement
* automatically commented out in netlist copy.

.DC V1 0 1 0.1
.PRINT DC V(2)


*XYCE-GENERATED OUTPUT:  Adding resistors between ground
* and nodes connected to only 1 device terminal:

RONETERM1 4 0 1M


*XYCE-GENERATED OUTPUT:  Adding resistors between ground
* and nodes with no DC path to ground:

RNODCPATH1 3 0 1G

.END
```

Figure 14-14. Output file `filename_xyce.cir` which results from the `.PREPROCESS ADDRESISTOR` statements for the netlist of figure 14-12 (with assumed file name `filename`).

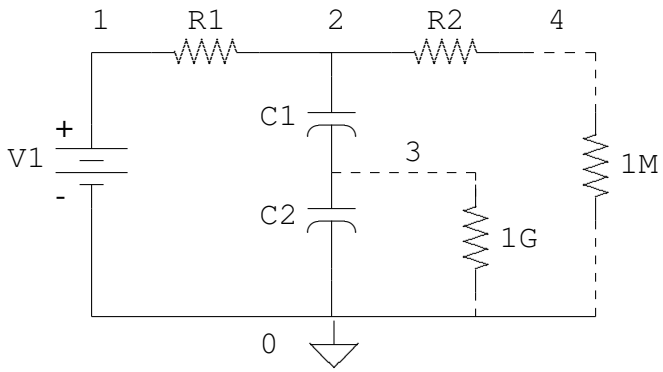

Figure 14-15. Schematic corresponding to the Xyce-generated netlist of figure 14-14.

This page intentionally left blank.

15. TCAD (PDE DEVICE) SIMULATION WITH XYCE

Chapter Overview

This chapter provides guidance for using the mesh-based device simulation capability of Xyce. It includes the following sections:

- Section 15.1, *Introduction*
- Section 15.2, *One-Dimensional Example*
- Section 15.3, *Two-Dimensional Example*
- Section 15.4, *Doping Profile*
- Section 15.5, *Electrodes*
- Section 15.6, *Meshing*
- Section 15.8, *Mobility Models*
- Section 15.9, *Bulk Materials*
- Section 15.10, *Output and Visualization*

15.1. Introduction

This chapter describes how to use the mesh-based device simulation functionality of Xyce, which is based on the solution a coupled set of partial differential equations (PDEs), discretized on a mesh. Such devices are often referred to as Technology Computer-Aided Design (TCAD) devices. While the rest of Xyce is intended to be similar to analog circuit simulators such as SPICE, the TCAD device capability is intended to be similar to commercial device simulators, such as PISCES [49] and DaVinci [50].

Xyce offers two different TCAD devices—a one-dimensional device and a two-dimensional device—and enables both to be invoked in the same way as a conventional lumped parameter circuit device. Generally, this capability is intended for very detailed simulation of semiconductor devices, such as diodes, bipolar transistors, and MOSFETs. As the Xyce TCAD devices can be invoked from the netlist, they can be embedded in a circuit as part of a mixed-mode simulation.

15.1.1. Equations

Kramer [51] and Selberherr [52], among others, describe device simulation equations. The most common formulation and the one used in Xyce, is the drift-diffusion (DD) formulation, which consists of three coupled PDEs (a single Poisson equation for electrostatic potential and two continuity equations; one each for electrons and holes).

15.1.1.1. Poisson equation

The electrostatic potential ϕ satisfies Poisson's equation:

$$-\nabla \cdot (\varepsilon \nabla \phi(x)) = \rho(x) \tag{15.1}$$

where ρ is the charge density and ε is the permittivity of the material. For semiconductor devices, local carrier densities and local doping determine charge density;

$$\rho(x) = q(p(x) - n(x) + C(x)) \tag{15.2}$$

Here, $p(x)$ is the spatially dependent concentration of holes; $n(x)$, the concentration of electrons; and q, the magnitude of the charge on an electron. $C(x)$ is the total doping concentration, which can also be represented as $C(x) = N_D^+(x) - N_A^-(x)$, where N_D^+ the concentration of positively ionized donors, N_A^- the concentration of negatively ionized acceptors.

15.1.1.2. Species continuity equations

Continuity equations relate the convective derivative of the species concentrations to the creation and destruction of particles ("recombination/generation").

$$\frac{\partial n(x)}{\partial t} + \nabla \cdot \Gamma_n = -R(x) \tag{15.3}$$

$$\frac{\partial p(x)}{\partial t} + \nabla \cdot \Gamma_p = -R(x) \tag{15.4}$$

Here n is the electron concentration and p is the hole concentration. R is the recombination rate for both species. Γ_n and Γ_p are particle fluxes for electrons and holes, respectively. R is the recombination rate for both species, and the right hand sides are equal since creation and destruction of carriers occurs in pairs. The quantities Γ_n and Γ_p are electron and hole fluxes, and are determined from the following expressions:

$$\Gamma_n = n(x)\mu_n E(x) + D_n \nabla n(x) \tag{15.5}$$

$$\Gamma_p = p(x)\mu_p E(x) + D_p \nabla p(x) \tag{15.6}$$

μ_n, μ_p are mobilities for electrons and holes, and D_n, D_p are diffusion constants. $E(x)$ is the electric field, which is given by the gradient of the potential, or $-\partial\phi/\partial x$.

15.1.2. Discretization

Xyce uses a box-integration discretization, with the Scharfetter-Gummel method, to model the flux of charged species. For a more-detailed description of this method, refer to [51] [52] [53].

15.2. One Dimensional Example

While one-dimensional device simulation has limits on its usefulness, the simulation much faster than 2D, and it can provide reasonable physical predictions in many situations. Two-terminal diodes are a good candidate for one dimensional simulation, because they allow for assumptions that simplify the specification and shorten the parameter list of the device.

Figure 15-1 provides an example netlist for a simulation of a one-dimensional diode, while figure 15-2 shows its corresponding schematic. This regulator circuit is based on the principle that connecting one or more diodes in series with a resistor and a power supply will produce a relatively constant voltage. The input voltage (node 2) is a sine wave, with a frequency of 50 Hz and an amplitude of 1 V. The expected output (node 3) signal should be (mostly) flat.

15.2.1. Netlist Explanation

The model line for PDE devices serves only to set the level. The default level is 1, for a one-dimensional device. Setting `level=2` will invoke two-dimensional devices. In this example, the level is not explicitly set, and so Xyce uses the default value which is 1.

The instance line is where most of the specific parameters are set for a TCAD device. In this example, the line appears as:

```
YPDE Z1 3 4 DIODE na=1.0e17 nd=1.0e17 graded=1 l=5.0e-4 nx=101
```

Doping parameters `na` and `nd` represent the majority carrier doping levels on the N- and the P-sides of the junction, respectively. `graded=1` is also a doping parameter, and specifies that the junction is a graded junction, rather than an abrupt step-function junction. `l=5.0e-4` specifies the length of the device, in cm. `nx=101` specifies that there are 101 mesh points, including the two endpoints. For the one-dimensional device, the mesh is always uniform, so the size of each mesh cell, Δx will be:

$$\Delta x = \frac{l}{nx-1} = \frac{\text{5.0e-4 cm}}{100} = \text{5.0e-6 cm} \tag{15.7}$$

```
PDE Diode Regulator Circuit
VP 1 0 PULSE(0 5 0.0 2.0e-2 0.0 1.0e+20 1.2e+20)
VF 2 1 SIN(0 1 50 2.0e-2)
VT1 4 0 0V
R1 2 3 1k

 * TCAD/PDE Device
YPDE Z1 3 4 DIODE na=1.0e17 nd=1.0e17 graded=1
+ l=5.0e-4 nx=101
 .MODEL DIODE  ZOD

.TRAN 1.0e-3 12.0e-2
.print TRAN format=tecplot
+ v(1) v(2) v(3) v(4) I(VF) I(VT1)

.options NONLIN maxstep=100 maxsearchstep=3
+ searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
.END
```

Figure 15-1. Voltage regulator circuit, using a one-dimensional TCAD diode. Figure 15-3 illustrates the result of this netlist. The PDE device instance line is in red, and the PDE device model line is in blue.

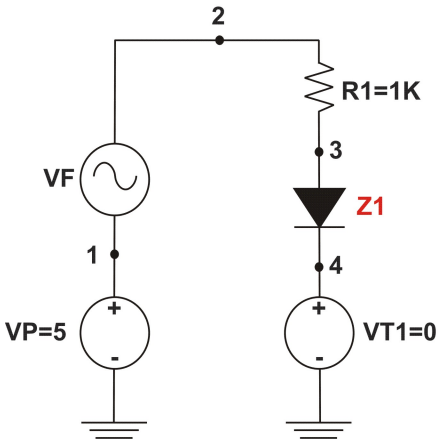

Figure 15-2. Voltage regulator schematic The diode, Z1, is the PDE device in this example.

The mesh points $i = 0 - 100$ will have the following locations, x_i:

$$\begin{aligned} x_i &= i\Delta x \\ x_o &= 0.0 \text{ cm} \\ x_1 &= 5.0\text{e-}6 \text{ cm} \\ x_2 &= 10.0\text{e-}6 \text{ cm} \\ &\vdots \\ x_{100} &= 5.0\text{e-}4 \text{ cm} \end{aligned}$$

15.2.2. Boundary Conditions and Doping Profile

The cited netlist example relies mostly on default parameters; therefore, it specifies nothing about electrodes, or boundary conditions, and has a minimal doping specification. A one-dimensional device can have only two electrodes connected to the circuit. The electrodes are at opposite ends of the domain, one at the first mesh point (`x=0.0 cm`, `i=0`) and the other at the opposite end of the domain, at the last mesh point (`x=5.0e-4 cm`, `i=101`).

The electrode associated with the first mesh point (`x=0.0 cm`) is connected to the *second* circuit node on the instance line, while the electrode associated with the last mesh point (`x=l`) is connected to the *first* circuit node on the instance line. For the doping used in this example, the junction is in the exact center of the device (`x=l/2`), and the n-side is the region defined by `x<l/2`, and the p-side is the region defined by `x>l/2`. This default doping, along with the electrode-circuit connectivity, results in a one-dimensional device that behaves like a traditional SPICE-style diode. For a complete discussion of how to specify a doping profile see section 15.4.1. For a complete discussion of how to specify electrodes (including boundary conditions), see section 15.5.

15.2.3. Results

Figure 15-3 shows the transient behavior of this circuit. The voltage drop across the diode (V(3)) is nearly the same for a wide range of currents, and is nearly constant. The voltage drop (V(2)-V(3)) across the series resistor, `R1`, is much more sensitive, and so most of the voltage variation of the input sine wave is accounted for by `R1`.

15.3. Two-Dimensional Example

Figure 15-4 presents an example netlist for a simulation of a two-dimensional bipolar transistor. As before, the PDE device instance line is in red, while the PDE device model line is in blue. In this case, note that the model line specifies the level, which is set to 2. This is required for the two-dimensional device. This particular example is a DC sweep of a bipolar transistor device. Figure 15-5 presents a schematic illustrating this circuit.

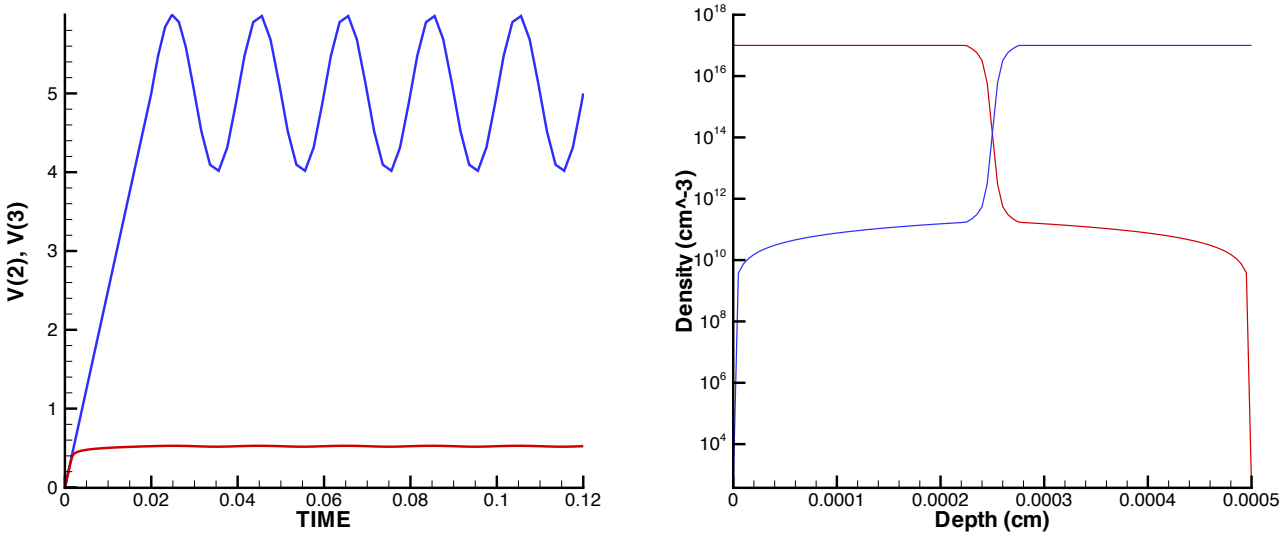

Figure 15-3. Results for voltage regulator. In the left plot, the transient output is shown, in which the input voltage is blue and the output voltage is red. In the right plot, the initial carrier densities are shown, with the electron density in red and the hole density in blue.

15.3.1. Netlist Explanation

The two-dimensional device can have 2 to 4 electrodes. In this example there are three; nodes 5, 3, and 7, corresponding to the three names on the "node" line, which appears as:

```
+ node = {name = collector, base, emitter}
```

This line specifies that node 5 is connected to an electrode named "collector," node 3 is connected to an electrode named "base," and node 7 is connected to an electrode named "emitter". Although this example only contains the electrode names, the "node" specification can contain a lot of information. Section 15.5 provides a full explanation of the electrode parameters.

The next line contains parameters concerned with plotting the results, and appears as follows:

```
+ tecplotlevel=2 txtdatalevel=1
```

These are not related to the output specified by `.PRINT`, which outputs circuit data. The `tecplotlevel` command enables files to be output readable by Tecplot, which can then be used to create contour plots of quantities such as the electron density, electrostatic potential and the doping profile. Figures 15-6 and 15-7 contain examples of Tecplot-generated contour plots, which were generated from the results of this example.

The `txtdatalevel` command enables a text file with volume averaged information to be output to a file. Xyce will update both of these output files at each time step or DC sweep step.

The next line, `mobmodel=arora`, specifies which mobility model to use. Section 15.8 provides for more detail on available mobility models.

The last two lines, specify the mesh of the device, and are given by:

```
+ l=2.0e-3 w=1.0e-3
+ nx=30 ny=15
```

```
Two-Dimensional Example
VPOS   1 0 DC 5V
VBB    6 0 DC -2V
RE     1 2 2K
RB     3 4 190K

 YPDE BJT 5 3 7 PDEBJT meshfile=internal.msh
+ node = {name = collector, base, emitter}
+ tecplotlevel=2 txtdatalevel=1
+ mobmodel=arora
+ l=2.0e-3  w=1.0e-3
+ nx=30     ny=15
.MODEL PDEBJT   ZOD  level=2

* Zero-volt sources acting as an ammeter to measure the
* base, collector, and emmitter currents, respectively
VMON1 4 6 0
VMON2 5 0 0
VMON3 2 7 0

.DC VPOS 0.0 12.0 0.5 VBB -2.0 -2.0 1.0
.options NONLIN maxstep=70 maxsearchstep=1
+ searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
+ firstdcopstep=0 lastdcopstep=1
.PRINT DC V(1) I(VMON1) I(VMON2) I(VMON3)
.END
```

Figure 15-4. Two-dimensional BJT netlist. Figures 15-6 and 15-7 provide some of the results of this netlist.

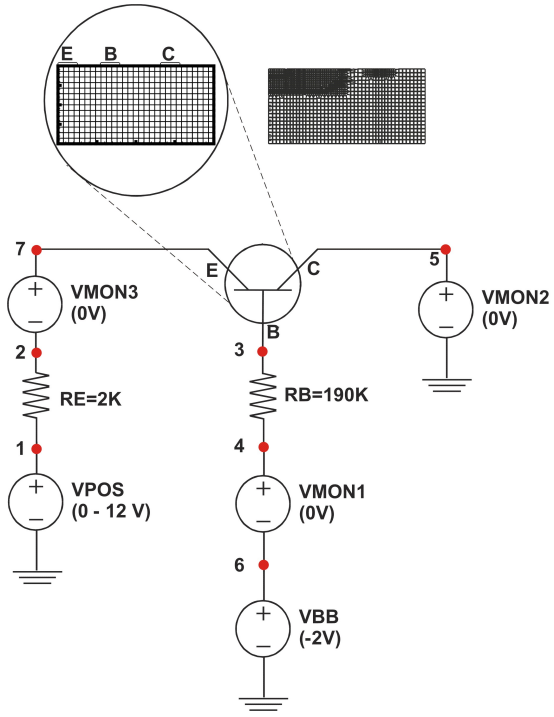

Figure 15-5. Two-dimensional BJT circuit schematic This schematic is for the circuit described by the netlist in figure 15-4. The mesh in the large circle is the mesh used in the example. The other mesh, which contains some mesh refinement, is included in the figure as an example of what is possible with an external mesh generator.

This numbers are used in nearly the same way as the one-dimensional case used the `l` and `nx` parameters. The mesh is Cartesian, and the spacing is uniform.

15.3.2. Doping Profile

As in the one-dimensional example, the two-dimensional example in figure 15-4 specifies nothing about the doping profile, and thus relies on default settings. In this case there are three specified electrodes, which by default results in the doping profile of the bipolar junction transistor (BJT). Section 15.4 provides a complete description of how to specify a doping profile, and describes the various default impurity profiles.

15.3.3. Boundary Conditions and Electrode Configuration

As in the one-dimensional example, the two-dimensional example in figure 15-4 specifies nothing about the electrode configuration or the boundary conditions, and relies on default settings. To be consistent with the default three-terminal doping, the device has terminals that correspond to that of a BJT. All three electrodes (collector, base, emitter) are along the top of the device.

By default all electrodes are considered to be neutral contacts. The boundary conditions applied to the electron density, hole density, and electrostatic potential are all Dirichlet conditions.

Section 15.5 discusses how to specify electrodes in detail (including boundary conditions).

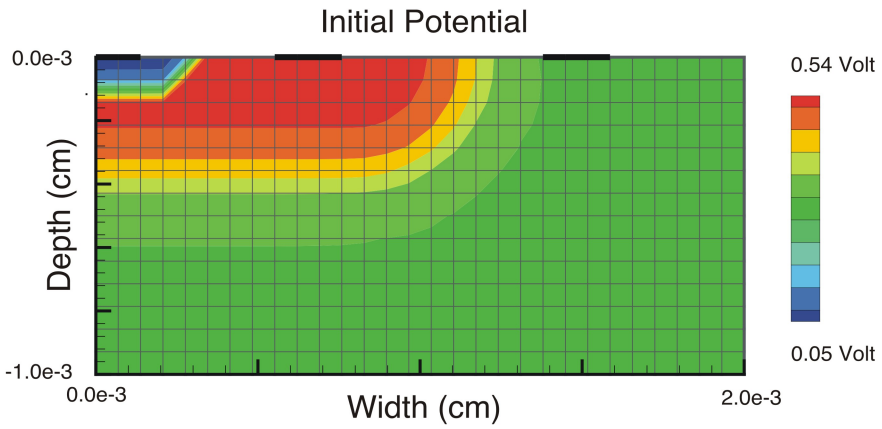

Figure 15-6. Initial two-dimensional BJT result A Tecplot-generated contour plot of the electrostatic potential at the first DC sweep step of the netlist in figure 15-4.

15.3.4. Results

Figures 15-6, 15-7 and 15-8 provide results for the two-dimensional example. The first two figures are contour plots of the electrostatic potential. The first corresponds to the first DC sweep step, where VPOS is set to 0.0 Volts. The second corresponds to the final DC sweep step, in which VPOS has a value of 12.0 volts. The voltage source, VPOS, applies a voltage to the emitter load resistor, RE, so some of the 12.0V is dropped across RE, an the rest is applied to the BJT.

The third figure is an I-V curve of the dependence of the three terminal currents on the applied emitter voltage. For the entire sweep, -2.0 V has been applied to the base load resistor and, as this transistor is a PNP transistor, this results in the transistor being in an "on" state. The emitter-collector current varies nearly linearly with the applied emitter voltage. Also, as can be expected because of current conservation, the three currents sum to nearly zero.

Note that the mesh used to generate these results is visible in figure 15-6, and was generated using the internal "uniform mesh". This mesh will not produce a very accurate result, as it does not resolve the depletion regions very well. Accuracy can often be improved using mesh refinement near the depletion regions. However, such meshes must be read in from an external mesh generator, which currently has limited support as a alpha-level capability.

15.4. Doping Profile

Xyce used default doping profiles in the two examples from the previous section, so no doping parameters were specified. Default profiles are uniquely specified by the number of electrodes. In practice, especially for two-dimensional simulations, the user will generally need to specify the doping profile manually.

15.4.1. Manually Specifying the Doping

Figure 15-9 shows a circuit netlist for a one-dimensional device with a detailed, manual specification of the doping profile. Figure 15-11 illustrates a similar, two-dimensional version of this problem. For this

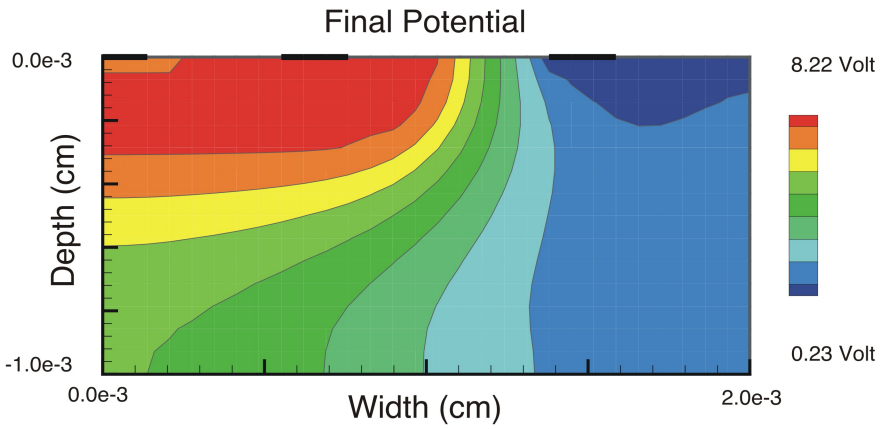

Figure 15-7. Final two-dimensional BJT result. A Tecplot-generated contour plot of the electrostatic potential at the last DC sweep step of the netlist in figure 15-4.

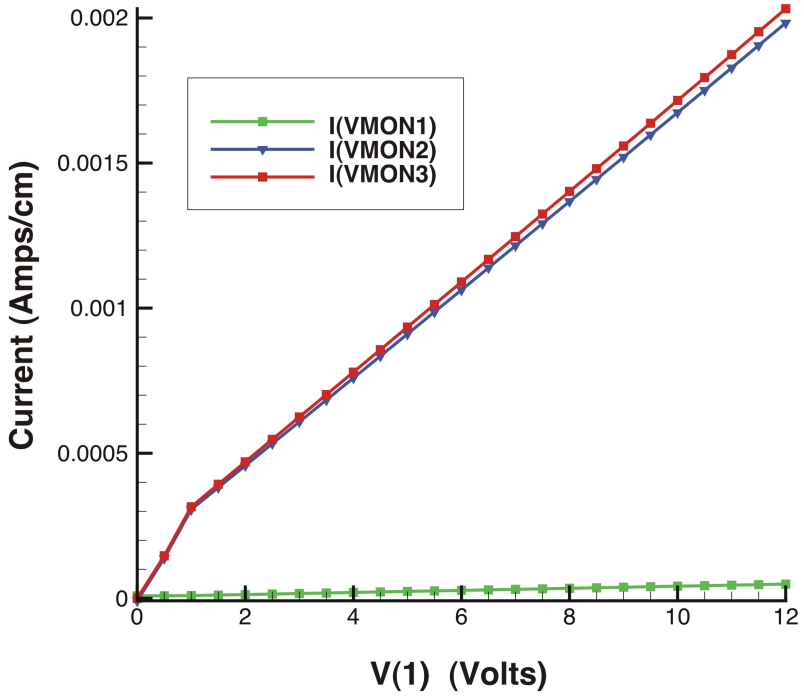

Figure 15-8. I-V two-dimensional BJT result for the netlist in figure 15-4. The three plotted currents are through the three BJT electrodes, and as expected they add (if corrected for sign) to zero. I(VMON1) is the base current, I(VMON2) the collector current, and I(VMON3) the emitter current. V(1) is the voltage applied to the emitter load resistor, RE.

```
Doping and Electrode specification example
vscope   0   1   0.0
rscope   2   1   50.0
cid      3   0   1.0u
r1       4   3   1515.0
vid      4   0   5.00
YPDE Z1DIODE 2 3 PDEDIODE nx=301 l=26.0e-4
* DOPING REGIONS:    region 1,  region 2,   region 3
+ region= {name     =    reg1,      reg2,  reg3
+          function = uniform,  gaussian,  gaussian
+          type     =   ntype,     ptype,     ntype
+          nmax     = 4.0e+12,   1.0e+19,   1.0e+18
+          nmin     = 0.0e+00,   4.0e+12,   4.0e+12
+          xloc     =    0.0 ,  24.5e-04,   9.0e-04
+          xwidth   =    0.0 ,   4.5e-04,   8.0e-04
+          flatx    =    0   ,        0 ,       -1 }
*--------end of  Diode PDE device ----------------
.MODEL PDEDIODE  ZOD  level=1
.options NONLIN maxsearchstep=1 searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
.DC vscope 0 0 1
.print DC v(1) v(2) v(3) v(4) I(vscope) I(vid)
.END
```

Figure 15-9. One-dimensional example, with detailed doping

discussion, the one-dimensional example will be referred to, but the information conveyed is equally applicable to the two-dimensional case.

In both examples, the parameters associated with doping are in red text. The doping is specified with one or more regions, which are summed together to obtain the total profile. Doping regions are specified in a tabular format, with each column representing a different region.

In the one-dimensional example, there are three regions, which are illustrated in figure 15-10. Region 1 is a uniform n-type doping, with a constant magnitude of `4.0e+12` donors per cubic cm. This magnitude is set by the parameter `nmax`. As the doping in this region is spatially uniform, the only meaningful parameters are `function` (which in this case specifies a spatially uniform distribution), `type` (ntype or ptype) and `nmax`. The other parameters, `nmin` through `flatx` (1D) or `flaty` (2D), are ignored for a spatially uniform region.

Region 2 is a more complicated region, in that the doping profile varies spatially. This region is doped with p-type impurities, and the doping profile has a Gaussian shape. Semiconductor processing often consists of an implant followed by an anneal, which results in a diffusive profile. The Gaussian function is a solution to the diffusion problem, when it is assumed that the impurity exists in a fixed quantity.

The peak of the Region 2 doping profile is given by the parameter `nmax`, and is `1.0e+19` acceptors per cubic cm. This peak has a location in the device specified by `xloc=24.5e-04 cm`. The parameters `nmin` and `xwidth` are fitting parameters.

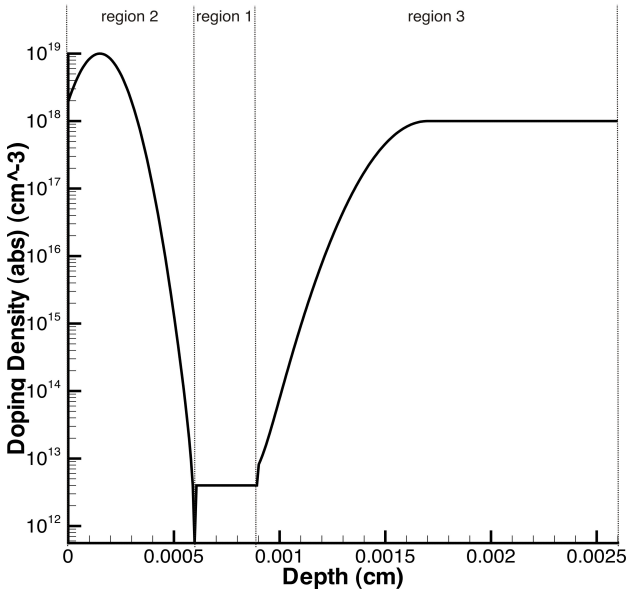

Figure 15-10. Doping profile, absolute value This corresponds to the doping specified by the netlist in figure 15-9

Region 3 is also based on a Gaussian function, but unlike Region 2, it is flat on one side of the peak. This is set by the `flatx` parameter. Table 15-1 lists conventions for "flat" parameters.

Table 15-1. Description of the flatx, flaty doping parameters

flatx or flaty value	Description	1D Cross Section
0	Gaussian on both sides of the peak (`xloc`) location.	
+1	Gaussian if `x>xloc`, flat (constant at the peak value) if `x<xloc`.	
-1	Gaussian if `x<xloc`, flat (constant at the peak value) if `x>xloc`.	

15.4.2. Default Doping Profiles

Xyce has a few default doping profiles that are invoked when the user doesn't specify detailed doping information. The default doping profiles are an artifact of early TCAD device development in Xyce, but are sometimes still useful. In particular, the simple step-junction diode is often a useful canonical problem. It is convenient to invoke a step-junction doping without having to use the tabular specification for more complex regions.

Most real devices will have doping profiles that do not exactly match the default profiles. When attempting to simulate a realistic device, it will be necessary to skip the defaults and use the region tables described in the previous section.

15.4.2.1. One-Dimensional Case

For the one-dimensional case, Xyce assumes that the doping profile is a simple junction diode, with the junction location exactly in the middle. The acceptor and donor concentrations are given by the parameters `Na` and `Nd`, respectively.

The use of `Na` and `Nd`, implicitly specifies a step junction doping profile, and is mutually exclusive with the more complex "doping region" table specification, described in section 15.4.1. If a netlist is input to Xyce with a "doping region" table and `Na` (or `Nd`), the code will immediately exit with an error.

15.4.2.2. Two-Dimensional Case

Doping level defaults in the two-dimensional case are somewhat more complicated than in the one-dimensional case, because having two-dimensions allows for more configurations, and an arbitrary number (2 to 4) of electrodes. During Xyce development, it was decided that the default doping profiles would be determined uniquely by the number of electrodes present. Table 15-2 provides the three available default dopings. In the case of the BJT and MOSFET dopings, it is possible to specify either n-type or p-type using the `type` instance parameter. If the detailed, manual doping is used, then the `type` parameter is ignored.

For a two-electrode device, the default doping is that of a simple diode. Xyce uses the acceptor and donor doping parameters, `Na` and `Nd`, in the same manner as in the one-dimensional device—the junction is assumed to be exactly in the middle of the domain.

For a three-electrode device (as shown in the example), the default doping is that of a bipolar junction transistor (BJT). By default the transistor is a PNP, but by setting the instance parameter `type=NPN`, an NPN transistor can be specified instead. The two-dimensional example in section 15.3 relies on this default.

For a four-terminal device, the default doping is that of a metal-oxide-semiconductor (MOSFET). The maximum number of electrodes is four, and no default profiles are available for more than four electrodes. By default this transistor is assumed to be NMOS, rather than PMOS.

Table 15-2. Default doping profiles for different numbers of electrodes

Number of Electrodes	Doping Profile
2	Step Function Diode
3	Bipolar Junction Transistor (BJT)
4	Metal-Oxide Semiconductor Field-Effect Transistor(MOSFET)

15.5. Electrodes

Because minimal electrodes were specified in the two examples given above, Xyce used the defaults. In practice, especially for two-dimensional simulations, the user must specify the electrodes in more detail.

15.5.1. Electrode Specification

A detailed electrode specification is specified in blue text in figure 15-11. As with the doping parameters, the electrode parameters are specified in a tabular format, in which each table column specifies the different electrode parameters. The `name` parameter is the only required parameter.

The number of specified electrodes must match the number of connected circuit nodes, and the order of the electrode columns, from left to right, is in the same order as the circuit nodes, also from left to right. In the figure 15-11 example, the first electrode column, which specifies an electrode named "anode," is connected to the circuit through circuit node 2. Respectively, the second column, for the "cathode" electrode, is connected to the circuit via circuit node 3.

15.5.1.1. Boundary Conditions

In the example, the default `bc` parameter has been set to "Dirichlet" on all the electrodes. The `bc` parameter sets the type of boundary condition applied to the density variables, the electron density and the hole density. Dirichlet and Neumann are two possible settings for the `bc` parameter. If Dirichlet is specified, the electron and hole densities are set to a specific value at the contact, and the applied values enforce charge neutrality. (Note: The Xyce Reference Guide [3] provides the charge-neutral equation.) If Neumann is specified, Xyce applies a zero-flux condition, which enforces that the current through the electrode will be zero.

This parameter does not affect the electrostatic potential boundary condition. The boundary condition applied to the potential is always Dirichlet, and is (in part) determined from the connected nodal voltage. To apply a specific voltage to an electrode contact, a voltage source should be attached to it, such as `VBB` in figure 15-5.

15.5.1.2. Electrode Material

Table 15-3 lists several different electrode materials that can be specified. The main effect of any metal (nonneutral) material is that Xyce imposes a Schottky barrier at the contact, generally making numerical solutions more difficult, so materials should be applied with caution.

The Xyce Reference Guide [3] provides a detailed description of Schottky barriers and how they are imposed on contacts in Xyce. The guide also provides values for electron affinities of various bulk materials and workfunction values for the various metal contacts.

Table 15-3. Electrode Material Options. NOTE: Neutral contacts are the default, and pose the least problem to the solvers.

Material	Symbol	Comments
neutral	neutral	Default

Table 15-3. Electrode Material Options. NOTE: Neutral contacts are the default, and pose the least problem to the solvers.

Material	Symbol	Comments
aluminum	al	
p+-polysilicon	ppoly	
n+-polysilicon	npoly	
molybdenum	mo	
tungsten	w	
molybdenum disilicide	modi	
tungsten disilicide	wdi	
copper	cu	
platinum	pt	
gold	au	

There is also an `oxideBndryFlag` parameter, which if set to true (`1`), will model the contact as having an oxide layer in between the metal contact and the bulk semiconductor. By default, `oxideBndryFlag` is false (`0`).

15.5.1.3. Location Parameters

Each electrode has three location parameters: `start`, `end`, and `side`.

Xyce assumes the internal mesh to be rectangular, and the electrodes can be on any of the sides. The four side possibilities are: `top`, `bottom`, `right` and `left`. These four sides are parallel to the mesh directions. The `start` and `end` parameters are floating-point numbers that specify the starting and ending location of an electrode, in centimeters.

The lower left hand corner of the mesh rectangle is located at the origin. A `side=bottom` electrode with `start=0.0` and `end=1.0e-4` will originate at the lower left hand corner of the mesh (x=0.0, y=0.0) and end at (x=1.0e-4, y=0.0).

Xyce will attempt to match the specified electrode to the specified mesh. However, if the user specifies a mesh that is not consistent with the electrode locations then the electrodes will not be able to have the exact length specified. For example, if the mesh spacing is $\Delta x =$ 1.0e-5, then the electrodes can only have a length that is a multiple of 1.0e-5.

15.5.2. Electrode Defaults

Defaults exist for each electrode parameter other than names. In practice, the electrode locations are usually explicitly specified using the electrode table. Default electrode locations were created to correspond with the default dopings; they should only be used in that context.

```
Doping and electrode specification example
vscope   1   0   0.0
rscope   2   1   50.0
cid      3   0   1.0u
r1       4   3   1515.0
vid      4   0   1.00
*------------- Diode PDE device -----------------
YPDE Z1DIODE 2 3 PDEDIODE
+ tecplotlevel=1 txtdatalevel=1 cyl=1
+ meshfile=internal.msh
+ nx=25  l=70.0e-4 ny=40  w=26.0e-4
 * ELECTRODES:              ckt node 2, ckt node 3
+ node = {name            =     anode,    cathode
+         bc              = dirichlet,  dirichlet
+         start           =     0.002, 0.002
+         end             =     0.005, 0.005
+         side            =       top,     bottom
+         material        =   neutral,    neutral
+         oxideBndryFlag =          0,           0 }
* DOPING REGIONS:    region 1,  region 2,   region 3
+ region= {name    =     reg1,      reg2,  reg3
+          function = uniform,  gaussian, gaussian
+          type     =   ntype,     ptype,     ntype
+          nmax     = 4.0e+12,   1.0e+19,   1.0e+18
+          nmin     = 0.0e+00,   4.0e+12,   4.0e+12
+          xloc     =    0.0 ,  60.0e-04,    100.0
+          xwidth   =    0.0 ,   4.0e-04,      1.0
+          yloc     =    0.0 ,  24.5e-04,   9.0e-04
+          ywidth   =    0.0 ,   4.5e-04,   8.0e-04
+          flatx    =    0   ,       -1 ,       -1
+          flaty    =    0   ,        0 ,       -1 }
*--------end of  Diode PDE device ---------------
.MODEL PDEDIODE  ZOD  level=2
.options NONLIN maxsearchstep=1 searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
.DC vscope 0 0 1
.print DC v(1) v(2) v(3) v(4) I(vscope) I(vid)
.END
```

Figure 15-11. Two-dimensional example, with detailed doping and detailed electrodes.

15.5.2.1. Location Parameters

In practice, the electrode locations will usually be explicitly specified, but they have defaults to correspond with the default dopings. The default electrode locations in one-dimensional devices are for a diode. One electrode is located at `x=xmin`, while the other is located at `x=xmax`.

The default electrode locations in two-dimensional devices depend on the number of electrodes, similar to the default dopings. Table 15-2 can be used to determine such configurations. For the two-terminal diode, the two electrodes are along the y-axis, at the `x=xmin` and `x=xmax` extrema. For the three-terminal BJT, all three electrodes are parallel to the x-axis, along the top, at `y=ymax`. For the four-terminal MOSFET, the drain, gate, and source electrodes are also along the top, but the bulk electrode spans the entire length of the bottom of the mesh, at `y=ymin`.

15.6. Meshes

One- and two-dimensional devices can create Cartesian meshes. For two-dimensional devices, users must specify `meshfile=internal.msh` to invoke the Cartesian meshing capability (this is necessary for historical reasons). Meshes generated in this manner are very simple as there are only two parameters per dimension, and the resulting mesh is uniform. Figure 15-6 provides an example of such a mesh. Mesh spacing is determined from the following expressions:

$$\Delta x = \frac{l}{nx - 1} \tag{15.8}$$

$$\Delta y = \frac{w}{ny - 1} \tag{15.9}$$

This mesh specification assumes the domain is a rectangle. Nonrectangular domains can only be described using an external mesh program.

Externally generated meshes for 1D devices can be including using `meshfile=<filename>` in the PDE device instance line. The file specified by `<filename>` must consist of two space-delimited columns of numbers. The first column specifies the location of the mesh points. The second column is not currently used, but it must exist, so it is suggested to make it a column of zeros.

15.7. Cylindrical meshes

For two-dimensional devices, the simulation area may be a cylinder slice. This capability is turned on by the instance parameter `cyl=1` It is assumed that the axis of the cylinder corresponds to the minimum radius (or x-axis value) of the mesh, while the circumference corresponds to the maximum radius (or maximum x-axis value).

15.8. Mobility Models

There are several mobility models available to the one- and two-dimensional devices, and they are listed in Table 15-4. These models are fairly common, and can be found in most device simulators. [49] [50]. The Xyce Reference Guide [3] descibes these models in more detail.

Table 15-4. Mobility models available for PDE devices

Mobility Name	Description	Reference
`arora`	Basic mobility model	Arora, et al. [54]
`analytic` or `caughey-thomas`	Basic mobility model	Caughy and Thomas [55]
`carr`	Includes carrier-carrier interactions	Dorkel and Leturq [56]
`philips`	Philips model	Klassen [57, 58]

Setting the `mobmodel` parameter to the name of the model (as provided in the first column of table 15-4) specifies the mobility model from the netlist. The mobility model is specified as an instance parameter on the device instance line, as (typically) `mobmodel=arora`. Figure 15-4 provides a more detailed example.

The default mobility is "arora", which is a basic model lacking carrier or field dependence. Because it lacks these dependencies, it generally is more numerically robust. The "carr" model include carrier-carrier dependence, as does the "philips" model. For all of these models, field dependence can be optionally turned on from the netlist, using the `fielddep=true` parameter.

15.9. Bulk Materials

The bulk material is specified using the `bulkmaterial` instance parameter. Xyce supports Silicon (`si`) as a default bulk material. It can also simulate several III-V materials, including Gallium Arsenide (`gaas`), Germanium (`ge`), Indium Aluminum Arsenide (`inalas` or `alinas`), Indium Galium Arsenide (`ingaas` or `gainas`), Indium Phosphide(`inp`), and Indium Galium Phosphide (`ingap`); but these materials have not been extensively tested. The mobility models described in the previous section each support most of these materials.

15.10. Output and Visualization

15.10.1. Using the `.PRINT` Command

For simple plots (such as I-V curves), output results for Xyce can be generated with the `.PRINT` statement, which is described in detail in section 9.1.1. Figures 15-3 and 15-8 are examples of the kind of data that is produced with `.PRINT` statement netlist commands. These particular figures were plotted in Tecplot, but many other plotting programs would also have worked, including XDAMP [59].

15.10.2. Multidimensional Plots

Device simulation has visualization needs which go beyond that of conventional circuit simulation. Multidimensional perspective and/or contour plots are often desirable. Xyce is capable of outputting multi-dimensional plot data in several formats, including Tecplot (available for purchase from http://www.tecplot.com), gnuplot (available free from http://www.gnuplot.info), and SGPLOT. Currently,

the options for each of these formats can only enable or disable the output of files, and when enabled, a new file (or a new append to an existing file) will happen at every time step or DC sweep step.

For long simulations, this may produce a prohibitive number of files. There is no equivalent to the `.OPTIONS OUTPUT INITIAL_INTERVAL` command, nor does the output of plot data use this command. Plot files are either output at every step or not at all.

For each type of plot file, the file is placed in the execution directory. Each individual device instance is given a unique file, or files, and the file names are derived from the name of the PDE device instance. The instance names provides the prefix, and the file type (Tecplot, gnuplot, Sgplot) determines the suffix.

15.10.2.1. Tecplot Data

Tecplot is a commercial plotting program from Amtec Engineering, Inc., and is a good choice for creating contour plots of spatially dependent data. All of the graphical examples in this chapter were created with Tecplot. (See figures 15-6 and 15-7 for examples.) The output of Tecplot files is enabled using the instance parameter, `tecplotlevel=1`. If set to zero, no Tecplot files are output. If set to one, Xyce outputs a separate Tecplot file for each nonlinear solve. If set to two, Xyce creates a single Tecplot file containing data for every nonlinear solve and appends the file at the end of each solve.

By default `tecplotlevel` is set to one, meaning the code will produce a separate Tecplot file for each time step or DC sweep step. The suffix for a Tecplot (ASCII text) data file is *.dat.

15.10.2.2. Gnuplot Data

Gnuplot is an open source plotting program available on most Linux/Unix platforms. The parameter for this type of output is `gnuplotlevel=1`. This type of output file is off (zero) by default, meaning no gnuplot files will be output. The suffix for gnuplot files is *gnu.dat. Like Tecplot files, gnuplot files are also in ASCII text format.

15.10.3. Additional Text Data

Xyce can also output additioanl information for each PDE device by setting the instance parameter, `txtdatalevel=1`. It is on (1) by default, so this output will happen unless specifically disabled by setting the parameter to zero. A typical output file (associated with the netlist given in figure 15-4) is shown in figure 15-12.

```
Global data for DC step    1:
Current Time =   0.0000e+00
       Vmin  =  -8.6931e-06
       Vmax  =   5.4030e-01
       NnMin =   0.0000e+00
       NnMax =   1.0000e+16
       NpMin =   3.9240e+03
       NpMax =   1.0000e+19

Information for electrode: COLLECTOR
potential:   2.9795e-01
  current:   8.5365e-06
  charge:   -6.6211e-15
  dIdVckt:   3.7993e-02
  dQdVckt:   0.0000e+00

Information for electrode: BASE
potential:   5.4030e-01
  current:  -8.5408e-06
  charge:    1.5958e-14
  dIdVckt:   1.0463e+01
  dQdVckt:   0.0000e+00

Information for electrode: EMITTER
potential:  -8.6931e-06
  current:   4.3465e-09
  charge:   -2.3232e-13
  dIdVckt:   7.2130e+01
  dQdVckt:   0.0000e+00
```

Figure 15-12. Text output, from the circuit given in figure 15-4.

This page intentionally left blank.

BIBLIOGRAPHY

[1] Laurence Nagel and Ronald Rohrer. Computer analysis of nonlinear circuits, excluding radiation (cancer). *IEEE Journal of Solid-State Circuits*, sc-6(4):166–182, 1971.

[2] Tom Quarles. Spice3f5 Users' Guide. Technical report, University of California-Berkeley, Berkeley, California, 1994.

[3] Eric R. Keiter, Thomas V. Russo, Richard L. Schiek, Heidi K. Thornquist, Ting Mei, Jason C. Verley, Karthik V. Aadithya, and Joshua D. Schickling. Xyce Parallel Electronic Simulator: Reference Guide, Version 7.6. Technical Report SAND2022-14886, Sandia National Laboratories, Albuquerque, NM, 2022.

[4] Orcad PSpice User's Guide. Technical report, Orcad, Inc., 1998.

[5] *gEDA Project Home Page*. `http://www.geda.seul.org/` .

[6] *Qucs-S: Qucs with SPICE*. `http://ra3xdh.github.io/` .

[7] A. S. Grove. *Physics and Technology of Semiconductor Devices*. John Wiley and Sons, Inc., 1967.

[8] Hiroshi Akima. A new method of interpolation and smooth curve fitting based on local procedures. *J. ACM*, 17(4):589–602, October 1970.

[9] Gisela Engeln-Müllges and Frank Uhlig. *Akima and Renner Subsplines*, pages 341–352. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[10] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric lagrange interpolation. *SIAM Rev*, 46:501–517.

[11] H. A. Watts, E. R. Keiter, S. A. Hutchinson, and R. J. Hoekstra. Time integration for the Xyce parallel electronic simulator. In *ISCAS 01*, October 2000.

[12] K.E. Brenan, S.L Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Society for Industrial and Applied Mathmatics, Philadelphia, 1996.

[13] J.C. Helton and F.J. Davis. Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. *Reliability Engineering and System Safety*, 81(1):23 – 69, 2003.

[14] G.S. Fishman. *Monte carlo: Concepts, algorithms, and applications*. Springer-Verlag, New York, NY, USA, 1996.

[15] B. M. Adams, L. E. Bauman, W. J. Bohnhoff, K. R. Dalbey, J. P. Eddy, M. S. Ebeida, M. S. Eldred, P. D. Hough, K. T. Hu, J. D. Jakeman, Ahmad Rushdi, L. P. Swiler, J. A. Stephens, D. M. Vigil, and T. M. Wildey. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.2 theory manual. Technical Report SAND2014-4253, Sandia National Laboratories, Albuquerque, NM, Updated May 2015. Available online from `http://dakota.sandia.gov/documentation.html`.

[16] B. M. Adams, L. E. Bauman, W. J. Bohnhoff, K. R. Dalbey, J. P. Eddy, M. S. Ebeida, M. S. Eldred, P. D. Hough, K. T. Hu, J. D. Jakeman, Ahmad Rushdi, L. P. Swiler, J. A. Stephens, D. M. Vigil, and T. M. Wildey. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.2 users manual. Technical Report SAND2014-4633, Sandia National Laboratories, Albuquerque, NM, Updated May 2015. Available online from `http://dakota.sandia.gov/documentation.html`.

[17] D. Xiu and G. M. Karniadakis. The Wiener-Askey polynomial chaos for stochastic differential equations. *SIAM J. Sci. Comput.*, 24(2):619–644, 2002.

[18] D. Xiu. *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton University Press, 2010.

[19] Eric T. Phipps. *Stokhos embedded uncertainty quantification methods*. `https://trilinos.github.io/stokhos.html`, 2015.

[20] Eric R. Keiter, Karthik V. Aadithya, Ting Mei, Heidi Thornquist, Pete Sholander, , and Ian Z. Wilcox. Exploring advanced embedded uncertainty quantification methods in xyce. Technical Report SAND2019-11297, Sandia National Laboratories, 2019.

[21] R. W. Walters. Towards stochastic fluid mechanics via polynomial chaos. In *Proceedings of the 41st AIAA Aerospace Sciences Meeting and Exhibit*, number AIAA-2003-0413, Reno, NV, January 6–9, 2003.

[22] S.A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Dokl. Akad. Nauk SSSR*, 4:240–243, 1963.

[23] A. Stroud. *Approximate Calculation of Multiple Integrals*. Prentice Hall, 1971.

[24] B. Gilbert. A precise four-quadrant multiplier with subnanosecond response. *IEEE Journal of Solid-State Circuits*, 3(4):365–373, Dec 1968.

[25] Dale E. Hocevar, Ping Yang, Timothy N. Trick, and Berton D. Epler. Transient sensitivity computation for mosfet circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(4), October 1985.

[26] Frank Liu and Peter Feldmann. A time-unrolling method to compute sensitivity of dynamic systems. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6, June 2014.

[27] Arie Meir and Jaijeet Roychowdhury. Blast: Efficient computation of nonlinear delay sensitivities in electronic and biological networks using barycentric lagrange enabled transient adjoint analysis. In *DAC '12: Proceedings of the 2012 Design Automation Conference*, New York, NY, USA, 2012. ACM.

[28] Eric R. Keiter, Laura P. Swiler, and Ian Z. Wilcox. Gradient-enhanced polynomial chaos methods for circuit simulation. In Ulrich Langer, Wolfgang Amrhein, and Walter Zulehner, editors, *Scientific Computing in Electrical Engineering*, pages 55–68, Cham, 2018. Springer International Publishing.

[29] Eric R. Keiter, Karthik Venkatraman Aadithya, Ting Mei, Heidi K. Thornquist, Peter E. Sholander, and Ian Zachary Wilcox. Exploring advanced embedded uncertainty quantification methods in xyce. 9 2019.

[30] *5. Newton's Method for Nonlinear Equations and Unconstrained Minimization*, pages 86–110.

[31] Touchstone File Format Specification, Version 2.0. Technical report, Open IBIS Forum, 2009.

[32] Ljiljunu Trujkovit, Robert C. Melville, and Sun-Chin Fang. Passivity and no-gain properties establish global convergence of a homotopy method for DC operating points. *Proceedings - IEEE International Symposium on Circuits and Systems*, 2:914–917, 1990.

[33] Robert C. Melville, Ljiljana Trajkovic, San-Chin Fang, and Layne T. Watson. Artificial parameter homotopy methods for the DC operating point problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(6):861–877, 1993.

[34] *LOCA: Library of Continuation Algorithms*. `http://trilinos.sandia.gov/packages/nox`, 2002.

[35] *LOCA: Library of Continuation Algorithms*. `http://www.cs.sandia.gov/loca/`, 2002.

[36] Andrew G. Salinger, Nawaf M. Bou-Rabee, Roger P. Pawlowski, , Edward D. Wilkes, Elizabeth A. Burroughs, Richard B. Lehoucq, and Louis A. Romero. Library of continuation algorithms: Theory and implementation manual. Technical Report SAND2002-0396, Sandia National Laboratories, Albuquerque, NM, 2002.

[37] J. Roychowdhury. *Private Communication*, 2003.

[38] Eric R. Keiter, Heidi K. Thornquist, Robert J. Hoekstra, Thomas V. Russo, Richard L. Schiek, and Eric L. Rankin. Parallel transistor-level circuit simulation. In Peng Li, Luís Miguel Silveira, and Peter Feldmann, editors, *Simulation and Verification of Electronic and Biological Systems*, pages 1–21. Springer Netherlands, 2011. 10.1007/978-94-007-0149-6_1.

[39] M. Heroux, et. al. An overview of the Trilinos project. *ACM TOMS*, 31(3):397–423, 2005. `http://trilinos.sandia.gov`.

[40] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. 7(3):856–869, 1986.

[41] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA., second edition, 2003.

[42] Heidi K. Thornquist, Eric R. Keiter, Robert J. Hoekstra, David M. Day, and Erik G. Boman. A parallel preconditioning strategy for efficient transistor-level circuit simulation. In *ICCAD '09: Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 410–417, New York, NY, USA, 2009. ACM.

[43] S. Rajamanickam, E.G. Boman, and M.A. Heroux. ShyLU: A hybrid-hybrid solver for multicore platforms. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 631 –643, May 2012.

[44] C. Bomhof and H. vanderVorst. A parallel linear system solver for circuit simulation problems. *Num. Lin. Alg. Appl.*, 7(7-8):649–665, 2000.

[45] Erik Boman, Karen Device, Robert Heaphy, Bruce Hendrickson, William F. Mitchell, Matthew St. John, and Courtenay Vaughan. *Zoltan: Data-Management Services for Parallel Applications:User's Guide*. `http://www.cs.sandia.gov/zoltan`, 2004.

[46] Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra, Eric L. Rankin, Thomas V. Russo, and Lon J. Waters. Computational algorithms for device-circuit coupling. Technical Report SAND2003-0080, Sandia National Laboratories, Albuquerque, NM, January 2003.

[47] Kartikeya Mayaram and Donald O. Pederson. Coupling algorithms for mixed-level circuit and device simulation. *IEEE Transactions on Computer Aided Design*, II(8):1003–1012, 1992.

[48] David A. Johns and Ken Martin. *Analog Integrated Circuit Design*. John Wiley & Sons, Inc., 1997.

[49] Z. Yu, D. CHen, L. So, and R. W. Dutton. PISCES-2ET–Two dimensional device simulation for silicon and heterostructures. Technical report, Stanford University, 1994.

[50] Davinci User's Manual. Technical report, TCAD Business Unit, Synopsys Corporation, 1998.

[51] Kevin M. Kramer and W. Nicholas G. Hitchon. *Semiconductor Devices: A Simulation Approach*. Prentice-Hall, Upper Saddle River, New Jersey, 1997.

[52] S. Selberherr. *Analysis and Simulation of Semiconductor Devices*. Springer-Verlag, New York, 1984.

[53] Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra, Eric L. Rankin, Thomas V. Russo, and Lon J. Waters. Computational algorithms for device-circuit coupling. Technical Report SAND2003-0080, Sandia National Laboratories, Albuquerque, NM, January 2003.

[54] N.D. Arora, J.R. Hauser, and D.J. Roulston. Electron and hole mobilities in silicon as a function of concentration and temperature. *IEEE Transactions on Electron Devices*, ED-29:292–295, 1982.

[55] D.M. Caughey and R.E. Thomas. Carrier mobilities in silicon empirically related to doping and field. *Proc. IEEE*, 55:2192–2193, 1967.

[56] J.M. Dorkel and Ph. Leturq. Carrier mobilities in silicon semi-empirically related to temperature, doping, and injection level. *Solid-State Electronics*, 24(9):821–825, 1981.

[57] D.B.M. Klaassen. A unified mobility model for device simulation - i. *Solid State Electronics*, 35:953–959, 1992.

[58] D.B.M. Klaassen. A unified mobility model for device simulation - ii. *Solid State Electronics*, 35:961–967, 1992.

[59] *XDAMP Graphical User Interface*. `http://www.cs.sandia.gov/esimtools/xdamp.html` .

APPENDIX A. Third Party Licenses

Xyce makes use of code developed by various third parties. The following text is provided to comply with the licenses of the codes that require it.

The ksparse solver in Xyce contains code derived from SPICE 3f5 source code:

```
Copyright (c) 1985-1991 The Regents of the University of California.
All rights reserved.

Permission is hereby granted, without written agreement and without
license or royalty fees, to use, copy, modify, and distribute this
software and its documentation for any purpose, provided that the
above copyright notice and the following two paragraphs appear in all
copies of this software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY
FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF
CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
ENHANCEMENTS, OR MODIFICATIONS.
```

Xyce's linear solver makes use of the AMD library:

```
AMD, Copyright (c), 1996-2015, Timothy A. Davis,
Patrick R. Amestoy, and Iain S. Duff.  All Rights Reserved.
Used in Xyce under the LGPL v2.1 license.
```

Parallel builds of Xyce use the Open MPI library:

```
Most files in this release are marked with the copyrights of the
organizations who have edited them.  The copyrights below are in no
particular order and generally reflect members of the Open MPI core
team who have contributed code to this release.  The copyrights for
code used under license from other parties are included in the
corresponding files.
```

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.

- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Xyce uses the Trilinos Solver Framework:

Trilinos: An Object-Oriented Solver Framework
Copyright (2001) Sandia Corporation

Copyright (2001) Sandia Corporation. Under the terms of Contract DE-AC04-94AL85000, there is a non-exclusive license for use of this work by or on behalf of the U.S. Government. Export of this program may require a license from the United States Government.

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the Corporation nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY SANDIA CORPORATION "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SANDIA CORPORATION OR THE CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NOTICE: The United States Government is granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license in this data to reproduce, prepare derivative works, and perform publicly and display publicly. Beginning five (5) years from July 25, 2001, the United States Government is granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license in this data to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so.

NEITHER THE UNITED STATES GOVERNMENT, NOR THE UNITED STATES DEPARTMENT OF ENERGY, NOR SANDIA CORPORATION, NOR ANY OF THEIR EMPLOYEES, MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR ASSUMES ANY LEGAL LIABILITY OR RESPONSIBILITY FOR THE ACCURACY, COMPLETENESS, OR USEFULNESS OF ANY INFORMATION, APPARATUS, PRODUCT, OR PROCESS DISCLOSED, OR REPRESENTS THAT ITS USE WOULD NOT INFRINGE PRIVATELY OWNED RIGHTS.

Some versions of Xyce use the Intel Math Kernel Library:

Intel Simplified Software License (Version April 2018)

Copyright (c) 2018 Intel Corporation.

Use and Redistribution. You may use and redistribute the software (the "Software"), without modification, provided the following conditions are met:

* Redistributions must reproduce the above copyright notice and the following terms of use in the Software and in the documentation and/or other materials provided with the distribution.

* Neither the name of Intel nor the names of its suppliers may be used to endorse or promote products derived from this Software without specific prior written permission.

* No reverse engineering, decompilation, or disassembly of this Software is permitted.

Limited patent license. Intel grants you a world-wide, royalty-free, non-exclusive license under patents it now or hereafter owns or controls to make, have made, use, import, offer to sell and sell ("Utilize") this Software, but solely to the extent that any such patent is necessary to Utilize the Software alone. The patent license shall not apply to any combinations which include this software. No hardware per se is licensed hereunder.

Third party and other Intel programs. "Third Party Programs" are the files listed in the "third-party-programs.txt" text file that is included with the Software and may include Intel programs under separate license terms. Third Party Programs, even if included with the distribution of the Materials, are governed by separate license terms and those license terms solely govern your use of those programs.

DISCLAIMER. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT ARE DISCLAIMED. THIS SOFTWARE IS NOT INTENDED FOR USE IN SYSTEMS OR APPLICATIONS WHERE FAILURE OF THE SOFTWARE MAY CAUSE PERSONAL INJURY OR DEATH AND YOU AGREE THAT YOU ARE FULLY RESPONSIBLE FOR ANY CLAIMS, COSTS, DAMAGES, EXPENSES, AND ATTORNEYS' FEES ARISING OUT OF ANY SUCH USE, EVEN IF ANY CLAIM ALLEGES THAT INTEL WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE MATERIALS.

LIMITATION OF LIABILITY. IN NO EVENT WILL INTEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. YOU AGREE TO INDEMNIFY AND HOLD INTEL HARMLESS AGAINST ANY CLAIMS AND EXPENSES RESULTING FROM YOUR USE OR UNAUTHORIZED USE OF THE SOFTWARE.

No support. Intel may make changes to the Software, at any time without notice, and is not obligated to support, update or provide training for the Software.

Termination. Intel may terminate your right to use the Software in the event of your breach of this Agreement and you fail to cure the breach within a reasonable period of time.

Feedback. Should you provide Intel with comments, modifications, corrections, enhancements or other input ("Feedback") related to the Software Intel will be free to use, disclose, reproduce, license or otherwise distribute or exploit the Feedback in its sole discretion without any obligations or restrictions of any kind, including without limitation, intellectual property rights or licensing obligations.

Compliance with laws. You agree to comply with all relevant laws and regulations governing your use, transfer, import or export (or prohibition thereof) of the Software.

Governing law. All disputes will be governed by the laws of the United States of America and the State of Delaware without reference to conflict of law principles and subject to the exclusive jurisdiction of the state or federal courts sitting in the State of Delaware, and each party agrees that it submits to the personal jurisdiction and venue of those courts and waives any objections. The United Nations Convention on Contracts for the International Sale of Goods (1980) is specifically excluded and will not apply to the Software.

Xyce's implementation of the Diode-CMC model, version 2.0.0, is derived from Verilog-A sources provided under the following license:

Silicon Integration Initiative (Si2)
Compact Model Coalition In-Code Statement

Software is distributed as is, completely without warranty or service support. NXP Semiconductors, Hiroshima University, and Silicon Integration Initiative, Inc. (Si2), along with their employees are not liable for the condition or performance of the software.

NXP Semiconductors, Hiroshima University, and Si2 own the copyright and grant users a perpetual, irrevocable, worldwide, non-exclusive, royalty-free license with respect to the software as set forth below.

NXP Semiconductors, Hiroshima University, and Si2 hereby disclaim all implied warranties.

NXP Semiconductors, Hiroshima University, and Si2 grant the users the right to modify, copy, and redistribute the software and

```
documentation, both within the user's organization and externally,
subject to the following restrictions

  1. The users agree not to charge for the NXP Semiconductors,
  Hiroshima University, and Si2 -developed code itself but may charge
  for additions, extensions, or support.

  2. In any product based on the software, the users agree to
  acknowledge NXP Semiconductors, Hiroshima University, and Si2 that
  developed the software.  This acknowledgment shall appear in the
  product documentation.

  3. Redistributions to others of source code and documentation must
  retain the copyright notice, disclaimer, and list of conditions.

  4. Redistributions to others in binary form must reproduce the
  copyright notice, disclaimer, and list of conditions in the
  documentation and/or other materials provided with the distribution.
```

Xyce's implementations of the L-UTSOI and EKV 2.6 models are derived from Verilog-A sources provided under the ECL-2.0 license. The L-UTSOI is Copyright 2020 CEA-Leti.

The ECL-2.0 license text is as follows:

```
Licensed under Educational Community License, Version 2.0 (the
"License"); you may not use this file except in compliance with the
License. You may obtain a copy of the license at

http://opensource.org/licenses/ECL-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied. See the License for the specific language governing
permissions and limitations under the License.
```

Xyce's implementation of the MEXTRAM model, version 504.12.1, is derived from Verilog-A sources provided under the following license:

```
Copyright (c) 2000-2007, NXP Semiconductors
Copyright (c) 2007-2014, Delft University of Technology
Copyright (c) 2015, Auburn University

INTELLECTUAL PROPERTY NOTICE, DISCLAIMER AND LICENSE

The Mextram model and documentation presented at this website, denoted
as the Model, has been developed by NXP Semiconductors until 2007,
Delft University of Technology from 2007 to 2014, and Auburn
```

University since April 2015.

The Model is distributed as is, completely without any expressed or implied warranty or service support. NXP Semiconductors, Delft University of Technology, Auburn University and their employees are not liable for the condition or performance of the Model.

NXP Semiconductors, Delft University of Technology, Auburn University own the copyright and grant users a perpetual, irrevocable, worldwide, non-exclusive, royalty-free license with respect to the Model as set forth below.

NXP Semiconductors, Delft University of Technology, Auburn University hereby disclaim all implied warranties.

NXP Semiconductors, Delft University of Technology, Auburn University grant the users the right to modify, copy, and redistribute the Model and documentation, both within the user's organization and externally, subject to the following restrictions

1. The users agree not to charge for the code itself but may charge for additions, extensions, or support.

2. In any product based on the Model, the users agree to acknowledge NXP Semiconductors, Delft University of Technology, Auburn University that developed the Model. This acknowledgment shall appear in the product documentation.

3. The users agree to obey all restrictions governing redistribution or export of the Model.

4. The users agree to reproduce any copyright notice which appears on the Model on any copy or modification of such made available to others.

Xyce's implementation of the HICUM/L0 model, version 1.32, is derived from Verilog-A sources provided under the following license:

Software is distributed as is, completely without warranty or service support. Michael Schroter and his team members are not liable for the condition or performance of the software.

Michael Schroter owns the copyright and grants users a perpetual, irrevocable, worldwide, non-exclusive, royalty-free license with respect to the software as set forth below.

Michael Schroter hereby disclaims all implied warranties.

Michael Schroter grants the users the right to modify, copy, and redistribute the software and documentation, both within the user's organization and externally, subject to the following restrictions.

1. The users agree not to charge for the model owner's code itself but may charge for additions, extensions, or support.

2. In any product based on the software, the users agree to acknowledge Michael Schroter who developed the model and software. This acknowledgment shall appear in the product documentation.

3. Redistributions to others of source code and documentation must retain the copyright notice, disclaimer, and list of conditions.

4. Redistributions to others in binary form must reproduce the copyright notice, disclaimer, and list of conditions in the documentation and/or other materials provided with the distribution

Xyce's implementation of the HICUM/L2 model, version 2.34, is derived from Verilog-A sources provided under the following license:

The Software is distributed as is, completely without expressed or implied warranty or service support. Michael Schroter and his team members are not liable for the condition or performance of the software.

Michael Schroter owns the copyright and grants users a perpetual, irrevocable, worldwide, non-exclusive, royalty-free license with respect to the software as set forth below.

Michael Schroter hereby disclaims all implied warranties.

Michael Schroter grants the users the right to modify, copy, and redistribute the software and documentation, both within the user's organization and externally, subject to the following restrictions.

1. The users agree not to charge for the model owner code itself but may charge for additions, extensions, or support.

2. In any product based on the software, the users agree to acknowledge Michael Schroter that developed the model and software. This acknowledgment shall appear in the product documentation.

3. Users agree to obey all government restrictions governing redistribution or export of the software.

4. The Users agree to reproduce any copyright notice which appears on the software and documentation on any copy or modification of such

```
   made available to others.
```

Xyce's implementations of the BSIM3 v.3.2.2, the BSIM4 v. 4.6.1, the BSIM4 v. 4.7.0, and the BSIM-SOI v. 3.2, are based on the original code of those devices provided by University of California, Berkeley. They all have the following license:

```
Software is distributed as is, completely without warranty or service
support. The University of California and its employees are not liable
for the condition or performance of the software.

The University owns the copyright but shall not be liable for any
infringement of copyright or other proprietary rights brought by third
parties against the users of the software.

The University of California hereby disclaims all implied warranties.

The University of California grants the users the right to modify,
copy, and redistribute the software and documentation, both within the
user's organization and externally, subject to the following
restrictions

1. The users agree not to charge for the University of California code
   itself but may charge for additions, extensions, or support.

2. In any product based on the software, the users agree to
   acknowledge the UC Berkeley BSIM Research Group that developed the
   software. This acknowledgment shall appear in the product
   documentation.

3. The users agree to obey all U.S. Government restrictions governing
   redistribution or export of the software.

4. The users agree to reproduce any copyright notice which appears on
   the software on any copy or modification of such made available
   to others.
```

Version 4.8.2 of the BSIM4 is licensed under the Educational Community License, Version 2.0:

```
Copyright 2020 University of California

Licensed under Educational Community License, Version 2.0 (the
"License"); you may not use this file except in compliance with the
License. You may obtain a copy of the license at

http://opensource.org/licenses/ECL-2.0

Unless required by applicable law or agreed to in writing, software
```

```
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied. See the License for the specific language governing
permissions and limitations under the License.
```

Xyce's implementation of the BSIM6 model, version 6.1.1, is derived from Verilog-A sources provided under the following license:

```
Software is distributed as is, completely without warranty or service
support. The University of California and its employees are not liable
for the condition or performance of the software.

The University owns the copyright but shall not be liable for any
infringement of copyright or other proprietary rights brought by third
parties against the users of the software.

The University of California hereby disclaims all implied warranties.

The University of California grants the users the right to modify,
copy, and redistribute the software and documentation, both within the
user's organization and externally, subject to the following
restrictions

1. The users agree not to charge for the University of California code
   itself but may charge for additions, extensions, or support.

2. In any product based on the software, the users agree to
   acknowledge the UC Berkeley BSIM Research Group that developed the
   software. This acknowledgment shall appear in the product
   documentation.

3. The users agree to obey all U.S. Government restrictions governing
   redistribution or export of the software.

4. The users agree to reproduce any copyright notice which appears on
   the software on any copy or modification of such made available
   to others.
```

Xyce's implementations of the BSIM-SOI model, versions 4.5 and 4.6.1, are derived from Verilog-A sources provided under the following license:

```
The Regents of the University of California and Analog Devices, Inc.
("Authors") own the copyright but shall not be liable for any
infringement of copyright or other proprietary rights brought by third
parties against the users of the software.

The Authors jointly grant the users the right to modify, copy, and
```

redistribute the software and documentation, both within the user's organization and externally, subject to the following conditions:

1. The users agree not to charge for the original source code itself but may charge for additions, extensions, or support.

2. In any product based on the software, the users agree to acknowledge the UC Berkeley BSIM Research Group that developed the software. This acknowledgment shall appear in the product documentation.

3. The users agree to obey all U.S. Government restrictions governing redistribution or export of the software.

4. Redistributions of source code must retain the above copyright notices, this list of conditions and the following disclaimer.

5. Redistributions in binary form must reproduce the above copyright notices, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

6. Users may not claim authorship of the original software. Modified versions of the software must be plainly marked as such.

7. Users may not suggest any sponsorship or endorsement by the Authors of users' product.

THIS SOFTWARE IS PROVIDED BY AUTHORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Xyce's implementations of the BSIM-CMG model, versions 107.0.0 and 110.0.0, are derived from Verilog-A sources provided under the following license:

Software is distributed as is, completely without warranty or service support. The University of California and its employees are not liable for the condition or performance of the software.

The University owns the copyright but shall not be liable for any infringement of copyright or other proprietary rights brought by third

parties against the users of the software.

The University of California hereby disclaims all implied warranties.

The University of California grants the users the right to modify, copy, and redistribute the software and documentation, both within the user's organization and externally, subject to the following restrictions

1. The users agree not to charge for the University of California code itself but may charge for additions, extensions, or support.

2. In any product based on the software, the users agree to acknowledge the UC Berkeley BSIM Research Group that developed the software. This acknowledgment shall appear in the product documentation.

3. The users agree to obey all U.S. Government restrictions governing redistribution or export of the software.

4. The users agree to reproduce any copyright notice which appears on the software on any copy or modification of such made available to others.

Xyce's implementation of the MVS model, version 2.0.0, is derived from Verilog-A sources provided under the following license:

Copyright @ 2013 Massachusetts Institute of Technology (MIT)

The terms under which the software and associated documentation (the Software) is provided are as the following:

The Software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the Software or the use or other dealings in the Software.

MIT grants, free of charge, to any users the right to modify, copy, and redistribute the Software, both within the user's organization and externally, subject to the following restrictions:

1. The users agree not to charge for the MIT code itself but may charge for additions, extensions, or support.

2. In any product based on the Software, the users agree to

acknowledge the MIT VS Model Research Group that developed the software. This acknowledgment shall appear in the product documentation.

3. The users agree to obey all U.S. Government restrictions governing redistribution or export of the software.

4. The users agree to reproduce any copyright notice which appears on the software on any copy or modification of such made available to others.

Xyce's implementation of the PSP model, version 102.5.0, is derived from Verilog-A sources provided under the following license:

INTELLECTUAL PROPERTY NOTICE, DISCLAIMER AND LICENSE

The compact model software and documentation presented at this website form a whole that will henceforth be denoted as the "Model".

The Model presented at this website has been co-developed by NXP Semiconductors and Arizona State University until and including 2011. For this part of the Model, NXP Semiconductors claims undivided ownership and copyrights.

Since 2012 until today the Model has been co-developed by NXP Semiconductors and Delft University of Technology and for this part each claim undivided ownership and copyrights.

DISCLAIMER

The owners are fully free to further develop, adapt and extend the Model as they judge necessary or desirable.

The Model is distributed as is, completely without any express or implied warranty, or service support. The owners and their employees are not liable in any way for the condition or performance of the Model. The owners hereby disclaim all implied warranties.

LICENSE

NXP Semiconductors and Delft University of Technology hereby grant users a perpetual, irrevocable, worldwide, non-exclusive, royalty-free license with respect to Versions of the Model which have been released through this website http://psp.ewi.tudelft.nl.

NXP Semiconductors and Delft University of Technology grant the users the right to modify, copy and redistribute the Model, both within the user's organization and externally, subject to the following restrictions.

RESTRICTIONS

1. The users agree not to charge for the Model itself but may charge for additions, extensions, or support.

2. In any product based on the Model, the users agree to acknowledge the owners as developers of the Model. This acknowledgment shall appear in the product documentation.

3. The users agree to only use the name of CMC standard models to identify implementations of the CMC standard models which produce the same outputs as Standard code for the same inputs passing all CMC QA tests.

4. The users agree to obey all government restrictions governing redistribution or export of the software.

5. The users agree to reproduce any copyright notice which appears on the software and documentation on any copy or modification of such made available to others.

Xyce's implementations of the PSP model, version 103.4.0, and the JUNCAP diode, are derived from Verilog-A sources provided under the following license:

Software is distributed as is, completely without warranty or service support. The Commissariat a l'energie atomique et aux energies alternatives (CEA), NXP Semiconductors, and Delft University of Technology, along with their employees are not liable for the condition or performance of the software.

NXP Semiconductors, Delft University of Technology, and CEA own the copyright and grant users a perpetual, irrevocable, worldwide, non-exclusive, royalty-free license with respect to the software as set forth below.

NXP Semiconductors, Delft University of Technology, and CEA hereby disclaim all implied warranties.

NXP Semiconductors, Delft University of Technology, and CEA grant the users the right to modify, copy, and redistribute the software and documentation, both within the user's organization and externally, subject to the following restrictions:

1. The users agree not to charge for the NXP Semiconductors, Delft University of Technology, and CEA-developed code itself but may charge for additions, extensions, or support.

2. In any product based on the software, the users agree to acknowledge NXP Semiconductors, Delft University of Technology, and CEA that developed the software. This acknowledgment shall appear in the product documentation.

3. Redistributions to others of source code and documentation must retain the copyright notice, disclaimer, and list of conditions.

4. Redistributions to others in binary form must reproduce the copyright notice, disclaimer, and list of conditions in the documentation and/or other materials provided with the distribution.

Xyce's implementation of the EKV 3.0 model is derived from Verilog-A sources developed by the EKV Team of the Electronics Laboratory-TUC (Technical University of Crete). They are included in Xyce under license from Technical University of Crete. The official web site of the EKV model is `http://ekv.epfl.ch/`.

Due to licensing restrictions, the EKV MOSFETs are not available in open-source versions of Xyce. The license for EKV3 authorizes Sandia National Laboratories to distribute EKV3 only in binary versions of the code.

Index

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.